Genesting Manual de referencia

0.3

Generado por Doxygen 1.4.7

Sun Oct 29 11:19:29 2006

Índice general

1. Proyecto Genesting **1**

1.1. Introduction 1

1.2. Definicion 2

1.3. Objetivo 2

1.4. Caso de Estudio 2

2. Genesting Indice de módulos **3**

2.1. Genesting Módulos 3

3. Genesting Indice de clases **5**

3.1. Genesting Lista de componentes 5

4. Genesting Indice de archivos **7**

4.1. Genesting Lista de archivos 7

5. Genesting Página indice **9**

5.1. Genesting Páginas relacionadas 9

6. Genesting Documentación de módulos **11**

6.1. Distancias 11

6.2. Geometria 17

6.3. Genetico 41

7. Genesting Documentación de clases **55**

7.1. Referencia de la Estructura _genesting 55

7.2. Referencia de la Estructura _individuo 58
7.3. Referencia de la Estructura _line 60
7.4. Referencia de la Estructura _point 62
7.5. Referencia de la Estructura _polygon 63
7.6. Referencia de la Estructura _polygon_holes 64
7.7. Referencia de la Estructura _population 66
7.8. Referencia de la Estructura _posicion 68

8. Genesting Documentación de archivos **69**
8.1. Referencia del Archivo distance.c 69
8.2. Referencia del Archivo distance.h 71
8.3. Referencia del Archivo genesting.c 72
8.4. Referencia del Archivo genesting.h 73
8.5. Referencia del Archivo graphics.c 74
8.6. Referencia del Archivo graphics.h 77
8.7. Referencia del Archivo individuo.c 78
8.8. Referencia del Archivo individuo.h 79
8.9. Referencia del Archivo line.c 80
8.10. Referencia del Archivo line.h 81
8.11. Referencia del Archivo main.c 82
8.12. Referencia del Archivo point.c 85
8.13. Referencia del Archivo point.h 86
8.14. Referencia del Archivo polygon.c 87
8.15. Referencia del Archivo polygon.h 88
8.16. Referencia del Archivo polygon_holes.c 90
8.17. Referencia del Archivo polygon_holes.h 91
8.18. Referencia del Archivo population.c 93
8.19. Referencia del Archivo population.h 94

9. Genesting Documentación de páginas **95**
9.1. Listado de Tareas Pendientes 95
9.2. Lista de Bugs 96

Capítulo 1

Proyecto Genesting

Autor:

John Edgar Congote Calle

Versión:

0.3a1

Genesting es un proyecto que intenta resolver el problema de nesting o anidamiento de figuras a través del uso de algoritmos genéticos.

1.1. Introduction

In the manufacturing industry the raw materials usually came in finite two-dimensional sheets, where the permanent goal is the reduction of waste materials. But there is a frequent problem: how to distribute two-dimensional patterns in a container sheet in order to get the maximum utilization of material? This is know as the Knapsack problem.

Nowdays many company make this job according with the empirical experience of their employers having two risks. The first one is that there is not way to know if their solution are going to be the best to minimize the amount of waste materials. The second risk is in the case of presuming that an employ has the optimal solution the knowledge would be in the hands of just one person or group of work and not as a part of a system or as a part of the company.

1.2. Definicion

El problema de nesting se puede definir como encontrar una dispocision de patrones que esten dentro de otro de forma que se maximice el area utilizada. Este problema tiene muchas variantes, pero en el proyecto se trabajara especificamente sobre el caso de Knapsack.

1.3. Objetivo

Encontrar una dispocicion de patrones que maximicen el area utilizada dentro de otro patron. Los patrones estan definidos como poligonos simples, los cuales pueden ser por definicion convexos o concavos pero solo pueden tener un adentro o mas claramente sus lineas no se intersectan.

1.4. Caso de Estudio

Aunque el problema es teorico, el projecto quiere generar una aplicacion que pueda ser utilizada por la industria marroquinera, donde tienen que definir como distribuir los moldes de corte dentro de un cuero, en este caso el cuero se puede definir como el patron o poligono donde se tienen que colocar los demas patrones, y los moldes como los patrones internos.

Capítulo 2

Genesting Indice de módulos

2.1. Genesting Módulos

Lista de todos los módulos:

Distancias 11
Geometria 17
Genetico 41

Capítulo 3

Genesting Indice de clases

3.1. Genesting Lista de componentes

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

_genesting 55
_individuo 58
_line 60
_point 62
_polygon 63
_polygon_holes 64
_population 66
_posicion 68

Capítulo 4

Genesting Indice de archivos

4.1. Genesting Lista de archivos

Lista de todos los archivos con descripciones breves:

distance.c . . . 69
distance.h . . . 71
genesting.c . . . 72
genesting.h . . . 73
graphics.c . . . 74
graphics.h . . . 77
individuo.c . . . 78
individuo.h . . . 79
line.c . . . 80
line.h . . . 81
main.c . . . 82
point.c . . . 85
point.h . . . 86
polygon.c . . . 87
polygon.h . . . 88
polygon_holes.c . . . 90
polygon_holes.h . . . 91
population.c . . . 93
population.h . . . 94

Capítulo 5

Genesting Página indice

5.1. Genesting Páginas relacionadas

Lista de toda la documentación relacionada:

Listado de Tareas Pendientes 95
Lista de Bugs 96

Capítulo 6

Genesting Documentación de módulos

6.1. Distancias

6.1.1. Descripción detallada

Para el projecto utilizamos la definicion de distancia Euclidiana, donde la distancia entre dos objetos es la longitud de la recta mas cercana que los toca.

Archivos

- archivo distance.h
- archivo distance.c

Funciones

- float distance_pointpoint (point *a, point *b)
- float distance_pointline (point *f, line *l, int *seg)
- float distance_pointpolygon (point *f, polygon *p, line *ref)
- float distance_pointpolygonholes (point *f, polygon_holes *p, line *ref)

6.1.2. Documentación de las funciones

6.1.2.1. float distance_pointline (point * *f*, line * *l*, int * *seg*)

La funcion calcula la distancia entre un punto un segmento de linea, se pueden dar dos casos:

- La perpendicular del punto a la recta intersecta el segmento de recta, en esta caso la distancia es la longitud del punto al punto de interseccion de la recta con la perpendicular que pasa por el punto
- La perpendicular del punto a la recta no intersecta el segmente, en este caso la distancia entre el punto y la recta es igual a la minima distancia entre el punto y los puntos extremos del segmento de recta.

Parámetros:

←*f* Punto en 2 dimensiones

←*l* Segmento de recta que tiene definido los dos puntos extremos

→*seg* Indica cual fue la refencia para tomar la distancia puede tomar tres valores:0 cuando la distancia es respecto a la perpendicular de la linea, 1 cuando la distancia es respecto al punto extremo 1 y 2 cuando la distancia es respecto al punto extremo 2

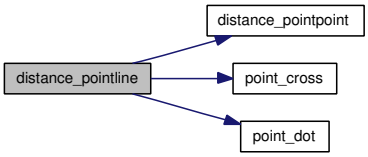
Devuelve:

La distancia entre el segmento de recta y el punto

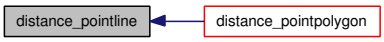
Definición en la línea 54 del archivo distance.c.

```
55 {
56     if(point_dot(&(l->v1),&(l->v2),f) > 0)
57     {
58         *seg = 2;
59         return distance_pointpoint(&(l->v2),f);
60     }
61
62     if(point_dot(&(l->v2),&(l->v1),f) > 0)
63     {
64         *seg =1;
65         return distance_pointpoint(&(l->v1),f);
66     }
67
68     *seg =0;
69     return fabsf(point_cross(&(l->v1),&(l->v2),f) / distance_pointpoint(&(l->v1),&(l->v2)));
70 }
```

Gráfico de llamadas para esta función:



Here is the caller graph for this function:



6.1.2.2. float distance_pointpoint (point * *a*, point * *b*)

La funcion compara la distancia entre dos puntos la formula para esto es: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Parámetros:

← ***a*** Punto en 2 dimensiones

← ***b*** Punto en 2 dimensiones

Devuelve:

La distancia entre los puntos

Definición en la línea 29 del archivo distance.c.

```
30 {
31     return sqrt(powf(a->x - b->x,2.0)+powf(a->y - b->y,2.0));
32 }
```

Here is the caller graph for this function:

6.1.2.3. float distance_pointpolygon (point * *f*, polygon * *p*, line * *ref*)

La funcion calcula la distancia entre un punto y el borde de un poligono simple, para esto calcula la distancia del punto contra todos los segmentos de linea que conforma el poligono y retorna la menor distancia, esta funcion no tiene consideracion si el punto esta adentro o afuera del poligono porque calcula la distancia al borde.

Parámetros:

←*f* Punto en 2 dimensiones

←*p* Poligono simple

→*ref* Escribe en la memoria el segmento de recta que define la menor distancia entre el poligono y el punto, si la distancia fue tomada respecto a un punto extremo, entonces el segmento de recta es un punto, el extremo.

Devuelve:

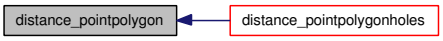
La distancia entre el punto y el poligono

Definición en la línea 87 del archivo distance.c.

```
88 {
89     int seg,i;
90     float min=FLT_MAX;
91 
92     for (i=0;i<p->nvertices;i++)
93     {
94         float d;
95         line l;
96 
97         l.v1=p->v[i];
98         l.v2=p->v[(i+1)%(p->nvertices)];
99         d=distance_pointline(f,&l,&seg);
100         if (i==0 || min>d)
101         {
102             min = d;
103             if (ref!=NULL)
104             {
105                 switch (seg)
106                 {
107                     case 0:
108                     ref->v1 = l.v1;
109                     ref->v2 = l.v2;
110                     break;
111                     case 1:
112                     ref->v1 = l.v1;
113                     ref->v2 = l.v1;
114                     break;
115                     case 2:
116                     ref->v1 = l.v2;
117                     ref->v2 = l.v2;
118                     break;
119                 }
120             }
121         }
122     }
123     return min;
124 }
```

Gráfico de llamadas para esta función:

Here is the caller graph for this function:



6.1.2.4. float distance_pointpolygonholes (point * *f*, polygon_holes * *p*, line * *ref*)

La funcion calcula la distancia entre un punto y un poligono con huecos, para esto calcula la distancia del punto con el borde del poligono y posteriormente con cada uno de los huecos de este, seleccionado la distancia minima

Parámetros:

← ***f*** Punto en 2 dimensiones

← ***p*** Poligono con huecos

→ ***ref*** Escribe en la memoria el segmento de recta que define la menor distancia entre el poligono y sus huecos con el punto, si la distancia fue tomada respecto a un punto extremo, entonces el segmento de recta es un punto, el extremo.

Devuelve:

La distancia entre un punto y un poligono con huecos

Definición en la línea 139 del archivo distance.c.

```
140 {
141     int i;
142     float min;
143 
144     min = distance_pointpolygon(f,p->p,ref);
145     for (i=0;i<p->nholes;i++)
146     {
147         float min2;
148         line ref2;
149         min2 = distance_pointpolygon(f,&(p->h[i]),&ref2);
150         if (min2 < min)
151         {
152             min = min2;
153             if (ref!=NULL)
154             {
155                 ref->v1 = ref2.v1;
156                 ref->v2 = ref2.v2;
157             }
```

```
158         }
159     }
160     return min;
161 }
```

Gráfico de llamadas para esta función:

Here is the caller graph for this function:

6.2. Geometria

6.2.1. Descripción detallada

Son los modelos y estructuras geometricas utilizadas para calcular las caracteristicas del problema, debido a que nesting esta fuertemente enlazado con poligonos, es necesario implementar el conjunto de objetos geometricos que lo manejen.

Archivos

- archivo line.h
- archivo point.h
- archivo polygon.h
- archivo polygon_holes.h
- archivo line.c
- archivo point.c
- archivo polygon.c
- archivo polygon_holes.c

Clases

- struct _line
- struct _point
- struct _polygon
- struct _polygon_holes

Tipos definidos

- typedef _line line
- typedef _point point
- typedef _polygon polygon
- typedef _polygon_holes polygon_holes

Funciones

- bool line_intersection (line *l1, line *l2)
- bool line_equal (line *l1, line *l2)
- bool line_ispoint (line *l1)
- float point_dot (point *a, point *b, point *c)
- float point_cross (point *a, point *b, point *c)
- float polygon_area (polygon *p)

- bool polygon_pointin (polygon *p, point *f)
- bool polygon_overlapping (polygon *p, polygon *q)
- void polygon_rotate (polygon *p, float t)
- void polygon_minbox (polygon *p, float *minx, float *miny, float *maxx, float *maxy)
- void polygon_translate (polygon *p, float x, float y)
- float polygonholes_area (polygon_holes *p)
- float polygonholes_volumen (polygon_holes *p)
- bool polygonholes_pointin (polygon_holes *p, point *f)
- bool polygonholes_polygonin (polygon_holes *p, polygon *q)
- bool polygonholes_pointinhole (polygon_holes *p, point *f)
- point polygon_center (polygon *p)

6.2.2. Documentación de los tipos definidos

6.2.2.1. typedef struct _line line

Definición en la línea 21 del archivo line.h.

