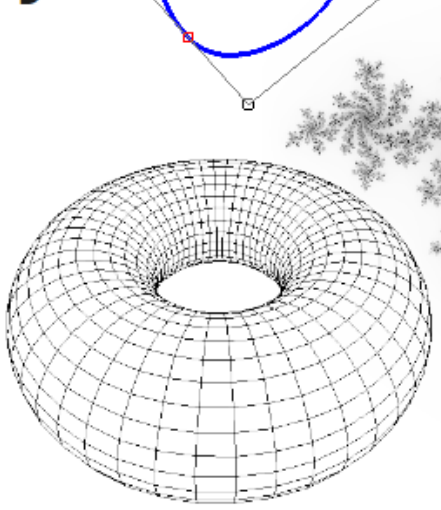
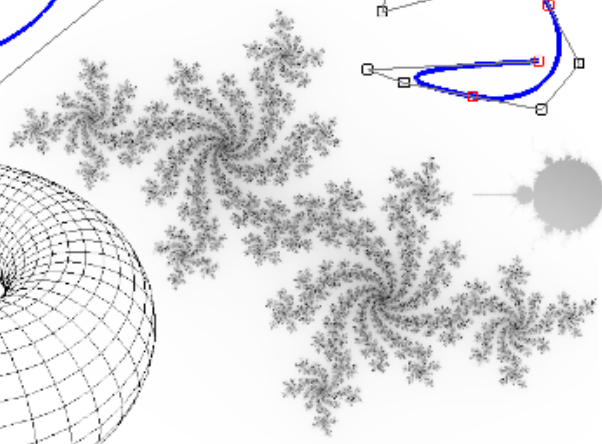
Una humilde introducción a la Graficación por Computadora y otras yerbas





v 1.0

Eduardo
NAVAS

Departamento de Electrónica e Informática,
Universidad Centroamericana
"José Simeón Cañas"

Una Humilde Introducción a la Graficación por Computadora y Otras Yerbas

por *Eduardo NAVAS*

versión 1.0
2010.03.08

Este libro fue desarrollado únicamente con software libre. Entre las herramientas usadas, se encuentran: LaTeX, LyX, GNU/Linux, GNOME, KDE, XaoS, Maxima, KmPlot, OpenOffice.org, Geany, Inkscape, GIMP, Python, GCC, SDL, PyGAME, GTK+, Qt4, etc.

La versión digital y el material adicional puede ser descargado de:
www.aliamondano-eo.wikidot.com/libro-graficos
http://dei.uca.edu.sv/publicaciones/

ISBN: 978-99923-73-36-1

Editado y preparado desde el
Departamento de Electrónica e Infomática de la
Universidad Centroamericana "José Simeón Cañas",
El Salvador, Centroamérica.

Dedico esta obra...

.- A todos los que confiaron en mí (con mucha frecuencia mucho más de lo que yo mismo) en este viaje en tren, que se llama *"mi vida"*;

.- A mi creador y señor que siempre se quedó conmigo, aún cuando yo no me quedé siempre con él;

.- A todos los que quieren ayudar a construir
el "Otro Mundo Posible"

.- Al tiuj kiuj eĉ kontraŭflue kunhelpas fari pli bonan mondon

Prólogo

Descripción general

Este libro está siendo desarrollado con el propósito de ser libro de texto para diversas materias impartidas y a impartir para la carrera de Licenciatura en Ciencias de la Computación y otras carreras de grado y postgrado en la Universidad Centroamericana "José Simeón Cañas" en El Salvador. Está pensado para estudiantes que ya han aprobado los cursos en los que se estudia cálculo infinitesimal, geometría analítica vectorial, estructura de datos y programación orientada a objetos.

La graficación por computadora es una de las principales áreas de estudio de las ciencias de la computación, con aplicación en todos los ámbitos de la computación; desde la necesidad de representar medianas o grandes cantidades de datos numéricos que serían ilegibles en texto, hasta el desarrollo de sofisticadas aplicaciones científicas de simulación de modelos matemáticos del clima; desde el uso de software de edición de fotografías, hasta el desarrollo de videojuegos. En todos los ámbitos, surge la necesidad de tener conocimientos elementales de graficación por computadora.