6.2.2.2. typedef struct _point point

Definición en la línea 17 del archivo point.h.

6.2.2.3. typedef struct _polygon polygon

Definición en la línea 24 del archivo polygon.h.

6.2.2.4. typedef struct _polygon_holes polygon_holes

Definición en la línea 24 del archivo polygon_holes.h.

6.2.3. Documentación de las funciones

6.2.3.1. bool line_equal (line * *l1*, line * *l2*)

Compara dos segmentos de linea, en este caso el orden de los extremos importa en la operacion

Parámetros:

← ***l1*** Linea en 2 dimensiones

← ***l2*** Linea en 2 dimensiones

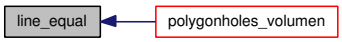
Devuelve:

Verdadero si los segmentos de recta son iguales, falso en caso contrario

Definición en la línea 109 del archivo line.c.

```
110 {
111     return (l1->v1.x == l2->v1.x &&
112             l1->v1.y == l2->v1.y &&
113             l1->v2.x == l2->v2.x &&
114             l1->v2.y == l2->v2.y
115            ) ? true : false;
116 }
```

Here is the caller graph for this function:



6.2.3.2. bool line_intersection (line ∗ *l1*, line ∗ *l2*)

Identifica si dos lineas se intersectan, el algoritmo utilizado fue sacado del libro de Graphics Gems del Articulo Faster Line Segment Intersection

Parámetros:

← ***l1*** Linea en 2 dimensiones

← ***l2*** Linea en 2 dimensiones

Devuelve:

Un valor booleano que es verdadero si las lineas se intersectan y falso en caso contrario

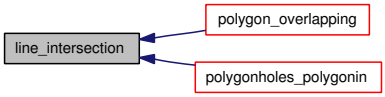
Definición en la línea 22 del archivo line.c.

```
23 {
24     float x1,y1,x2,y2,x3,y3,x4,y4;
25     float Ax,Bx,Cx,Ay,By,Cy;
26     float x1lo,x1hi,y1lo,y1hi;
27     float d,e,f;
28
29     x1 = l1->v1.x;
30     y1 = l1->v1.y;
31
```

```
32      x2 = l1->v2.x;
33      y2 = l1->v2.y;
34
35      x3 = l2->v1.x;
36      y3 = l2->v1.y;
37
38      x4 = l2->v2.x;
39      y4 = l2->v2.y;
40
41      Ax = x2 - x1;
42      Bx = x3 - x4;
43
44      if (Ax < 0){
45          x1lo=x2;
46          x1hi=x1;
47      } else {
48          x1hi=x2;
49          x1lo=x1;
50      }
51
52      if (Bx > 0){
53              if (x1hi < x4 || x2 < x1lo) return false;
54      } else {
55              if (x1hi < x3 || x4 < x1lo) return false;
56      }
57
58      Ay = y2 - y1;
59      By = y3 - y4;
60
61      if (Ay < 0){
62              y1lo = y2;
63              y1hi = y1;
64      } else {
65              y1hi = y2;
66              y1lo = y1;
67      }
68
69      if (By > 0){
70              if (y1hi < y4 || y3 < y1lo) return false;
71      } else {
72              if (y1hi < y3 || y4 < y1lo) return false;
73      }
74
75      Cx = x1 - x3;
76      Cy = y1 - y3;
77
78      d = By*Cx - Bx*Cy;
79      f = Ay*Bx - Ax*By;
80
81      if (f>0){
82              if (d<0 || d>f) return false;
83      } else {
84              if (d>0 || d<f) return false;
85      }
86
87      e = Ax*Cy - Ay*Cx;
88
```

```
89     if (f>0){
90             if (e<0 || e>f) return false;
91     } else {
92             if (e>0 || e<f) return false;
93     }
94
95     if (f==0) return false;
96
97     return true;
98 }
```

Here is the caller graph for this function:



6.2.3.3. bool line_ispoint (line * *l*)

Identifica si un segmento de recta es en realidad un unico punto, esto lo hace comprobando los puntos extremos del segmento de recta, si los extremos son iguales entonces se puede definir este segmento de recta como una linea

Parámetros:

← ***l*** Linea en 2 dimensiones

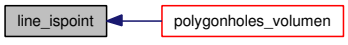
Devuelve:

Verdadero si la linea es un punto, falso en caso contrario.

Definición en la línea 128 del archivo line.c.

```
129 {
130     return (l->v1.x == l->v2.x && l->v1.y == l->v2.y) ? true : false;
131 }
```

Here is the caller graph for this function:



6.2.3.4. float point_cross (point * *a*, point * *b*, point * *c*)

Calcula el producto cruz entro los punto a, b y c. Aunque esta funcion matematicamente esta definida en los vectores y no en los puntos, podemos relacionar un punto como el vector que hay desde el origen del plano hasta las coordenadas de este y asi calculamos el producto cruz.

Parámetros:

← ***a*** Punto en 2 dimensiones

← ***b*** Punto en 2 dimensiones

← ***c*** Punto en 2 dimensiones

Devuelve:

El producto cruz

Definición en la línea 38 del archivo point.c.

```
39 {
40     return ((b->x - a->x)*(c->y - a->y)) - ((b->y - a->y)*(c->x - a->x));
41 }
```

Here is the caller graph for this function:

6.2.3.5. float point_dot (point * *a*, point * *b*, point * *c*)

Calcula el producto punto entre los puntos a, b y c. Aunque esta funcion matematicamente esta definida en los vectores y no en los puntos, podemos relacionar un punto como el vector que hay desde el origen del plano hasta las coordenadas de este y asi calculamos el producto punto.

Parámetros:

← ***a*** Punto en 2 dimensiones

← ***b*** Punto en 2 dimensiones

← ***c*** Punto en 2 dimensiones

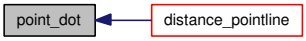
Devuelve:

El producto punto

Definición en la línea 21 del archivo point.c.

```
22 {
23     return ((b->x - a->x) * (c->x - b->x)) + ((b->y - a->y) * (c->y - b->y));
24 }
```

Here is the caller graph for this function:



6.2.3.6. float polygon_area (polygon * p)

Esta funcion calcula el area de un poligono simple, el algoritmo fue tomado de Graphic Gems II, del articulo The Area of a Simple Polygon.

Parámetros:

← ***p*** Poligono simple

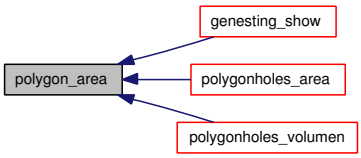
Devuelve:

El area del poligono

Definición en la línea 20 del archivo polygon.c.

```
21 {
22     unsigned int i;
23     float area = 0;
24     for (i=0;i<p->nvertices;i++)
25     {
26         area += p->v[i].x * p->v[(i + 1) % p->nvertices].y;
27         area -= p->v[i].y * p->v[(i + 1) % p->nvertices].x;
28     }
29     area /= 2;
30 
31     return(area < 0 ? -area : area);
32 }
```

Here is the caller graph for this function:



6.2.3.7. point polygon_center (polygon * *p*)

Definición en la línea 137 del archivo polygon.c.

```
138 {
139     int i=0;
140     float minx, maxx, miny, maxy;
141
142     point r;
143
144     polygon_minbox(p, &minx, &miny, &maxx, &maxy)
145
146     r.x = ((maxx - minx)/2)+minx;
147     r.y = ((maxy - miny)/2)+miny;
148
149     return r;
150 }
```

Gráfico de llamadas para esta función:

Here is the caller graph for this function:

6.2.3.8. void polygon_minbox (polygon * *p*, float * *minx*, float * *miny*, float * *maxx*, float * *maxy*)

La funcion calcula las coordenadas mas extremas del poligono, conformando con estas 2 puntos que pueden formar un rectangulo que contiene completamente el poligono

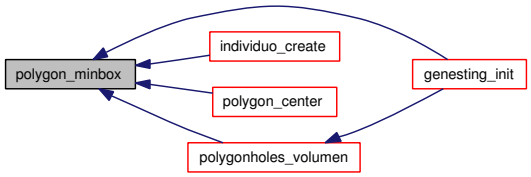
Parámetros:

- ← ***p*** Poligono simple
- → ***minx*** Coordenada mas pequeña del poligono en el eje x
- → ***miny*** Coordenada mas pequeña del poligono en el eje y
- → ***maxx*** Coordenada mas grande del poligono en el eje x
- → ***maxy*** Coordenada mas grande del poligono en el eje y

Definición en la línea 163 del archivo polygon.c.

```
164 {
165     int i;
166 
167     *minx = p->v[0].x;
168     *miny = p->v[0].y;
169     *maxx = p->v[0].x;
170     *maxy = p->v[0].y;
171 
172     for (i=1;i<p->nvertices;i++)
173     {
174         if (*minx > p->v[i].x)
175         {
176             *minx = p->v[i].x;
177         }
178         if (*miny > p->v[i].y)
179         {
180             *miny = p->v[i].y;
181         }
182         if (*maxx < p->v[i].x)
183         {
184             *maxx = p->v[i].x;
185         }
186         if (*maxy < p->v[i].y)
187         {
188             *maxy = p->v[i].y;
189         }
190     }
191 }
```

Here is the caller graph for this function:



6.2.3.9. bool polygon_overlapping (polygon * *p*, polygon * *q*)

Esta funcion identifica si dos poligonos se solapan de alguna forma para verificar esto se realizan 3 comprobaciones:

- Ningun vertice del poligono p esta dentro del poligono q
- Ningun vertice del poligono q esta dentro del poligono p
- Ninguna linea de los poligonos p y q se intersectan

Parámetros:

← ***p*** Poligono Simple en 2 dimensiones

← ***q*** Poligono Simple en 2 dimensiones

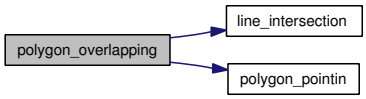
Devuelve:

Verdadero si los poligonos se solapan, falso en caso contrario.

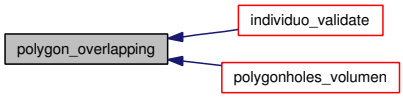
Definición en la línea 74 del archivo polygon.c.

```
75 {
76     int i,j;
77
78     for (i=0;i<p->nvertices;i++)
79     {
80         if (polygon_pointin(q,&p->v[i]))
81             return true;
82     }
83
84     for (i=0;i<q->nvertices;i++)
85     {
86         if (polygon_pointin(p,&q->v[i]))
87             return true;
88     }
89
90     for (i=0;i<p->nvertices;i++)
91     {
92         line t1,t2;
93         t1.v1=p->v[i];
94         t1.v2=p->v[(i+1)%(p->nvertices)];
95         for (j=0;j<q->nvertices;j++)
96         {
97             t2.v1=q->v[j];
98             t2.v2=q->v[(j+1)%q->nvertices];
99             if (line_intersection(&t1,&t2))
100                 return true;
101         }
102     }
103     return false;
104 }
```

Gráfico de llamadas para esta función:



Here is the caller graph for this function:



6.2.3.10. bool polygon_pointin (polygon * *p*, point * *f*)

La funcion identifica si un punto esta dentro de un poligono o no.

Parámetros:

← ***p*** Poligono simple en 2 dimensiones

← ***f*** Punto en 2 dimensiones

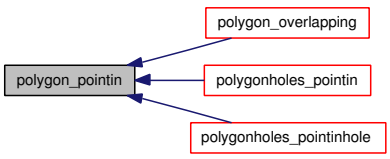
Devuelve:

Un booleano que es verdadero en caso de que el punto este dentro del poligono y falso en caso contrario

Definición en la línea 48 del archivo polygon.c.

```
49 {
50     bool c=false;
51     unsigned int i,j;
52 
53     for (i=0,j=p->nvertices-1;i<p->nvertices;j=i++)
54     {
55         if ((((p->v[i].y<=f->y) && (f->y<p->v[j].y)) ||
56                 ((p->v[j].y<=f->y) && (f->y<p->v[i].y))) &&
57                 (f->x < (p->v[j].x - p->v[i].x) * (f->y - p->v[i].y) / (p->v[j].y - p->v[i].y) + p->v[i
58             c = !c;
59     }
60     return c;
61 }
```

Here is the caller graph for this function:



6.2.3.11. void polygon_rotate (polygon * *p*, float *t*)

La funcion rota el poligono p un angulo t, teniendo como eje de rotacion el centro de la caja mas pequena que contiene el poligono. Aunque se pueden elegir otros puntos como eje, o de hecho implementar un eje arbitrario, el centro de la caja que contiene el poligono es un punto facil de calcular y ademas que evita exageradas translaciones relativas del poligono al ser rotado.