El contendio del libro comienza con una breve introducción a SDL (y PyGame) que es la biblioteca gráfica base a ser usada en los primeros capítulos del libro. Se sigue con una introducción teórica a la graficación por computador, en la que se presentan algunos conceptos elementales que son imprescindibles para la programación de aplicaciones gráficas interactivas.

En el capítulo siguiente se aborda ámpliamente el tema de la discretización de líneas rectas y circunferencias de un pixel de grosor. Se aborda también el tema de relleno de circunferencias. Luego se procede a hacer un riguroso análisis vectorial del tema de cambio de coordenadas o cambio de marco de referencia. Allí se incluye una aplicación de ejemplo en la que se pone en práctica la teoría presentada.

Después, se describe la teoría matemática matricial relacionada con las transformaciones geométricas bidimensionales y tridimensionales. Se incluye entonces una aplicación de corte pedagógico que permite practicar transformaciones geométricas tridimensionales y permite ver en tiempo real sus efectos sobre un conjunto de objetos geométricamente sencillos.

El siguiente capítulo, muestra la teoría necesaria para comprender cómo transformar

un objeto de tres dimensiones a una imagen bidimensional y cómo implementar tal transformación.

A continuación, se incluye una reflexión sobre el diseño de la interacción entre el humano y las aplicaciones gráficas.

Una vez hecha dicha reflexión, se procede al estudio de la aproximación de curvas arbitrarias por medio de segmentos cúbicos paramétricos. Y luego de la presentación de la teoría matemática implicada, se presentan cinco implementaciones sencillas. Después se hace una breve mención de superficies paramétricas.

Hacia el final del contenido principal del libro, se presenta una descripción de las estructuras de datos utilizables para representar mallas poligonales, que representan la técnica usada para modelar superficies arbitrarias en todo el software de modelación tridimensional.

Finalmente se hace una introducción a la graficación por medio de técnicas fractales que incluye implementación de aplicaciones de dibujo fractal.

Por último, pero no menos importante, se incluyen algunos temas no relacionados directamente con la graficación por computadora, pero sí relevantes para facilitar la lectura y comprensión de algunos temas y códigos fuente incluídos. Entre ellos, instrucciones sencillas y concisas para compilar proyectos en lenguaje C estándar distribuidos en varios archivos fuente; y un brevísimo resumen de notación UML para diagramas de clases y diagramas de objeto.

Lo que queda fuera

Como se dijo antes, la graficación por computadora es una de las grandes áreas de las ciencias de la computación, razón por la cual, cualquier obra que pretenda cubrirla toda, se quedará corta. En el caso de este libro, estos son los temas que no se cubren en la versión actual de la obra:

- Temas avanzados de SDL
- Interfaces gráficas de usuario (GUI)
- Modelado de sólidos
- Renderización de texto
- Operaciones de bajo nivel (en ensamblador, por ejemplo)
- Manipulación de mapas de bits
- Hardware gráfico
- Iluminación, sombreado y texturas

- Detalles de instalación de las herramientas en ambientes Windows

Esto no significa que no sean temas interesantes para un computólogo, o para un científico. Pero esta edición simplemente no los contempla.

Lo que conviene saber

Antes de escrutar este libro, es recomendable tener una base aceptable en los siguientes temas:

- Operaciones básicas con matrices
- Álgebra vectorial y nociones de cálculo vectorial
- Operaciones básicas con números complejos
- Parametrización de curvas
- Sistemas de coordenadas rectangulares, cilíndricas y esféricas
- Estructuras de datos básicas (listas, pilas, colas, árboles, directorios de nodos, grafos, etc.)
- Nociones claras de Programación Estructurada y Programación Orientada a Objetos
- Conocimientos sólidos del lenguaje C estándar en ambiente GNU/Linux
- Conocimientos intermedios en lenguaje Python en ambiente GNU/Linux
- Conocimientos intermedios del lenguaje Java
- Destrezas elementales del uso de sistemas operativos GNU/Linux

Motivación al lector

En general, el autor desea expresar su deseo de seguir mejorando y ampliando, en la medida de lo posible, esta obra para beneficio de la sociedad salvadoreña. Y también desea invitar al apreciable lector a que le saque todo el provecho posible, que aplique sus aportes en todas las áreas posibles de su trayectoria profesional; y también que no lo considere una serie de afirmaciones incuestionables, sino una contribución al desarrollo tecnológico salvadoreño y a la independencia tecnológica de la región centroamericana.