Parámetros:

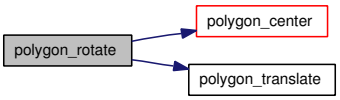
↔ ***p*** Poligono simple

← ***t*** Angulo en radianes a rotar la figura en sentido antihorario.

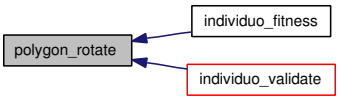
Definición en la línea 116 del archivo polygon.c.

```
117 {
118     int i;
119     point pc;
120 
121     float sent = sin(t);
122     float cost = cos(t);
123 
124     pc = polygon_center(p);
125 
126     polygon_translate(p, -pc.x, -pc.y);
127 
128     for (i=0;i<p->nvertices;i++)
129     {
130         p->v[i].x=(p->v[i].x*cost - p->v[i].y * sent);
131         p->v[i].y=(p->v[i].y*cost + p->v[i].x * sent);
132     }
133 
134     polygon_translate(p, pc.x, pc.y);
135 }
```

Gráfico de llamadas para esta función:



Here is the caller graph for this function:



6.2.3.12. void polygon_translate (polygon * p, float x, float y)

Translada el poligono una distancia definida por x,y.

Parámetros:

↔ ***p*** Poligono simple

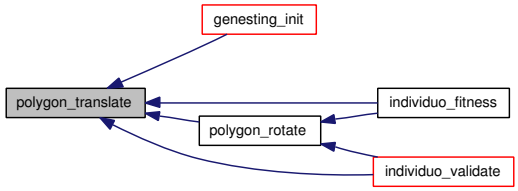
← ***x*** Distancia en x a transladar el poligono

← ***y*** Distancia en y a transladar el poligono

Definición en la línea 200 del archivo polygon.c.

```
201 {
202     int i;
203     for (i=0;i<p->nvertices;i++)
204     {
205         p->v[i].x+=x;
206         p->v[i].y+=y;
207     }
208 }
```

Here is the caller graph for this function:



6.2.3.13. float polygonholes_area (polygon_holes * p)

La funcion calcula el area de un Poligono con huecos, para hacer esto calculamos el area del poligono formado por el borde exterior y le restamos el area de cada uno de los huecos.

Parámetros:

← ***p*** Poligono simple en 2 dimensiones

Devuelve:

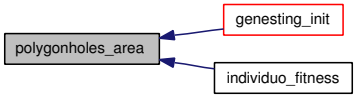
Area del poligono simple con huecos

Definición en la línea 30 del archivo polygon_holes.c.

```
31 {
32     unsigned int i;
33     float area;
34     area = polygon_area(p->p);
35     for (i=0;i<p->nholes;i++)
36     {
37         area -= polygon_area(&(p->h[i]));
38     }
39     return area;
40 }
```

Gráfico de llamadas para esta función:

Here is the caller graph for this function:



6.2.3.14. bool polygonholes_pointin (polygon_holes ∗ *p*, point ∗ *f*)

Esta funcion identifica si un punto esta dentro de un poligono con huecos por lo tanto evalua inicialmente que el punto este dentro del poligono exterior y posterioremente que no se encuentre dentro de los huecos

Parámetros:

← ***p*** Poligono simple con huecos en 2 dimensiones

← ***f*** Punto en 2 dimensiones

Devuelve:

Verdadero en caso que el punto este dentro del poligono, falso en caso contrario

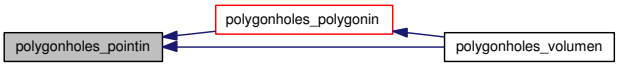
Definición en la línea 345 del archivo polygon_holes.c.

```
346 {
347     bool valido=false;
348     if (polygon_pointin(p->p, f))
```

```
349     {
350         unsigned int i;
351         valido=true;
352 
353         for (i=0;i<p->nholes && valido;i++)
354         {
355             if (polygon_pointin(&(p->h[i]), f))
356             {
357                 valido=false;
358             }
359         }
360     }
361     return valido;
362 }
```

Gráfico de llamadas para esta función:

Here is the caller graph for this function:



6.2.3.15. bool polygonholes_pointinhole (polygon_holes * *p*, point * *f*)

Identifica si un punto esta dentro de un hueco del poligono.

Parámetros:

← ***p*** Poligono Simple con huecos

← ***f*** Punto en 2 dimensiones

Devuelve:

Verdadero si el punto esta dentro de uno de los huecos, falso en caso contrario

Definición en la línea 373 del archivo polygon_holes.c.

```
374 {
375     bool valido=false;
376     unsigned int i;
377 
```

```
378     for (i=0;i<p->nholes && !valido;i++)
379     {
380         if (polygon_pointin(&(p->h[i]), f))
381         {
382             valido=true;
383         }
384     }
385     return valido;
386 }
```

Gráfico de llamadas para esta función:

Here is the caller graph for this function:

6.2.3.16. bool polygonholes_polygonin (polygon_holes * *p*, polygon * *q*)

La funcion evalua que el poligono q esta completamente adentro del poligono p entonces todos los puntos de q deben estar dentro de p y no se deben intersectar las lineas que los conforman

Parámetros:

← ***p*** Poligono Simple con huecos

← ***q*** Poligono Simple

Devuelve:

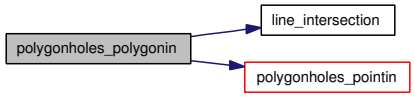
Verdadero en caso del que poligono q este dentro del poligono p, falso en caso contrario

Definición en la línea 399 del archivo polygon_holes.c.

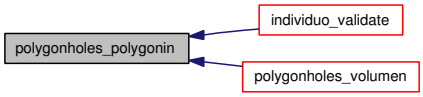
```
400 {
401     int i,j,k;
402     bool adentro = true;
403 
404     for (i=0;i<q->nvertices && adentro;i++)
405     {
406         line t1,t2;
407         t1.v1=q->v[i];
```

```
408         t1.v2=q->v[(i+1)%(q->nvertices)];
409
410         if (!polygonholes_pointin(p,&q->v[i]))
411             adentro = false;
412
413         if (!polygonholes_pointin(p,&q->v[(i+1)%(q->nvertices)]))
414             adentro = false;
415
416         for (j=0;j<p->p->nvertices && adentro;j++)
417         {
418             t2.v1=p->p->v[j];
419             t2.v2=p->p->v[(j+1)%p->p->nvertices];
420             if (line_intersection(&t1,&t2))
421                 adentro = false;
422         }
423
424         for (j=0;j<p->nholes && adentro; j++)
425         {
426             for (k=0;k<p->h[j].nvertices && adentro; k++)
427             {
428                 t2.v1 = p->h[j].v[k];
429                 t2.v2 = p->h[j].v[(k+1)%p->h[j].nvertices];
430                 if (line_intersection(&t1,&t2))
431                     adentro = false;
432             }
433         }
434     }
435     return adentro;
436 }
```

Gráfico de llamadas para esta función:



Here is the caller graph for this function:



6.2.3.17. float polygonholes_volumen (polygon_holes * *p*)

Calcula el volumen generado por el poligono con huecos, definiendo asi el poligono como una figura tridimensional de la siguiente manera, la base del solido esta conformada por el poligono con huecos y la altura del solido en cada punto es igual a la distancia del punto a uno de los bordes.

El metodo para calcular el volumen se basa en encerrar el poligono en un rectangulo que lo contenga, y sucesivamente partir el poligono en dos subrectangulos respecto al lado mas largo calculando el volumen de cada uno de los subrectangulos y sumando el valor y esto se hace de manera recursiva.

- Cuando el subrectangulo esta completamente fuera del poligono no calculamos el volumen y regresamos un valor de 0
- Cuando el subrectangulo tiene un area muy pequena definida por un da se calcula el centro del subrectangulo y se verifica que este dentro del poligono, si este es el caso, entonces se devuelve el valor del area multiplicada por la distancia del punto medio al borde, si el punto medio es 0 se devuelve 0.
- Cuando el subrectangulo esta dentro del poligono y los vertices de este tienen su menor distancia respecto a un mismo segmento de recta se calcula el volumen del poliedro formado como el promedio de las distancias de los vertices multiplicados por el area del subrectangulo.
- Cuando el subrectangulo esta dentro del poligono y los vertices de este tienen su menor distancia respecto a un mismo punto, que se da en el caso de los poligonos concavos, entonces se calcula el volumen como el volumen formado por la interseccion entre el cono que genera el punto y el subrectangulo, la formula que da este volumen es: $\int_a^b \int_c^d \sqrt{((x-h)^2+(y-k)^2)}dxdy$ donde (h,k) son las coordenadas del punto con respecto al que se toma la distancia, (a,b) son los valores en que se desplaza y, (c,d) son los valores en que se desplaza x.

Esta funcion evalua menos puntos si en ves de dividir el rectangulo en 2 segun su lado mas largo, se divide en 4 partes iguales, pero esta aproximacion puede tener problemas cuando la proporcion del los lados del subrectangulo es muy grande.

Parámetros:

$\leftarrow$ ***p*** Poligono simple con huecos en 2 dimensiones

Devuelve:

Volumen generado

Bug

Esta funcion para calcular el volument de un rectangulo que intersecta un cono tiene valores indeterminados en algunos segmentos del problema, es posible que este error aparesca solo en algunos casos, pero hay que examinar mas la funcion.

Definición en la línea 86 del archivo polygon_holes.c.

```
87 {
88 #if graphics
89     void draw_polygon(polygon *p)
90     {
91         int i;
92         for (i=0;i<p->nvertices;i++)
93         {
94             draw_line(p->v[i].x,p->v[i].y,p->v[(i+1)%p->nvertices].x,p->v[(i+1)%p->nvertices].y);
95         }
96     }
97 #endif
98
99     float polygonholes_volumen_box( polygon_holes *p,
100                                     float minx,
101                                     float miny,
102                                     float maxx,
103                                     float maxy)
104     {
105
106         float vol=0;
107         float da;
108
109         polygon rec;
110         point q[4];
111
112         q[0].x = minx;
113         q[0].y = miny;
114
115         q[1].x = maxx;
116         q[1].y = miny;
117
118         q[2].x = maxx;
119         q[2].y = maxy;
120
121         q[3].x = minx;
122         q[3].y = maxy;
123
124         rec.v =(point *) &q;
125         rec.nvertices = 4;
126
127         da = polygon_area(&rec);
128
129
130         if (!polygon_overlapping(p->p, &rec)) //OK
131         {
132             vol = 0;
133 #if graphics
134
135             draw_rect(minx, miny, maxx, maxy,255,255,255);
136 #endif
137
138         }
139         else if (da < DELTA) // OK
140         {
141             point pm;
142
143             pm.x=minx+((maxx-minx)/2.0);
```