Como toda obra académica de programación, o como la mayoría, los códigos (y programas) incluídos pretenden mantener simplicidad y claridad para no dificultar innecesariamente la comprensión, pero se recomienda hacer el ejercicio de optimizarlos.

Cambios en esta versión

En esta nueva versión (que se corresponde con la segunda edición), se han hecho muchos cambios a lo largo de todo el texto. Se han corregido diversos errores tipográficos en todos los capítulos, se han revisado todos los códigos fuente que ya estaban y las imágenes se han revisado en su resolución y distribución. El tamaño de las páginas también se ha cambiado al tamaño carta, que es el estándar de facto en El Salvador.

Todos los bloques de código fuente se han resaltado para su más cómoda y más fácil lectura.

El capítulo 1 se ha ampliado en el tema de la instalación de SDL. Los ejemplos básicos fueron revisados y reordendados.

El capítulo 2 incluye ahora una sección especial sobre el ciclo de interacción de aplicaciones gráficas interactivas.

El capítulo 9 incluye mucho nuevo material, incluyendo nuevas imágenes, más algoritmos, más profundidad y nuevos programas de ejemplo.

El capítulo 10 también cuenta con nuevas imágenes, hechas con Maxima. El contenido del 12 fue revisado y cuenta con nuevas imágenes.

Y finalmente, se ha integrado la biblioteca `pygame` para Python en la primera parte del texto, en paralelo con `SDL` para C.

Índice general

Índice de figuras

Parte I

Graficación por computadora

1 Introducción a SDL y PyGAME

SDL, *Simple DirectMedia Layer*, es una biblioteca de funciones multimedia programada en lenguaje C estándar, disponible para usarse en UNIX / GNU/Linux, en Windows, en MacOS y en muchas otras plataformas. También existe una considerable cantidad de bibliotecas derivadas de ella para otros lenguajes de programación. Para tener una descripción más completa, refiérase al sitio oficial: www.libsdl.org.

1.1. Instalación de las bibliotecas para C

Al igual que con muchas otras bibliotecas de funciones de lenguaje C, la expresión "Instalar SDL" tiene dos significados: El primero es instalar las bibliotecas compiladas que contienen el código objeto de las funciones de SDL que los demás programas ya compilados pueden utilizar; y el segundo es, instalar los archivos de cabecera necesarios para la compilación de código nuevo que use las funciones de SDL.

Los usuarios de programas hechos con SDL (juegos, emuladores, etc.) sólo necesitan una instalación del primer tipo, y se dice "instalación de la biblioteca de ejecución" o "instalación de los paquetes de ejecución". Los programadores necesitamos de la segunda, que se dice "instalación de la biblioteca de desarrollo" o "instalación de los paquetes de desarrollo".

1.1.1. Verificar si ya están instalados los paquetes de desarrollo

SDL es una biblioteca muy utilizada en el mundo de los juegos de computadora y los emuladores, especialmente si son Software Libre, por lo que es muy probable que en una instalación típica de una distribución de GNU/Linux para escritorio, ya estén instalados los paquetes de ejecución. Así que es probable, que únicamente sea necesario instalar los paquetes de desarrollo.

Para averiguar si ya se tienen instalados los paquetes de desarrollo, basta con ejecutar la siguiente instrucción:

```
$ sdl-config --version
```

Si devuelve algo como:

```
1.2.12
```

significa que la biblioteca de desarrollo está instalada y el texto devuelto indica la versión de la biblioteca en esa máquina. Pero si devuelve algo como:
`bash: sdl-config: orden no encontrada`
significa que no está instalada la biblioteca de desarrollo, aunque tal vez sí la de ejecución.

1.1.2. Diseño modular de SDL

SDL consiste de una única biblioteca de funciones base, pero existen otras bibliotecas de funciones adicionales, desarrolladas por la amplia comunidad de usuarios/programadores de SDL.

Aunque la biblioteca base incluye lo necesario para desarrollar aplicaciones de todo tipo, conviene instalar además esas otras bibliotecas porque extienden su funcionalidad, permitiendo desarrollar aplicaciones más rápidamente.

Por ejemplo, se recomienda instalar las siguientes bibliotecas:

sdl-gfx Incluye funciones de dibujo de primitivas gráficas[1] huecas y rellenas, con o sin antialiasing[2]. Se le conoce también como *SDL Graphics Efects Primitives*.

sdl-image Incluye la funcionalidad de abrir un amplio grupo de formatos de imágenes desde archivo (la biblioteca base es muy limitada en este aspecto, porque sólo puede abrir archivos `*.bmp`).

sdl-sound Permite reproducir sonido en diversos formatos.

sdl-mixer Permite manipular (*mezclar*) sonido en diversos formatos.

sdl-net Permite la programación de aplicaciones con acceso a redes de computadoras.

sdl-pango Manipulación y renderización de fuentes con la biblioteca Pango (que es la biblioteca de renderización de texto de GTK+).

sdl-ttp Manipulación y renderización de fuentes TrueType.

sdl-stretch Permite manipulación bidimensional de mapas de bits.

Hay muchos otros módulos/bibliotecas desarrollados por la comunidad, pero estos son los más portables y populares. Los demás habrá que evaluarlos a la luz de su portabilidad (si es que ese factor es relevante) y la funcionalidad que aportan.

1.1.3. Instalación en distribuciones basadas en Debian

Para instalar todos los paquetes de desarrollo, ejecutar la instrucción:
`# apt-get install libsdl*-dev`

[1] Ver capítulo 3
[2] http://es.wikipedia.org/wiki/Antialiasing
http://en.wikipedia.org/wiki/Anti-aliasing

Con esto se instalarán todos los paquetes de desarrollo (*development*) que a su vez, tienen como dependencias a los paquetes de ejecución.

Si no se desean instalar todos los paquetes de desarrollo, se pueden consultar todos los paquetes que contengan el texto "`libsdl`" con el comando:
`$ aptitude search libsdl`
e instalar sólo aquellos que se requieran.

El comando anterior muestra "banderas" para cada paquete, y en particular, la bandera "`i`" indica que el paquete listado ya está instalado. Si en lugar de dicha letra, aparece una "`p`" o una "`v`" significa que el paquete no está instalado.

1.1.4. Instalación en openSuSE

La distribución OpenSuSE tiene una herramienta de configuración global llamada YaST que incluye una interfaz para instalar paquetes. Esta aplicación tiene interface de texto con ncurses, pero tiene una hermana con interfaz de alto nivel, YaST2.

Para instalar SDL, invocar el módulo de instalación:
`# yast2 --install &`
o
`# yast --install`
Hay que esperar a que se descargue y actualize la información de los repositorios en línea y entonces hay que buscar en la interfaz del YaST / YaST2 los paquetes del tipo `libSDL*-devel` e instalarlos.

También se pueden instalar los paquetes desde la línea de comandos. Para hacerlo, se utiliza el comando `zypper`:
`# zypper install libSDL*-devel`

Si no se desean instalar todos los paquetes de desarrollo, se pueden consultar todos los paquetes que contengan el texto "`libSDL`" con el comando:
`$ zypper search libSDL`
e instalar sólo aquellos que se requieran.

El comando anterior muestra, para cada paquete, la bandera "`i`" si el paquete listado ya está instalado y no muestra nada si no lo está.

1.1.5. Instalación en Fedora y derivados de RedHat

Para instalar todos los paquetes de desarrollo, ejecutar la instrucción:
`# yum install SDL*-devel`

Si no se desean instalar todos los paquetes de desarrollo, se pueden consultar todos los paquetes que contengan el texto "`libsdl`" con el comando:
`$ yum search libsdl`
e instalar sólo aquellos que se requieran.

1.1.6. Otras distribuciones

Para lograr instalar con éxito los paquetes de desarrollo de SDL (y sus dependencias), se recomienda usar las interfaces (gráficas o de linea de comandos) para la instalación desde los repositorios de cada distribución. Alternativamente se pueden descargar los instaladores del sitio de SDL www.libsdl.org en diversos formatos, y además, también se pueden descargar los fuentes para compilar los paquetes, *in situ*.