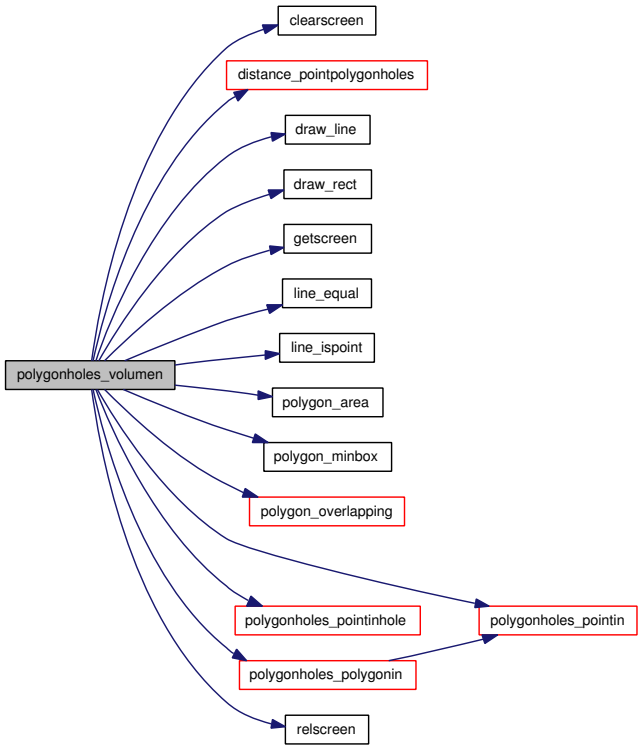
```
144             pm.y=miny+((maxy-miny)/2.0);
145
146             if (polygonholes_pointin(p, &pm))
147             {
148                 vol = da * distance_pointpolygonholes(&pm, p,NULL);
149 #if graphics
150
151                 draw_rect(minx, miny, maxx, maxy,128,128,128);
152 #endif
153
154             }
155             else
156             {
157                 vol = 0;
158 #if graphics
159
160                 draw_rect(minx, miny, maxx, maxy,225,255,255);
161 #endif
162
163             }
164         }
165         else
166         {
167
168             int i;
169             float dis[4];
170             line ref[4];
171             for (i=0;i<4;i++)
172             {
173                 dis[i]=distance_pointpolygonholes(&q[i], p,&ref[i]);
174             }
175
176             if( polygonholes_polygonin(p, &rec)
177                     &&
178                     (line_equal(&ref[0],&ref[1]) &&
179                      line_equal(&ref[1],&ref[2]) &&
180                      line_equal(&ref[2],&ref[3]))
181
182               ) // OK
183             {
184                 if (line_ispoint(&ref[0]))// Si es un punto entonces se utiliza la formula
185                 {
186                     double a,b,c,d,h,k;
187                     c=minx;
188                     d=maxx;
189                     a=miny;
190                     b=maxy;
191                     h=ref[0].v1.x;
192                     k=ref[0].v1.y;
193                     /*!\bug
194                     Esta funcion para calcular el volument de un rectangulo que intersecta
195                     cono tiene valores indeterminados en algunos segmentos del problema, es
196                     posible que este error aparesca solo en algunos casos, pero hay que exa
197                     mas la funcion.
198                     */
199
200                     vol = (2*a*c*sqrt(pow(-c + h,2) + pow(a - k,2))
```

```
201                             - 2*a*h*sqrt(pow(-c + h,2) + pow(a - k,2))
202                             - 2*a*d*sqrt(pow(-d + h,2) + pow(a - k,2))
203                             + 2*a*h*sqrt(pow(-d + h,2) + pow(a - k,2))
204                             - 2*b*c*sqrt(pow(-c + h,2) + pow(b - k,2))
205                             + 2*b*h*sqrt(pow(-c + h,2) + pow(b - k,2))
206                             + 2*b*d*sqrt(pow(-d + h,2) + pow(b - k,2))
207                             - 2*b*h*sqrt(pow(-d + h,2) + pow(b - k,2))
208                             - 2*c*k*sqrt(pow(-c + h,2) + pow(a - k,2))
209                             + 2*h*k*sqrt(pow(-c + h,2) + pow(a - k,2))
210                             + 2*d*k*sqrt(pow(-d + h,2) + pow(a - k,2))
211                             - 2*h*k*sqrt(pow(-d + h,2) + pow(a - k,2))
212                             + 2*c*k*sqrt(pow(-c + h,2) + pow(b - k,2))
213                             - 2*h*k*sqrt(pow(-c + h,2) + pow(b - k,2))
214                             - 2*d*k*sqrt(pow(-d + h,2) + pow(b - k,2))
215                             + 2*h*k*sqrt(pow(-d + h,2) + pow(b - k,2))
216                             - pow(a,3)*log(-c + h + sqrt(pow(-c + h,2) + pow(a - k,2)))
217                             + 3*pow(a,2)*k*log(-c + h + sqrt(pow(-c + h,2) + pow(a - k,2)))
218                             - 3*a*pow(k,2)*log(-c + h + sqrt(pow(-c + h,2) + pow(a - k,2)))
219                             + pow(k,3)*log(-c + h + sqrt(pow(-c + h,2) + pow(a - k,2)))
220                             + pow(a,3)*log(-d + h + sqrt(pow(-d + h,2) + pow(a - k,2)))
221                             - 3*pow(a,2)*k*log(-d + h + sqrt(pow(-d + h,2) + pow(a - k,2)))
222                             + 3*a*pow(k,2)*log(-d + h + sqrt(pow(-d + h,2) + pow(a - k,2)))
223                             - pow(k,3)*log(-d + h + sqrt(pow(-d + h,2) + pow(a - k,2)))
224                             + pow(b,3)*log(-c + h + sqrt(pow(-c + h,2) + pow(b - k,2)))
225                             - 3*pow(b,2)*k*log(-c + h + sqrt(pow(-c + h,2) + pow(b - k,2)))
226                             + 3*b*pow(k,2)*log(-c + h + sqrt(pow(-c + h,2) + pow(b - k,2)))
227                             - pow(k,3)*log(-c + h + sqrt(pow(-c + h,2) + pow(b - k,2)))
228                             - pow(b,3)*log(-d + h + sqrt(pow(-d + h,2) + pow(b - k,2)))
229                             + 3*pow(b,2)*k*log(-d + h + sqrt(pow(-d + h,2) + pow(b - k,2)))
230                             - 3*b*pow(k,2)*log(-d + h + sqrt(pow(-d + h,2) + pow(b - k,2)))
231                             + pow(k,3)*log(-d + h + sqrt(pow(-d + h,2) + pow(b - k,2)))
232                             + pow(c,3)*log(a + sqrt(pow(-c + h,2) + pow(a - k,2)) - k)
233                             - 3*pow(c,2)*h*log(a + sqrt(pow(-c + h,2) + pow(a - k,2)) - k)
234                             + 3*c*pow(h,2)*log(a + sqrt(pow(-c + h,2) + pow(a - k,2)) - k)
235                             - pow(h,3)*log(a + sqrt(pow(-c + h,2) + pow(a - k,2)) - k)
236                             - pow(d,3)*log(a + sqrt(pow(-d + h,2) + pow(a - k,2)) - k)
237                             + 3*pow(d,2)*h*log(a + sqrt(pow(-d + h,2) + pow(a - k,2)) - k)
238                             - 3*d*pow(h,2)*log(a + sqrt(pow(-d + h,2) + pow(a - k,2)) - k)
239                             + pow(h,3)*log(a + sqrt(pow(-d + h,2) + pow(a - k,2)) - k)
240                             - pow(c,3)*log(b + sqrt(pow(-c + h,2) + pow(b - k,2)) - k)
241                             + 3*pow(c,2)*h*log(b + sqrt(pow(-c + h,2) + pow(b - k,2)) - k)
242                             - 3*c*pow(h,2)*log(b + sqrt(pow(-c + h,2) + pow(b - k,2)) - k)
243                             + pow(h,3)*log(b + sqrt(pow(-c + h,2) + pow(b - k,2)) - k)
244                             + pow(d,3)*log(b + sqrt(pow(-d + h,2) + pow(b - k,2)) - k)
245                             - 3*pow(d,2)*h*log(b + sqrt(pow(-d + h,2) + pow(b - k,2)) - k)
246                             + 3*d*pow(h,2)*log(b + sqrt(pow(-d + h,2) + pow(b - k,2)) - k)
247                             - pow(h,3)*log(b + sqrt(pow(-d + h,2) + pow(b - k,2)) - k))/6;
248 #if graphics
249
250                     draw_rect(minx, miny, maxx, maxy,0,128,0);
251 #endif
252
253                 }
254                 else // Si es una linea se utiliza la formula de la cuna
255                 {
256                     vol = (dis[0]+dis[1]+dis[2]+dis[3])/4.0 * da;
257 #if graphics
```

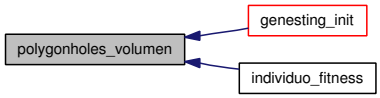
```
258
259                     draw_rect(minx, miny, maxx, maxy,0,0,128);
260 #endif
261
262                 }
263             }
264             else
265             {
266                 if (!(
267                             polygonholes_pointinhole(p, &q[0]) &&
268                             polygonholes_pointinhole(p, &q[1]) &&
269                             polygonholes_pointinhole(p, &q[2]) &&
270                             polygonholes_pointinhole(p, &q[3])
271                         )) // el poligono esta dentro de p pero no en un hueco
272                 {
273                     //Opcion de corte 1
274                     /*
275                                         vol+= polygonholes_volumen_box(p, minx, miny, (minx
276                                         vol+= polygonholes_volumen_box(p, (minx+maxx)/2.0,
277                                         vol+= polygonholes_volumen_box(p, minx, (miny+maxy)
278                                         vol+= polygonholes_volumen_box(p, (minx+maxx)/2.0,
279                     */
280
281                     if (maxx-minx > maxy-miny)
282                     {
283                         vol+= polygonholes_volumen_box(p, minx, miny, (minx+maxx)/2.0, maxy
284                         vol+= polygonholes_volumen_box(p, (minx+maxx)/2.0, miny, maxx, maxy
285                     }
286                     else
287                     {
288                         vol+= polygonholes_volumen_box(p, minx, miny, maxx, (miny+maxy)/2.0
289                         vol+= polygonholes_volumen_box(p, minx, (miny+maxy)/2.0, maxx, maxy
290                     }
291
292                 }
293 #if graphics
294                 else
295                 {
296                     draw_rect(minx, miny, maxx, maxy,0,0,255);
297                 }
298 #endif
299
300             }
301         }
302         if (isnan(vol))
303         {
304 #if graphics
305             draw_rect(minx, miny, maxx, maxy,255,0,0);
306 #endif
307
308             fprintf(stderr,"Error: (%f, %f) (%f, %f)\n",minx,miny,maxx,maxy);
309             vol=0;
310         }
311         return vol;
312     }
313
314     float minx, miny, maxx, maxy;
```

```
315     polygon_minbox(p->p, &minx, &miny, &maxx, &maxy);
316
317 #if graphics
318     getscreen();
319     clearscreen();
320     draw_polygon(p->p);
321
322     {
323         int j;
324         for (j=0;j<p->nholes;j++)
325         {
326             draw_polygon(&(p->h[j]));
327         }
328     }
329     relscreen();
330 #endif
331
332     return polygonholes_volumen_box(p,minx,miny,maxx,maxy);
333 }
```

Gráfico de llamadas para esta función:



Here is the caller graph for this function:


genesting_init
polygonholes_volumen
individuo_fitness

6.3. Genetico

6.3.1. Descripción detallada

Dentro del modulo genetico encontraremos una implementacion de heuristicas de algoritmos geneticos y evoluticos que seleccionara la solucion.

Archivos

- archivo genesting.h
- archivo individuo.h
- archivo population.h
- archivo genesting.c
- archivo individuo.c
- archivo population.c

Clases

- struct _genesting
- struct _posicion
- struct _individuo
- struct _population

Tipos definidos

- typedef _genesting genesting
- typedef _posicion posicion
- typedef _individuo individuo
- typedef _population population

Funciones

- genesting * leer_archivo (char *arc_name)
- void genesting_init (genesting *g)
- void genesting_show (genesting *g)
- float individuo_fitness (individuo *ind)
- void individuo_create (genesting *g, individuo *ind)
- void individuo_mutate (individuo *ind)
- void individuo_procreate (individuo *p, individuo *m, individuo *ind)
- bool individuo_validate (individuo *ind)
- int comparar_individuos (individuo *ind1, individuo *ind2)

- void population_create (population *p, genesting *g, int n)
- void population_evaluate (population *p)
- void population_generation (population *p)

6.3.2. Documentación de los tipos definidos

6.3.2.1. typedef struct _genesting genesting

Definición en la línea 25 del archivo genesting.h.

6.3.2.2. typedef struct _individuo individuo

Definición en la línea 19 del archivo individuo.h.

6.3.2.3. typedef struct _population population

Definición en la línea 21 del archivo population.h.

6.3.2.4. typedef struct _posicion posicion

Definición en la línea 17 del archivo individuo.h.

6.3.3. Documentación de las funciones

6.3.3.1. int comparar_individuos (individuo * *ind1*, individuo * *ind2*)

Compara entre los dos individuos cual es mejor segun su fitness

Definición en la línea 275 del archivo individuo.c.

```
276 {
277     return ((int)(((ind1->fitness)-(ind2->fitness))*10000.0));
278 }
```

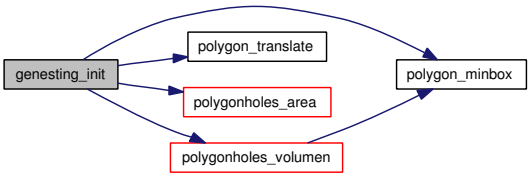
6.3.3.2. void genesting_init (genesting * *g*)

Realiza unas correcciones iniciales al archivo de entrada y calcula los valores fijos del problema, como el area maxima y volumen maximo que se dan cuando generamos una solucion vacia.