1.2. Compilación de los programas que usan SDL en C

Si el programa está en un sólo archivo fuente (y no utiliza módulos adicionales al base), se procede de la siguietne manera:

```
$ gcc archivo_fuente.c [-o archivo_objeto] $(sdl-config --cflags --libs)
```

o

```
$ gcc archivo_fuente.c [-o archivo_objeto] `sdl-config --cflags --libs`
```

Valga la aclaración que la instrucción

```
$ sdl-config --cflags --libs
```

imprime los parámetros necesarios para que el programa `gcc` pueda *enlazar* correctamente el código máquina del programa compilado.

1.3. Ejemplos básicos en SDL

A continuación se presenta una serie de ejemplos elementales a manera de introducción a la programación con SDL en lenguaje C estándar. No pretenden ser exhaustivos, sino solamente introductorios. Se asume que el lector tiene buen dominio del lenguaje C estándar y que tiene buenas prácticas autodidactas.

1.3.1. Inicialización básica del sistema de video (el *Hola Mundo* de SDL):

Listing 1.1: Hola Mundo en SDL

```
1 /* c01/ejemplo-01.c
```

```
2  Hola Mundo en SDL
3  */
4  #include <SDL/SDL.h>
5  #define ANCHO 323
6  #define ALTO  400
7  #define PROFUNDIDAD_COLOR 16
8
9  int main(void){
10     SDL_Surface *pantalla = NULL;
11     SDL_Surface *imagen = NULL;
12
13     //Inicializar el subsistema principal de SDL, el de video
14     if (SDL_Init(SDL_INIT_VIDEO) < 0){
15         printf("Error al iniciar SDL: %s\n", SDL_GetError());
16         exit(1);
17     }
18     /*
19      * Fuerza la ejecución de la función 'SDL_Quit()'
20      * aún en caso de salida con error, al llamar
21      * a 'exit(int)' o al retornar de la función 'main'.
22      * */
23     atexit(SDL_Quit);
24
25     //Abrir la ventana para gráficos
26     pantalla = SDL_SetVideoMode(ANCHO, ALTO, PROFUNDIDAD_COLOR,
           SDL_SWSURFACE);
27     if (pantalla == NULL){
28         printf("Error al inicializar el modo de video: '%s'\n",
               SDL_GetError());
29         exit(1);
30     }
31
32     //Cambiar el título de la ventana
33     SDL_WM_SetCaption("¡Hola mundo!", NULL);
34
35     //Cargar una imagen:
36     imagen = SDL_LoadBMP("logo_uca.bmp");
37     if (imagen == NULL){
38         printf("Error al cargar la imagen: '%s'\n", SDL_GetError());
39         exit(1);
40     }
41
42     //Copiar la imagen al buffer temporal en 'pantalla':
43     SDL_BlitSurface(imagen, NULL, pantalla, NULL);
44
45     //Volcar el buffer a la memoria de video:
46     SDL_Flip(pantalla);
47
48     //Esperar un tiempo de 5000 milisegundos
49     SDL_Delay(5000);
```

```
50
51      //y luego apagar el sistema de SDL con 'SDL_Quit()',
52      //en este caso, invocada automáticamente con el return.
53      return 0;
54  }
```

1.3.2. Inicialización de los subsistemas

SDL cuenta de forma nativa con 5 subsistemas. Las constantes para identificarlos son:

SDL_INIT_TIMER Subsistema de temporizador.

SDL_INIT_AUDIO Subsistemas de carga y reproducción de pistas de audio.

SDL_INIT_VIDEO Subsistema de video.

SDL_INIT_CDROM Subsistema de control y reproducción de cdrom de audio.

SDL_INIT_JOYSTICK Subsistema de interacción con palancas de mando o controles de juego.