Definición en la línea 87 del archivo genesting.c.

```
88 {
89     int i;
90     float minx, miny, maxx, maxy;
91     polygon_holes p;
92 
93     polygon_minbox(&(g->plantilla), &minx, &miny, &maxx, &maxy);
94 
95     polygon_translate(&(g->plantilla), -minx, -miny);
96 
97     for (i=0;i<g->nhuecos;i++)
98     {
99         polygon_translate(&(g->huecos[i]), -minx, -miny);
100     }
101 
102     for (i=0;i<g->npatrones;i++)
103     {
104         polygon_minbox(&(g->patrones[i]), &minx, &miny, &maxx, &maxy);
105         polygon_translate(&(g->patrones[i]), -minx, -miny);
106     }
107 
108     p.nholes = g->nhuecos;
109     p.p = &(g->plantilla);
110     p.h = g->huecos;
111 
112     g->area = polygonholes_area(&p);
113 
114     g->volumen = polygonholes_volumen(&p);
115 }
```

Gráfico de llamadas para esta función:



Here is the caller graph for this function:

6.3.3.3. void genesting_show (genesting * *g*)

Muestra la informacion del programa

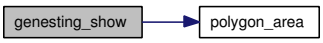
Parámetros:

← ***g*** Genesting

Definición en la línea 121 del archivo genesting.c.

```
122 {
123     int i;
124 
125     printf("Mostrando informacion de genesting\n");
126     printf("Plantilla:\n");
127     printf("  Vertices: %i\n",g->plantilla.nvertices);
128     printf("  Area: %f\n",polygon_area(&(g->plantilla)));
129 
130     printf("Huecos: %i\n",g->nhuecos);
131     for (i=0;i<g->nhuecos;i++)
132     {
133         printf("  Hueco %i:\n",i+1);
134         printf("    Vertices: %i\n",g->huecos[i].nvertices);
135         printf("    Area: %f\n",polygon_area(&(g->huecos[i])));
136     }
137 
138     printf("Area Util: %f\n",g->area);
139     printf("Volumen Util: %f\n",g->volumen);
140 
141     printf("Patrones: %i\n",g->npatrones);
142     for (i=0;i<g->npatrones;i++)
143     {
144         printf("  Patrones %i:\n",i+1);
145         printf("    Vertices: %i\n",g->patrones[i].nvertices);
146         printf("    Area: %f\n",polygon_area(&(g->patrones[i])));
147     }
148 }
```

Gráfico de llamadas para esta función:



Here is the caller graph for this function:

6.3.3.4. void individuo_create (genesting ∗ *g*, individuo ∗ *ind*)

Esta funcion crea un individuo de configuracion aleatoria.

El individuo con el ambiente definido en el objeto genesting y en una posicion aleatoria entre el rectangulo creado por la plantilla, Ademas se selecciona solo un patron que

conforma inicialmente el individuo, es posible que la configuracion obtenido genere un individuo no valido, pero esto debera ser verificado posteriormente por el algoritmo genetico.

Parámetros:

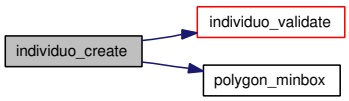
← ***g*** Obtiene el contexto del individuo

→ ***ind*** Una direccion de memoria donde esta el individuo

Definición en la línea 91 del archivo individuo.c.

```
92 {
93     float maxx,maxy,minx,miny;
94 
95     ind->ambiente = g;
96     ind->ngenes = 1;
97     ind->posgen = (posicion*) malloc (sizeof(posicion));
98     polygon_minbox(&(g->plantilla), &minx, &miny, &maxx, &maxy);
99 
100     do{
101     ind->posgen->x = (rand()%(int)(maxx-minx))+minx;
102     ind->posgen->y = (rand()%(int)(maxy-miny))+miny;
103     ind->posgen->t = (rand()%628)/100.0;
104     ind->posgen->id= rand()%g->npatrones;
105     } while (!individuo_validate(ind));
106 }
```

Gráfico de llamadas para esta función:



Here is the caller graph for this function:

6.3.3.5. float individuo_fitness (individuo ∗ *ind*)

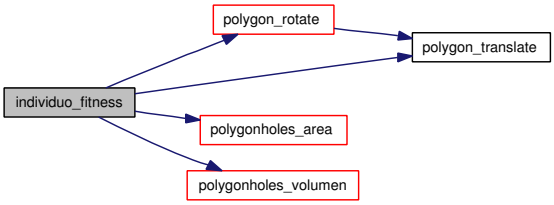
Tareas Pendientes

No es necesario recalcular el fitness de un individuo si este no ha cambiado

Definición en la línea 26 del archivo individuo.c.

```
27 {
28     int i,j,cont;
29 
30     polygon_holes temp;
31 
32     temp.p = &(ind->ambiente->plantilla);
33 
34     temp.h = (polygon*) malloc(sizeof(polygon)*(ind->ngenes+ind->ambiente->nhuecos));
35 
36     temp.nholes = ind->ngenes+ind->ambiente->nhuecos;
37 
38     for (i=0,cont=0;i<ind->ambiente->nhuecos;i++)
39     {
40         temp.h[cont].nvertices = ind->ambiente->huecos[i].nvertices;
41         temp.h[cont].v=(point*) malloc(sizeof(point)*temp.h[i].nvertices);
42 
43         for (j=0;j<temp.h[cont].nvertices;j++)
44         {
45             temp.h[cont].v[j].x=ind->ambiente->huecos[i].v[j].x;
46             temp.h[cont].v[j].y=ind->ambiente->huecos[i].v[j].y;
47         }
48         cont++;
49     }
50     for (i=0;i<ind->ngenes;i++)
51     {
52         temp.h[cont].nvertices = ind->ambiente->patrones[ind->posgen[i].id].nvertices;
53         temp.h[cont].v = (point*) malloc(sizeof(point)*temp.h[cont].nvertices);
54         for (j=0;j<temp.h[cont].nvertices;j++)
55         {
56             temp.h[cont].v[j].x=ind->ambiente->patrones[ind->posgen[i].id].v[j].x;
57             temp.h[cont].v[j].y=ind->ambiente->patrones[ind->posgen[i].id].v[j].y;
58         }
59         polygon_rotate(&(temp.h[cont]),ind->posgen[i].t);
60         polygon_translate(&(temp.h[cont]),ind->posgen[i].x, ind->posgen[i].y);
61         cont++;
62     }
63 
64     ind->fitness = polygonholes_volumen(&temp)/(ind->ambiente->volumen);
65 
66     ind->areautil = polygonholes_area(&temp);
67 
68     for (i=0;i<temp.nholes;i++)
69     {
70         free(temp.h[i].v);
71     }
72     free (temp.h);
73 
74     return (ind->fitness);
75 }
```

Gráfico de llamadas para esta función:



6.3.3.6. void individuo_mutate (individuo * *ind*)

Esta funcion modifica un individuo aleatoriamente.

Parámetros:

↔ ***ind*** Individuo

Definición en la línea 113 del archivo individuo.c.

```
114 {
115     int i;
116     for (i=0;i<ind->ngenes;i++){
117         ind->posgen[i].x += (rand()%4)-2;
118         ind->posgen[i].y += (rand()%4)-2;
119         ind->posgen[i].t += ((rand()%100)-50)/100.0;
120     }
121 };
```

6.3.3.7. void individuo_procreate (individuo * *p*, individuo * *m*, individuo * *ind*)

Esta funcion crea un nuevo individuo partiendo de los individuos padres. La mescla se hace de la siguiente manera, se toman los patrones del padre y luego los patrones de la madre si ya no estan ya colocados, si hay patrones comunes de forma cuasi aleatoria se escoge cual de las posiciones del patron es heredada.

Parámetros:

← ***p*** Individuo padre

← ***m*** Individuo madre

→ ***ind*** Nuevo individuo

Definición en la línea 121 del archivo individuo.c.

6.3.3.8. bool individuo_validate (individuo * *ind*)

Esta funcion identifica si un individuo es una solucion validad, esto es verdadero si se cumplen las siguientes condiciones:

- No se repite ningun patron dentro de la solucion
- Todos los patrones estan dentro de la plantilla
- Ningun patron se solapa con la plantilla, huecos u otro patron En caso contrario el individuo no es valido.

Parámetros:

← ***ind*** Individuo a validar

Devuelve:

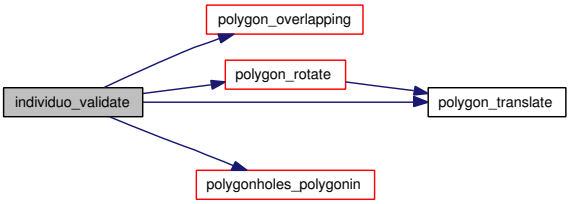
Verdadero si es un individuo valido, falso en caso contrario.

Definición en la línea 190 del archivo individuo.c.

```
191 {
192     bool valido = true;
193     int i,j;
194     polygon_holes ph;
195     polygon *pat;
196 
197     for (i=0;i<ind->ngenes-1 && valido;i++)
198     {
199         for (j=i+1;j<ind->ngenes && valido;j++)
200         {
201             if (ind->posgen[i].id == ind->posgen[j].id)
202                 valido = false;
203         }
204     }
205 
206     ph.p = &(ind->ambiente->plantilla);
207     ph.nholes = ind->ambiente->nhuecos;
208     ph.h = (polygon*) malloc(sizeof(polygon)*ph.nholes);
209 
210     for (i=0;i<ph.nholes;i++)
211     {
212         ph.h[i].nvertices = ind->ambiente->huecos[i].nvertices;
213         ph.h[i].v=(point*) malloc(sizeof(point)*ph.h[i].nvertices);
214 
215         for (j=0;j<ph.h[i].nvertices;j++)
216         {
217             ph.h[i].v[j].x=ind->ambiente->huecos[i].v[j].x;
218             ph.h[i].v[j].y=ind->ambiente->huecos[i].v[j].y;
219         }
220     }
221 
222     pat = (polygon*) malloc(sizeof(polygon)*ind->ngenes);
```

```
223
224     for (i=0;i<ind->ngenes;i++)
225     {
226         pat[i].nvertices = ind->ambiente->patrones[ind->posgen[i].id].nvertices;
227         pat[i].v = (point*) malloc(sizeof(point)*pat[i].nvertices);
228         for (j=0;j<pat[i].nvertices;j++)
229         {
230             pat[i].v[j].x=ind->ambiente->patrones[ind->posgen[i].id].v[j].x;
231             pat[i].v[j].y=ind->ambiente->patrones[ind->posgen[i].id].v[j].y;
232         }
233         polygon_rotate(&(pat[i]),ind->posgen[i].t);
234         polygon_translate(&(pat[i]),ind->posgen[i].x, ind->posgen[i].y);
235     }
236
237     for (i=0;i<ind->ngenes && valido;i++)
238     {
239         if(!polygonholes_polygonin(&ph, &(pat[i])))
240         {
241             valido=false;
242         }
243     }
244
245     for (i=0;i<ind->ngenes-1 && valido;i++)
246     {
247         for (j=i+1;j<ind->ngenes && valido;j++)
248         {
249             if (polygon_overlapping(&(pat[i]),&(pat[j])))
250             {
251                 valido=false;
252             }
253         }
254     }
255
256     for (i=0;i<ind->ngenes;i++)
257     {
258         free(pat[i].v);
259     }
260     free(pat);
261
262     for (i=0;i<ph.nholes;i++)
263     {
264         free(ph.h[i].v);
265     }
266     free(ph.h);
267
268     return valido;
269 }
```

Gráfico de llamadas para esta función:



Here is the caller graph for this function:

6.3.3.9. genesting * leer_archivo (char * *arc_name*)

Tareas Pendientes

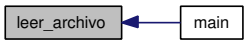
Documentar como es la estructura leida por el programa.

Definición en la línea 25 del archivo genesting.c.

```
26 {
27     FILE* arc;
28     genesting *g;
29     int npoly;
30     int i,j;
31 
32     g =(genesting*) malloc (sizeof(genesting));
33 
34     arc=fopen(arc_name,"r");
35 
36     if(arc==NULL){
37             fprintf(stderr,"No se pudo abrir el archivo\n");
38             exit(1);
39     }
40 
41     fscanf(arc,"%i",&npoly);
42 
43     g->nhuecos =0;
44     g->npatrones =0;
45 
46     g->huecos=(polygon*) malloc(sizeof(polygon)*npoly-1);
47     g->patrones=(polygon*) malloc(sizeof(polygon)*npoly-1);
48 
49     for (i=0;i<npoly;i++)
50     {
51         int nvert,tipo;
52 
53         polygon *p=NULL;
```

```
54          fscanf(arc,"%i %i",&nvert,&tipo);
55          switch(tipo)
56          {
57              case 1:
58              p=&(g->plantilla);
59              break;
60              case 2:
61              p=&(g->patrones[g->npatrones++]);
62              break;
63              case 3:
64              p=&(g->huecos[g->nhuecos++]);
65              break;
66          }
67          p->nvertices = nvert;
68          p->v=(point*) malloc(sizeof(point)*nvert);
69          for (j=0;j<nvert;j++)
70          {
71              fscanf(arc,"%f %f",&(p->v[j].x),&(p->v[j].y));
72          }
73      }
74      g->huecos=(polygon*) realloc(g->huecos,sizeof(polygon)*g->nhuecos);
75      g->patrones=(polygon*) realloc(g->patrones,sizeof(polygon)*g->npatrones);
76
77      fclose(arc);
78
79      return g;
80  }
```

Here is the caller graph for this function:



6.3.3.10. void population_create (population * *p*, genesting * *g*, int *n*)

Esta funcion crea una poblacion inicial de soluciones.

Parámetros:

→ ***p*** Poblacion

← ***g*** Genesting

← ***n*** Numero de elementos de la poblacion

Definición en la línea 23 del archivo population.c.

```
23                                                      {
24      int i;
25      p->ambiente=g;
26      p->nindividuos=n;
```

```
27     p->individuos = (individuo*) malloc(sizeof(individuo)*p->nindividuos);
28
29     for (i=0;i<p->nindividuos;i++){
30             individuo_create(p->ambiente,&(p->individuos[i]));
31     }
32 };
```

Gráfico de llamadas para esta función:

Here is the caller graph for this function:

6.3.3.11. void population_evaluate (population * p)

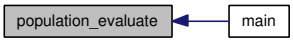
Evalua una poblacion, para esto calcula todos los fitness, y ordena los individuos segun su fitness

Parámetros:

↔ ***p*** Poblacion

Definición en la línea 32 del archivo population.c.

Here is the caller graph for this function:



6.3.3.12. void population_generation (population * p)

Crea una nueva poblacion, dividiendo la exitente en tres grupos, dejando el primer grupo intacto, el segundo lo muta, y el tercero es un cruce de los 2 primeros

Parámetros:

↔ ***p*** Poblacion

Definición en la línea 47 del archivo population.c.

Here is the caller graph for this function:

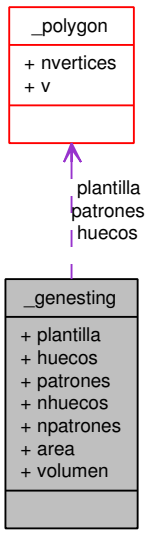
Capítulo 7

Genesting Documentación de clases

7.1. Referencia de la Estructura _genesting

`#include <genesting.h>`

Diagrama de colaboración para _genesting:



7.1.1. Descripción detallada

Definición en la línea 14 del archivo genesting.h.

Atributos públicos

- polygon plantilla
- polygon * huecos
- polygon * patrones
- unsigned int nhuecos
- unsigned int npatrones
- float area
- float volumen

7.1.2. Documentación de los datos miembro

7.1.2.1. float _genesting::area

Definición en la línea 21 del archivo genesting.h.

7.1.2.2. polygon* _genesting::huecos

Definición en la línea 17 del archivo genesting.h.

7.1.2.3. unsigned int _genesting::nhuecos

Definición en la línea 19 del archivo genesting.h.

7.1.2.4. unsigned int _genesting::npatrones

Definición en la línea 20 del archivo genesting.h.

7.1.2.5. polygon* _genesting::patrones

Definición en la línea 18 del archivo genesting.h.

7.1.2.6. polygon _genesting::plantilla

Definición en la línea 16 del archivo genesting.h.

7.1.2.7. float _genesting::volumen

Definición en la línea 22 del archivo genesting.h.

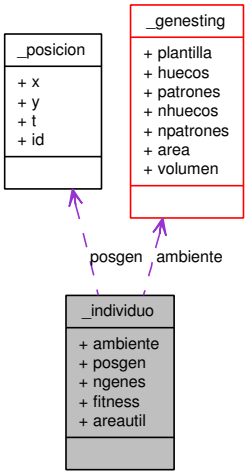
La documentación para esta estructura fué generada a partir del siguiente archivo:

- genesting.h

7.2. Referencia de la Estructura _individuo

`#include <individuo.h>`

Diagrama de colaboración para _individuo:



7.2.1. Descripción detallada

Definición en la línea 29 del archivo individuo.h.

Atributos públicos

- genesting * ambiente
- posicion * posgen
- unsigned int ngenes
- float fitness
- float areautil

7.2.2. Documentación de los datos miembro

7.2.2.1. genesting* _individuo::ambiente

Definición en la línea 31 del archivo individuo.h.

7.2.2.2. float _individuo::areautil

Definición en la línea 36 del archivo individuo.h.

7.2.2.3. float _individuo::fitness

Definición en la línea 35 del archivo individuo.h.

7.2.2.4. unsigned int _individuo::ngenes

Definición en la línea 34 del archivo individuo.h.

7.2.2.5. posicion* _individuo::posgen

Definición en la línea 32 del archivo individuo.h.

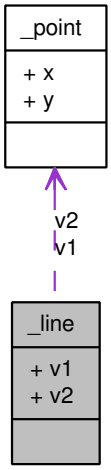
La documentación para esta estructura fué generada a partir del siguiente archivo:

- individuo.h

7.3. Referencia de la Estructura _line

`#include <line.h>`

Diagrama de colaboración para _line:



7.3.1. Descripción detallada

Definición en la línea 16 del archivo line.h.

Atributos públicos

- point v1
- point v2

7.3.2. Documentación de los datos miembro

7.3.2.1. point _line::v1

Definición en la línea 18 del archivo line.h.

7.3.2.2. point _line::v2

Definición en la línea 18 del archivo line.h.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- line.h

7.4. Referencia de la Estructura _point

`#include <point.h>`

7.4.1. Descripción detallada

Definición en la línea 12 del archivo point.h.

Atributos públicos

- float x
- float y

7.4.2. Documentación de los datos miembro

7.4.2.1. float _point::x

Definición en la línea 14 del archivo point.h.

7.4.2.2. float _point::y

Definición en la línea 15 del archivo point.h.

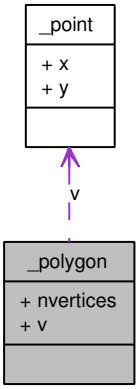
La documentación para esta estructura fué generada a partir del siguiente archivo:

- point.h

7.5. Referencia de la Estructura _polygon

`#include <polygon.h>`

Diagrama de colaboración para _polygon:



7.5.1. Descripción detallada

Definición en la línea 18 del archivo polygon.h.

Atributos públicos

- unsigned int nvertices
- point * v

7.5.2. Documentación de los datos miembro

7.5.2.1. unsigned int _polygon::nvertices

Definición en la línea 20 del archivo polygon.h.

7.5.2.2. point* _polygon::v

Definición en la línea 21 del archivo polygon.h.

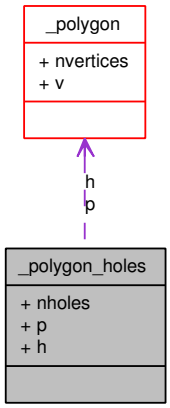
La documentación para esta estructura fué generada a partir del siguiente archivo:

- polygon.h

7.6. Referencia de la Estructura _polygon_holes

`#include <polygon_holes.h>`

Diagrama de colaboración para _polygon_holes:



7.6.1. Descripción detallada

Definición en la línea 17 del archivo polygon_holes.h.

Atributos públicos

- unsigned int nholes
- polygon * p
- polygon * h

7.6.2. Documentación de los datos miembro

7.6.2.1. polygon* _polygon_holes::h

Definición en la línea 21 del archivo polygon_holes.h.

7.6.2.2. unsigned int _polygon_holes::nholes

Definición en la línea 19 del archivo polygon_holes.h.

7.6.2.3. polygon* _polygon_holes::p

Definición en la línea 20 del archivo polygon_holes.h.

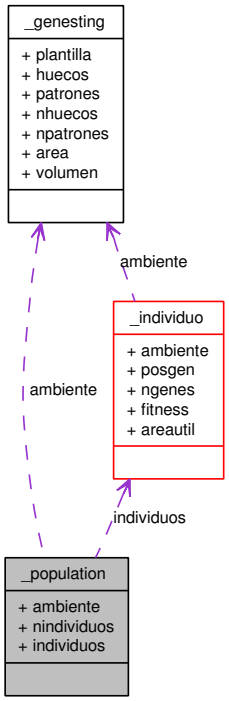
La documentación para esta estructura fué generada a partir del siguiente archivo:

- polygon_holes.h

7.7. Referencia de la Estructura _population

`#include <population.h>`

Diagrama de colaboración para _population:



7.7.1. Descripción detallada

Definición en la línea 15 del archivo population.h.

Atributos públicos

- genesting * ambiente
- int nindividuos
- individuo * individuos

7.7.2. Documentación de los datos miembro

7.7.2.1. genesting* _population::ambiente

Definición en la línea 16 del archivo population.h.

7.7.2.2. individuo* _population::individuos

Definición en la línea 18 del archivo population.h.

7.7.2.3. int _population::nindividuos

Definición en la línea 17 del archivo population.h.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- population.h

7.8. Referencia de la Estructura _posicion

`#include <individuo.h>`

7.8.1. Descripción detallada

Definición en la línea 21 del archivo individuo.h.

Atributos públicos

- float x
- float y
- float t
- unsigned int id

7.8.2. Documentación de los datos miembro

7.8.2.1. unsigned int _posicion::id

Definición en la línea 26 del archivo individuo.h.

7.8.2.2. float _posicion::t

Definición en la línea 25 del archivo individuo.h.

7.8.2.3. float _posicion::x

Definición en la línea 23 del archivo individuo.h.

7.8.2.4. float _posicion::y

Definición en la línea 24 del archivo individuo.h.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- individuo.h

Capítulo 8

Genesting Documentación de archivos

8.1. Referencia del Archivo distance.c

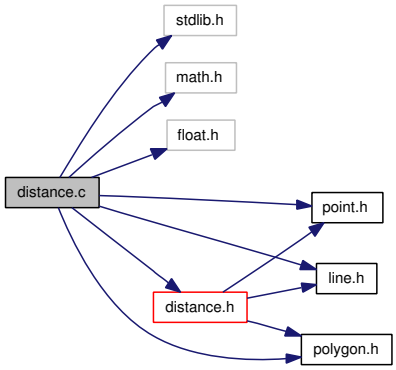
8.1.1. Descripción detallada

En este archivo se definen las diferentes funciones para encontrar la distancia entre varios objetos.

Definición en el archivo distance.c.

```
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include "distance.h"
#include "point.h"
#include "line.h"
#include "polygon.h"
```

Dependencia gráfica adjunta para distance.c:



Funciones

- float distance_pointpoint (point *a, point *b)
- float distance_pointline (point *f, line *l, int *seg)
- float distance_pointpolygon (point *f, polygon *p, line *ref)
- float distance_pointpolygonholes (point *f, polygon_holes *p, line *ref)

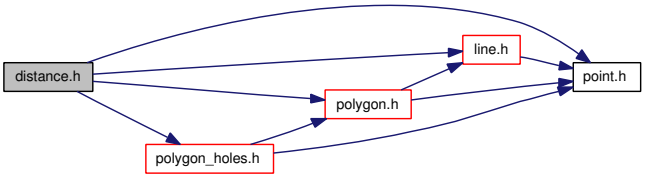
8.2. Referencia del Archivo distance.h

8.2.1. Descripción detallada

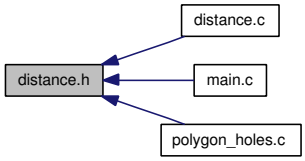
Definición en el archivo distance.h.

```
#include "point.h"
#include "line.h"
#include "polygon.h"
#include "polygon_holes.h"
```

Dependencia gráfica adjunta para distance.h:



Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Funciones

- float distance_pointpoint (point *a, point *b)
- float distance_pointline (point *f, line *l, int *seg)
- float distance_pointpolygon (point *f, polygon *p, line *ref)
- float distance_pointpolygonholes (point *f, polygon_holes *p, line *ref)

8.3. Referencia del Archivo genesting.c

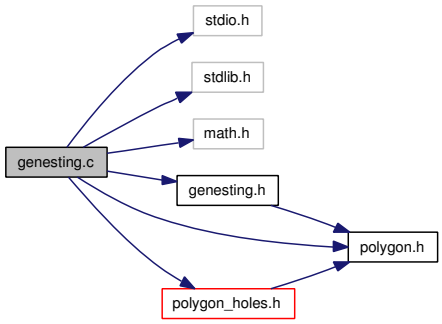
8.3.1. Descripción detallada

Aqui estan las estructuras principales del programa la estructura es muy sencilla

Definición en el archivo genesting.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "genesting.h"
#include "polygon.h"
#include "polygon_holes.h"
```

Dependencia gráfica adjunta para genesting.c:



Funciones

- genesting * leer_archivo (char *arc_name)
- void genesting_init (genesting *g)
- void genesting_show (genesting *g)

8.4. Referencia del Archivo genesting.h

8.4.1. Descripción detallada

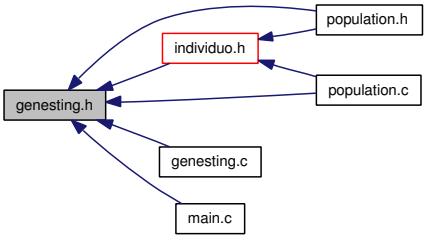
Declaraciones del Objeto Genesting

Definición en el archivo genesting.h.