Listing 1.2: Inicialización de subsistemas en SDL

```
1  /* c01/ejemplo-02.c
2  Inicialización de subsistemas en SDL
3  */
4  #include <SDL/SDL.h>
5
6  int main(void){
7      SDL_Surface *pantalla = NULL;
8
9      /* Los subsistemas disponibles son:
10      * SDL_INIT_TIMER
11      * SDL_INIT_AUDIO
12      * SDL_INIT_VIDEO
13      * SDL_INIT_CDROM
14      * SDL_INIT_JOYSTICK
15      * SDL_INIT_EVERYTHING
16     * */
17     if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_CDROM | SDL_INIT_AUDIO) < 0){
18         printf("Error␣al␣iniciar␣SDL␣con␣los␣subsitemas␣de␣video,␣de␣
               unidad␣óptica␣y␣de␣sonido:␣%s\n", SDL_GetError());
19         exit(1);
20     }
21     atexit(SDL_Quit);
22
23     //inicializar el subsistema de palanca de mandos:
24     SDL_InitSubSystem(SDL_INIT_JOYSTICK);
25
26     //apaga los subsistemas de video y de audio
```

```
27      SDL_QuitSubSystem(SDL_INIT_VIDEO | SDL_INIT_CDROM);
28
29      if(SDL_WasInit(SDL_INIT_VIDEO))
30          printf("El video está encendido\n");
31      else
32          printf("El video está apagado\n");
33
34      if(SDL_WasInit(SDL_INIT_CDROM))
35          printf("El cdrom está encendido\n");
36      else
37          printf("El cdrom está apagado\n");
38
39      if(SDL_WasInit(SDL_INIT_AUDIO))
40          printf("El audio está encendido\n");
41      else
42          printf("El audio está apagado\n");
43
44      if(SDL_WasInit(SDL_INIT_JOYSTICK))
45          printf("El joystick está encendido\n");
46      else
47          printf("El joystick está apagado\n");
48
49      //Apagar todos los subsistemas de SDL automáticamente
50      return 0;
51  }
```

1.3.3. Modos de video

Los modos de video disponibles son:

`SDL_SWSURFACE` Usa la memoria de sistema

`SDL_HWSURFACE` Usa la memoria de video

`SDL_DOUBLEBUF` Activa doble buffer en la memoria de video

`SDL_FULLSCREEN` Crea una superficie de dibujo que ocupa toda la pantalla.

`SDL_OPENGL` Crea una superficie renderizable con opengl.

`SDL_RESIZABLE` Crea una ventana de dibujo redimensionable.

`SDL_NOFRAME` Crea una ventana de dibujo sin borde.

Listing 1.3: Modos de video

```
1   /* c01/ejemplo-03.c
2   Modos de video en SDL
3   */
4   #include <SDL/SDL.h>
5   #define ANCHO 400
```

```
6  #define ALTO  400
7
8  int main(void){
9      SDL_Surface *pantalla = NULL;
10     SDL_Surface *imagen = NULL;
11
12     if (SDL_Init(SDL_INIT_VIDEO) < 0){
13         printf("Error al iniciar SDL con el subsitemas de video: %s\n",
               SDL_GetError());
14         exit(1);
15     }
16     atexit(SDL_Quit);
17
18     /* Modos disponibles:
19      * SDL_SWSURFACE usa la memoria de sistema
20      * SDL_HWSURFACE usa la memoria de video
21      * SDL_DOUBLEBUF activa doble buffer en la memoria de video
22      * SDL_FULLSCREEN
23      * SDL_OPENGL
24      * SDL_RESIZABLE
25      * SDL_NOFRAME
26     * */
27
28     // Ventana normal de tamaño fijo:
29     pantalla = SDL_SetVideoMode(ANCHO, ALTO, 0,  SDL_SWSURFACE);
30
31     // Ventana normal de tamaño variable:
32     //pantalla = SDL_SetVideoMode(ANCHO, ALTO, 0,  SDL_SWSURFACE|
           SDL_RESIZABLE);
33
34     // Ventana normal de tamaño fijo maximizada (tamaño igual a pantalla
           completa)
35     //pantalla = SDL_SetVideoMode(    0,    0, 0,  SDL_SWSURFACE);
36
37     // Ventana sin borde de tamaño fijo
38     //pantalla = SDL_SetVideoMode(ANCHO, ALTO, 0,  SDL_SWSURFACE|
           SDL_NOFRAME);
39
40     // Pantalla completa con resolución fija
41     //pantalla = SDL_SetVideoMode(ANCHO, ALTO, 0,  SDL_SWSURFACE|
           SDL_FULLSCREEN);
42
43     // Pantalla completa con resolución máxima
44     //pantalla = SDL_SetVideoMode(    0,    0, 0,  SDL_SWSURFACE|
           SDL_FULLSCREEN);
45
46     if (pantalla == NULL){
47         printf("Error al inicializar el modo de video: '%s'\n",
               SDL_GetError());
48         exit(1);
```

```
49      }
50
51      printf("Tamaño␣de␣la␣pantalla:␣%dx%d\n", pantalla->w, pantalla->h);
52
53      //Cargar una imagen:
54      imagen = SDL_LoadBMP("logo_uca.bmp");
55      if (imagen == NULL){
56          printf("Error␣al␣cargar␣la␣imagen:␣'%s'\n", SDL_GetError());
57          exit(1);
58      }
59
60      //Copiar la imagen al buffer temporal en 'pantalla':
61      SDL_BlitSurface(imagen, NULL, pantalla, NULL);
62
63      //Volcar el buffer a la memoria de video:
64      SDL_Flip(pantalla);
65
66      //Espera un tiempo de 10 segundos
67      SDL_Delay(10000);
68
69      //Apaga manualmente los subsistemas de SDL
70      SDL_Quit();
71
72      return 0;
73  }
```

1.3.4. Eventos de ratón

Listing 1.4: Eventos de ratón

```
1   /* c01/ejemplo-04.c
2   Eventos del ratón en SDL
3   */
4   #include <SDL/SDL.h>
5   #define ANCHO 400
6   #define ALTO  400
7
8   char *nombreBotones[] = {"izquierdo", "medio", "derecho", "rueda␣arriba",
        "rueda␣abajo"};
9
10  char *nombreBoton(Uint8 boton){
11      switch(boton){
12          case SDL_BUTTON_LEFT:
13              return nombreBotones[0];
14          case SDL_BUTTON_MIDDLE:
15              return nombreBotones[1];
16          case SDL_BUTTON_RIGHT:
17              return nombreBotones[2];
```

```
18          case SDL_BUTTON_WHEELUP:
19              return nombreBotones[3];
20          case SDL_BUTTON_WHEELDOWN:
21              return nombreBotones[4];
22      }
23  }
24
25  int main(void){
26      SDL_Surface *pantalla = NULL;
27      SDL_Event evento;
28      int corriendo = 1;
29
30      if (SDL_Init(SDL_INIT_VIDEO) < 0){
31          printf("Error al iniciar SDL: %s\n", SDL_GetError());
32          exit(1);
33      }
34      atexit(SDL_Quit);
35
36      pantalla = SDL_SetVideoMode(ANCHO, ALTO, 0,  SDL_SWSURFACE);
37      if (pantalla == NULL){
38          printf("Error al inicializar el modo de video: '%s'\n",
                SDL_GetError());
39          exit(1);
40      }
41
42      SDL_WM_SetCaption("Prueba de ratón, mueva el ratón dentro de la
            ventana", NULL);
43
44      while(corriendo) {
45          while(SDL_PollEvent(&evento)) {
46              switch(evento.type){
47                  /*
48                      Consultar el archivo:
49                      /usr/include/SDL/SDL_mouse.h
50                      Allí se encuentra la declaración de los botones
51                      en forma de constantes
52                  */
53                  case SDL_MOUSEBUTTONDOWN:
54                      printf("Botón de ratón '%s' pulsado en (%d,%d)\n",
55                          nombreBoton(evento.button.button), evento.button.
                                x, evento.button.y);
56                      break;
57
58                  case SDL_MOUSEBUTTONUP:
59                      printf("Botón de ratón '%s' liberado en (%d,%d)\n",
60                          nombreBoton(evento.button.button), evento.button.
                                x, evento.button.y);
61                      break;
62
63                  case SDL_MOUSEMOTION:
```

```
64                      printf("El␣ratón␣se␣movió␣%d,%d␣pixeles␣hasta␣(%d,%d)
                            \n",
65                          evento.motion.xrel, evento.motion.yrel, evento.
                                motion.x, evento.motion.y);
66                      break;
67
68                  case SDL_QUIT:
69                      corriendo = 0;
70                      break;
71              }
72          }
73      }
74
75      return 0;
76  }
```