```
#include "polygon.h"
```

Dependencia gráfica adjunta para genesting.h:

Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

- struct _genesting

Tipos definidos

- typedef _genesting genesting

Funciones

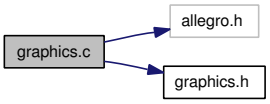
- genesting * leer_archivo (char *arc_name)
- void genesting_init (genesting *g)
- void genesting_show (genesting *g)

8.5. Referencia del Archivo graphics.c

```
#include <allegro.h>
```

```
#include "graphics.h"
```

Dependencia gráfica adjunta para graphics.c:



Funciones

- void init_graphics ()
- void draw_rect (float minx, float miny, float maxx, float maxy, int r, int g, int b)
- void draw_line (float x1, float y1, float x2, float y2)
- void getscreen ()
- void relscreen ()
- void clearscreen ()

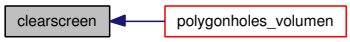
8.5.1. Documentación de las funciones

8.5.1.1. void clearscreen ()

Definición en la línea 52 del archivo graphics.c.

```
52                  {
53 #if graphics
54 clear_bitmap(screen);
55 #endif
56 }
```

Here is the caller graph for this function:

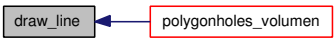


8.5.1.2. void draw_line (float *x1*, float *y1*, float *x2*, float *y2*)

Definición en la línea 34 del archivo graphics.c.

```
34                                                      {
35 #if graphics
36     line(screen,(int)x1,(int)y1,(int)x2,(int)y2,makecol(255,0,0));
37 #endif
38 }
```

Here is the caller graph for this function:



8.5.1.3. void draw_rect (float *minx*, float *miny*, float *maxx*, float *maxy*, int *r*, int *g*, int *b*)

Definición en la línea 28 del archivo graphics.c.

```
28                                                                                                  {
29 #if graphics
30 //    rect(screen, (int)minx, (int)miny, (int) maxx, (int) maxy, makecol(r,g,b));
31 #endif
32 }
```

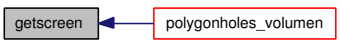
Here is the caller graph for this function:

8.5.1.4. void getscreen ()

Definición en la línea 40 del archivo graphics.c.

```
40                 {
41 #if graphics
42      acquire_screen();
43 #endif
44 }
```

Here is the caller graph for this function:

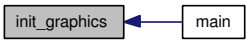


8.5.1.5. void init_graphics ()

Definición en la línea 6 del archivo graphics.c.

```
7 {
8 #if graphics
9     int depth, res;
10     allegro_init();
11     depth = desktop_color_depth();
12     if (depth == 0)
13         depth = 32;
14     set_color_depth(depth);
15     res = set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);
16     if (res != 0)
17     {
18         allegro_message(allegro_error);
19         exit(-1);
20     }
21
22     install_timer();
23     //install_keyboard();
24     //install_mouse();
25 #endif
26 }
```

Here is the caller graph for this function:



8.5.1.6. void relscreen ()

Definición en la línea 46 del archivo graphics.c.

```
46                 {
47 #if graphics
48     release_screen();
49 #endif
50 }
```

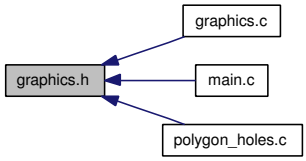
Here is the caller graph for this function:

8.6. Referencia del Archivo graphics.h

8.6.1. Descripción detallada

Definición en el archivo graphics.h.

Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Definiciones

- #define graphics 0
- #define DELTA 10

8.6.2. Documentación de las definiciones

8.6.2.1. #define DELTA 10

Definición en la línea 8 del archivo graphics.h.

8.6.2.2. #define graphics 0

Definición en la línea 6 del archivo graphics.h.

8.7. Referencia del Archivo individuo.c

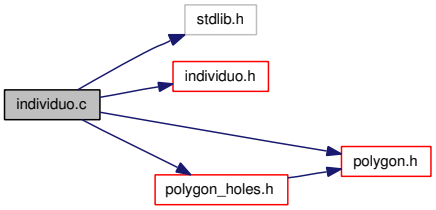
8.7.1. Descripción detallada

Aqui estan las estructuras principales del programa la estructura es muy sencilla

Definición en el archivo individuo.c.

```
#include <stdlib.h>
#include "individuo.h"
#include "polygon.h"
#include "polygon_holes.h"
```

Dependencia gráfica adjunta para individuo.c:



Funciones

- float individuo_fitness (individuo *ind)
- void individuo_create (genesting *g, individuo *ind)
- void individuo_mutate (individuo *ind)
- void individuo_procreate (individuo *p, individuo *m, individuo *ind)
- bool individuo_validate (individuo *ind)
- int comparar_individuos (individuo *ind1, individuo *ind2)

8.8. Referencia del Archivo individuo.h

8.8.1. Descripción detallada

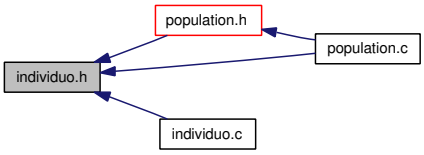
Declaraciones del Objeto Individuo

Definición en el archivo individuo.h.

```
#include "genesting.h"
```

Dependencia gráfica adjunta para individuo.h:

Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

- struct _posicion
- struct _individuo

Tipos definidos

- typedef _posicion posicion
- typedef _individuo individuo

Funciones

- float individuo_fitness (individuo *ind)
- void individuo_create (genesting *g, individuo *ind)
- void individuo_mutate (individuo *ind)
- void individuo_procreate (individuo *p, individuo *m, individuo *ind)
- bool individuo_validate (individuo *ind)
- int comparar_individuos (individuo *ind1, individuo *ind2)

8.9. Referencia del Archivo line.c

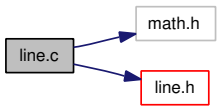
8.9.1. Descripción detallada

Este archivo define las funciones que operan sobre el objeto segmento de linea

Definición en el archivo line.c.

```
#include <math.h>
#include "line.h"
```

Dependencia gráfica adjunta para line.c:



Funciones

- bool line_intersection (line *l1, line *l2)
- bool line_equal (line *l1, line *l2)
- bool line_ispoint (line *l1)

8.10. Referencia del Archivo line.h

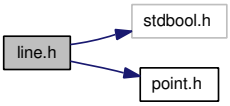
8.10.1. Descripción detallada

Declaraciones del Objeto Segmento de Linea

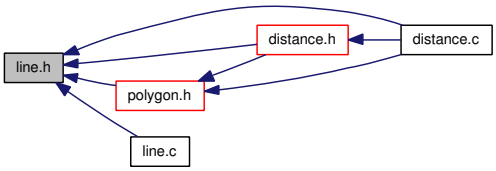
Definición en el archivo line.h.

```
#include <stdbool.h>
#include "point.h"
```

Dependencia gráfica adjunta para line.h:



Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

- struct _line

Tipos definidos

- typedef _line line

Funciones

- bool line_intersection (line *l1, line *l2)
- bool line_equal (line *l1, line *l2)
- bool line_ispoint (line *l1)

8.11. Referencia del Archivo main.c

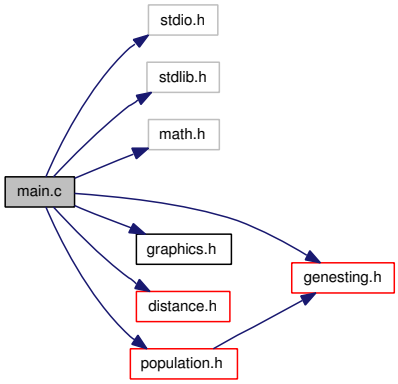
8.11.1. Descripción detallada

Programa principal, usa y prueba la libreria genesting.

Definición en el archivo main.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "genesting.h"
#include "graphics.h"
#include "distance.h"
#include "population.h"
```

Dependencia gráfica adjunta para main.c:



Funciones

- int main (int argc, char **argv)

8.11.2. Documentación de las funciones

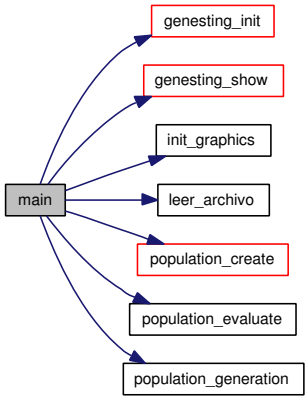
8.11.2.1. int main (int *argc*, char ** *argv*)

Definición en la línea 82 del archivo main.c.

```
83 {
84     genesting *g;
85     population *p;
86 
87 #if graphics
88 
89     init_graphics();
90 #endif
91 
92     if (argc<2)
93     {
94         fprintf(stderr,"Se deben entrar ingresar el nombre del archivo a leer\n");
95         return 1;
96     }
97 
98     g=leer_archivo(argv[1]);
99     p=(population*) malloc(sizeof(population));
100 
101     srand((int)p);
102 
103     genesting_init(g);
104 
105     genesting_show(g);
106 
107     population_create(p,g, 200);
108 
109     int i,k;
110     for (k=0;k<30;k++)
111     {
112         printf("Iteracion: %i\n",k);
113 
114         /*
115         for (i=0;i<5;i++)
116         {
117             printf("Individuo %i: [%i] \n",i,p->individuos[i].ngenes);
118             for (j=0;j<p->individuos[i].ngenes;j++)
119             {
120                 printf("Pat[%i] x[%f] y[%f] t[%f]\n",
121                        p->individuos[i].posgen[j].id,
122                        p->individuos[i].posgen[j].x,
123                        p->individuos[i].posgen[j].y,
124                        p->individuos[i].posgen[j].t);
125             }
126         }
127         */
128         population_evaluate(p);
129 
130         for (i=0;i<10;i++)
131         {
132             printf("Individuo %i: [%i] fitness: %f areautil: %f\n",i,p->individuos[i].ngenes,p->indivi
133             /*
134                         for (j=0;j<p->individuos[i].ngenes;j++)
135                         {
136                             printf("Pat[%i] x[%f] y[%f] t[%f]\n",
137                                    p->individuos[i].posgen[j].id,
138                                    p->individuos[i].posgen[j].x,
139                                    p->individuos[i].posgen[j].y,
```

```
140                                     p->individuos[i].posgen[j].t);
141                             }
142                 */
143             }
144
145             population_generation(p);
146         }
147         free(p);
148
149         return 0;
150 }
```

Gráfico de llamadas para esta función:



8.12. Referencia del Archivo point.c

8.12.1. Descripción detallada

En este archivo definimos las funciones utilizadas por el objeto punto.

Definición en el archivo point.c.

```
#include "point.h"
```

Dependencia gráfica adjunta para point.c:

Funciones

- float point_dot (point *a, point *b, point *c)
- float point_cross (point *a, point *b, point *c)

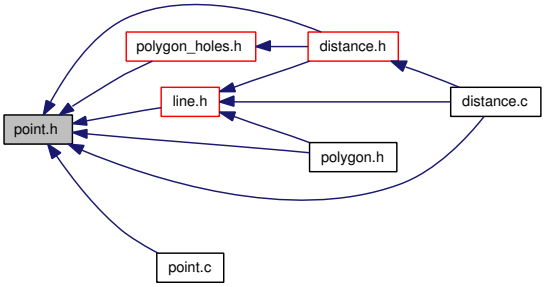
8.13. Referencia del Archivo point.h

8.13.1. Descripción detallada

Implementacion de un objeto punto en 2 dimensiones

Definición en el archivo point.h.

Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

- struct _point

Tipos definidos

- typedef _point point

Funciones

- float point_dot (point *a, point *b, point *c)
- float point_cross (point *a, point *b, point *c)

8.14. Referencia del Archivo polygon.c

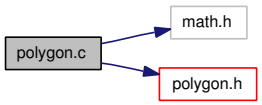
8.14.1. Descripción detallada

Este archivo define las funciones utilizadas por el objeto poligono

Definición en el archivo polygon.c.

```
#include <math.h>
#include "polygon.h"
```

Dependencia gráfica adjunta para polygon.c:



Funciones

- float polygon_area (polygon *p)
- bool polygon_pointin (polygon *p, point *f)
- bool polygon_overlapping (polygon *p, polygon *q)
- void polygon_rotate (polygon *p, float t)
- point polygon_center (polygon *p)
- void polygon_minbox (polygon *p, float *minx, float *miny, float *maxx, float *maxy)
- void polygon_translate (polygon *p, float x, float y)

8.15. Referencia del Archivo polygon.h

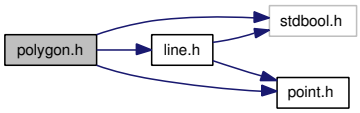
8.15.1. Descripción detallada

Declaraciones del Objeto Poligono

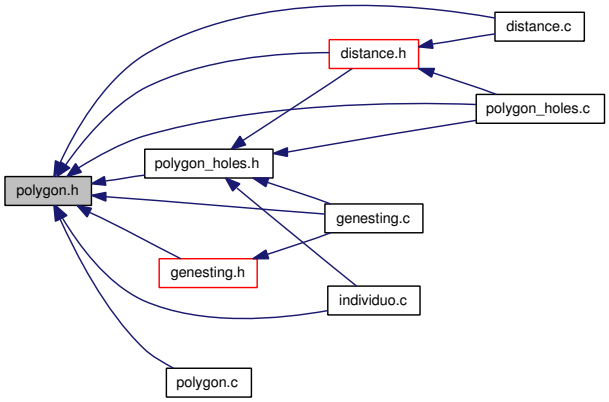
Definición en el archivo polygon.h.

```
#include <stdbool.h>
#include "point.h"
#include "line.h"
```

Dependencia gráfica adjunta para polygon.h:



Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

- struct _polygon

Tipos definidos

- typedef _polygon polygon

Funciones

- float polygon_area (polygon *p)
- bool polygon_pointin (polygon *p, point *f)
- bool polygon_overlapping (polygon *p, polygon *q)
- void polygon_rotate (polygon *p, float t)
- void polygon_minbox (polygon *p, float *minx, float *miny, float *maxx, float *maxy)
- void polygon_translate (polygon *p, float x, float y)

8.16. Referencia del Archivo polygon_holes.c

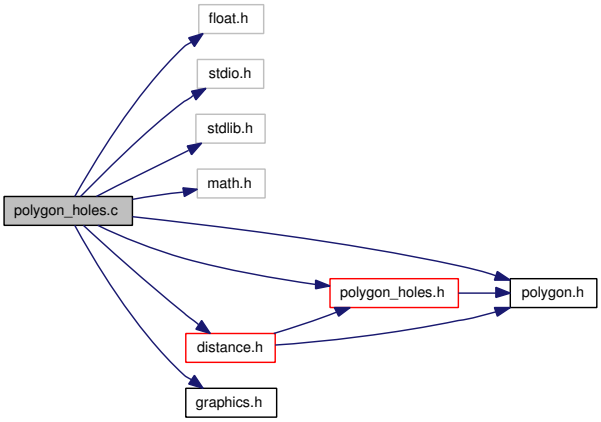
8.16.1. Descripción detallada

Este archivo define las funciones que se utilizan en el Objeto Poligono con huecos

Definición en el archivo polygon_holes.c.

```
#include <float.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "polygon_holes.h"
#include "polygon.h"
#include "distance.h"
#include "graphics.h"
```

Dependencia gráfica adjunta para polygon_holes.c:



Funciones

- float polygonholes_area (polygon_holes *p)
- float polygonholes_volumen (polygon_holes *p)
- bool polygonholes_pointin (polygon_holes *p, point *f)
- bool polygonholes_pointinhole (polygon_holes *p, point *f)
- bool polygonholes_polygonin (polygon_holes *p, polygon *q)

8.17. Referencia del Archivo polygon_holes.h

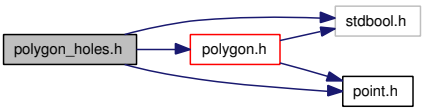
8.17.1. Descripción detallada

Declaraciones del Objeto Poligono con huecos

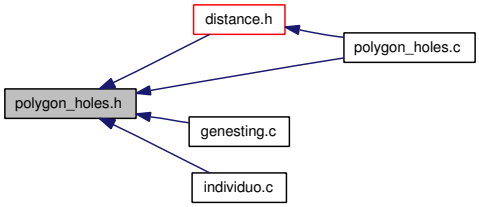
Definición en el archivo polygon_holes.h.

```
#include <stdbool.h>
#include "point.h"
#include "polygon.h"
```

Dependencia gráfica adjunta para polygon_holes.h:



Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

- struct _polygon_holes

Tipos definidos

- typedef _polygon_holes polygon_holes

Funciones

- float polygonholes_area (polygon_holes *p)

- float polygonholes_volumen (polygon_holes *p)
- bool polygonholes_pointin (polygon_holes *p, point *f)
- bool polygonholes_polygonin (polygon_holes *p, polygon *q)
- bool polygonholes_pointinhole (polygon_holes *p, point *f)

8.18. Referencia del Archivo population.c

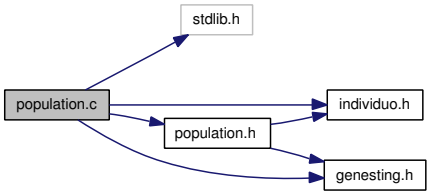
8.18.1. Descripción detallada

Definiciones del Objeto Poblacion

Definición en el archivo population.c.

```
#include <stdlib.h>
#include "population.h"
#include "genesting.h"
#include "individuo.h"
```

Dependencia gráfica adjunta para population.c:



Funciones

- void population_create (population *p, genesting *g, int n)
- void population_evaluate (population *p)
- void population_generation (population *p)

8.19. Referencia del Archivo population.h

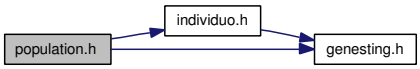
8.19.1. Descripción detallada

Declaraciones del Objeto Poblacion

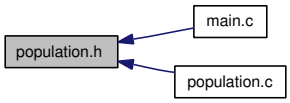
Definición en el archivo population.h.

```
#include "individuo.h"
#include "genesting.h"
```

Dependencia gráfica adjunta para population.h:



Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

- struct _population

Tipos definidos

- typedef _population population

Funciones

- void population_create (population *p, genesting *g, int n)
- void population_evaluate (population *p)
- void population_generation (population *p)

Capítulo 9

Genesting Documentación de páginas

9.1. Listado de Tareas Pendientes

Archivo main.c
- Se deben cambiar los typedef de los objetos para que sean punteros
- Para calcular el fitness es mejor comprobar primero el area y despues el fitness, asi se ahorra mucho calculo

Global individuo_fitness No es necesario recalcular el fitness de un individuo si este no ha cambiado

Global leer_archivo Documentar como es la estructura leida por el programa.

9.2. Lista de Bugs

Global polygonholes_volumen Esta funcion para calcular el volument de un rectangulo que intersecta un cono tiene valores indeterminados en algunos segmentos del problema, es posible que este error aparesca solo en algunos casos, pero hay que examinar mas la funcion.

Índice alfabético

_genesting, 55
    area, 56
    huecos, 56
    nhuecos, 56
    npatrones, 56
    patrones, 56
    plantilla, 56
    volumen, 56
_individuo, 58
    ambiente, 58
    areautil, 58
    fitness, 59
    ngenes, 59
    posgen, 59
_line, 60
    v1, 60
    v2, 60
_point, 62
    x, 62
    y, 62
_polygon, 63
    nvertices, 63
    v, 63
_polygon_holes, 64
    h, 64
    nholes, 64
    p, 64
_population, 66
    ambiente, 67
    individuos, 67
    nindividuos, 67
_posicion, 68
    id, 68
    t, 68
    x, 68
    y, 68
ambiente
    _individuo, 58
    _population, 67
area
    _genesting, 56
areautil
    _individuo, 58

clearscreen
    graphics.c, 74
comparar_individuos
    genetic, 42

DELTA
    graphics.h, 77
distance
    distance_pointline, 12
    distance_pointpoint, 13
    distance_pointpolygon, 13
    distance_pointpolygonholes, 15
distance.c, 69
distance.h, 71
distance_pointline
    distance, 12
distance_pointpoint
    distance, 13
distance_pointpolygon
    distance, 13
distance_pointpolygonholes
    distance, 15
Distancias, 11
draw_line
    graphics.c, 74
draw_rect
    graphics.c, 75

fitness

_individuo, 59

genesting
genetic, 42
genesting.c, 72
genesting.h, 73
genesting_init
genetic, 42
genesting_show
genetic, 43
genetic
comparar_individuos, 42
genesting, 42
genesting_init, 42
genesting_show, 43
individuo, 42
individuo_create, 44
individuo_fitness, 45
individuo_mutate, 47
individuo_procreate, 47
individuo_validate, 47
leer_archivo, 50
population, 42
population_create, 51
population_evaluate, 52
population_generation, 52
posicion, 42
Genetico, 41
Geometria, 17
geometry
line, 18
line_equal, 18
line_intersection, 19
line_ispoint, 21
point, 18
point_cross, 21
point_dot, 22
polygon, 18
polygon_area, 23
polygon_center, 23
polygon_holes, 18
polygon_minbox, 24
polygon_overlapping, 25
polygon_pointin, 27
polygon_rotate, 27
polygon_translate, 28
polygonholes_area, 29
polygonholes_pointin, 30
polygonholes_pointinhole, 31
polygonholes_polygonin, 32
polygonholes_volumen, 33
getscreen
graphics.c, 75
graphics
graphics.h, 77
graphics.c, 74
clearscreen, 74
draw_line, 74
draw_rect, 75
getscreen, 75
init_graphics, 75
relscreen, 76
graphics.h, 77
DELTA, 77
graphics, 77

h
_polygon_holes, 64
huecos
_genesting, 56

id
_posicion, 68
individuo
genetic, 42
individuo.c, 78
individuo.h, 79
individuo_create
genetic, 44
individuo_fitness
genetic, 45
individuo_mutate
genetic, 47
individuo_procreate
genetic, 47
individuo_validate
genetic, 47
individuos
_population, 67
init_graphics
graphics.c, 75

leer_archivo

genetic, 50
line
geometry, 18
line.c, 80
line.h, 81
line_equal
geometry, 18
line_intersection
geometry, 19
line_ispoint
geometry, 21

main
main.c, 82
main.c, 82
main, 82

ngenes
_individuo, 59
nholes
_polygon_holes, 64
nhuecos
_genesting, 56
nindividuos
_population, 67
npatrones
_genesting, 56
nvertices
_polygon, 63

p
_polygon_holes, 64
patrones
_genesting, 56
plantilla
_genesting, 56
point
geometry, 18
point.c, 85
point.h, 86
point_cross
geometry, 21
point_dot
geometry, 22
polygon
geometry, 18
polygon.c, 87
polygon.h, 88
polygon_area
geometry, 23
polygon_center
geometry, 23
polygon_holes
geometry, 18
polygon_holes.c, 90
polygon_holes.h, 91
polygon_minbox
geometry, 24
polygon_overlapping
geometry, 25
polygon_pointin
geometry, 27
polygon_rotate
geometry, 27
polygon_translate
geometry, 28
polygonholes_area
geometry, 29
polygonholes_pointin
geometry, 30
polygonholes_pointinhole
geometry, 31
polygonholes_polygonin
geometry, 32
polygonholes_volumen
geometry, 33
population
genetic, 42
population.c, 93
population.h, 94
population_create
genetic, 51
population_evaluate
genetic, 52
population_generation
genetic, 52
posgen
_individuo, 59
posicion
genetic, 42

relscreen

graphics.c, 76

t
_posicion, 68

v
_polygon, 63
v1
_line, 60
v2
_line, 60
volumen
_genesting, 56

x
_point, 62
_posicion, 68

y
_point, 62
_posicion, 68