1.3.5. Eventos de teclado

Listing 1.5: Eventos de teclado

```
1   /* c01/ejemplo-05.c
2   Eventos del teclado en SDL
3   */
4   #include <SDL/SDL.h>
5   #include <stdio.h>
6
7   #define ANCHO 640
8   #define ALTO  480
9
10  void mostrarEstado(SDL_KeyboardEvent *tecla){
11      if (tecla->type == SDL_KEYUP)
12          printf("SOLTADA:␣␣␣␣");
13      else //SDL_KEYDOWN
14          printf("PRESIONADA:␣");
15  }
16
17  void mostrarModificadores(SDL_KeyboardEvent *tecla){
18      SDLMod modificador = tecla->keysym.mod;
19      if( modificador & KMOD_NUM ) printf( "NUMLOCK␣" );
20      if( modificador & KMOD_CAPS ) printf( "CAPSLOCK␣" );
21      if( modificador & KMOD_MODE ) printf( "MODE␣" );
22      if( modificador & KMOD_LCTRL ) printf( "LCTRL␣" );
23      if( modificador & KMOD_RCTRL ) printf( "RCTRL␣" );
24      if( modificador & KMOD_LSHIFT ) printf( "LSHIFT␣" );
25      if( modificador & KMOD_RSHIFT ) printf( "RSHIFT␣" );
26      if( modificador & KMOD_LALT ) printf( "LALT␣" );
27      if( modificador & KMOD_RALT ) printf( "RALT␣" );
28      if( modificador & KMOD_LMETA ) printf( "LMETA␣" );
```

```
29      if( modificador & KMOD_RMETA ) printf( "RMETA " );
30  }
31
32  /*
33      Consultar el archivo:
34      /usr/include/SDL/SDL_keysym.h
35      Allí se encuentra la declaración de las teclas
36      en forma de constantes
37  */
38  void mostrarTecla(SDL_KeyboardEvent *tecla){
39      printf( "Código: %d, Nombre: %s\n", tecla->keysym.sym,
40          SDL_GetKeyName(tecla->keysym.sym));
41  }
42
43  int main(void){
44      SDL_Surface *pantalla = NULL;
45      SDL_Event evento;
46      int corriendo = 1;
47
48      if(SDL_Init(SDL_INIT_VIDEO) < 0 ){
49          fprintf(stderr, "No se puede iniciar SDL: %s\n", SDL_GetError());
50          exit(1);
51      }
52      atexit(SDL_Quit);
53
54      pantalla = SDL_SetVideoMode(ANCHO, ALTO, 0, SDL_SWSURFACE);
55      if(pantalla == NULL){
56          fprintf(stderr, "No se puede establecer el modo de video %dx%d: %
                s\n",
57              ANCHO, ALTO, SDL_GetError());
58          exit(1);
59      }
60
61      SDL_WM_SetCaption("Prueba de teclado, presione las teclas", NULL);
62
63
64      while(corriendo) {
65          while(SDL_PollEvent(&evento)) {
66              switch(evento.type){
67                  case SDL_KEYDOWN:
68                  case SDL_KEYUP:
69                      mostrarEstado(&evento.key);
70                      mostrarModificadores(&evento.key);
71                      mostrarTecla(&evento.key);
72                      break;
73
74                  case SDL_QUIT:
75                      corriendo = 0;
76                      break;
77              }
```

```
78          }
79      }
80
81      return 0;
82  }
```

1.3.6. Redimensionamiento de la ventana

Listing 1.6: Redimensionamiento

```
1   /* c01/ejemplo-06.c
2    * Redimensionamiento de la ventana en SDL
3   */
4   #include <SDL/SDL.h>
5
6   int main(void){
7       SDL_Surface *pantalla = NULL;
8       SDL_Event evento;
9       int ANCHO = 400, ALTO = 400;
10      int corriendo = 1;
11
12      if (SDL_Init(SDL_INIT_VIDEO) < 0){
13          printf("Error al iniciar SDL: %s\n", SDL_GetError());
14          exit(1);
15      }
16      atexit(SDL_Quit);
17
18      pantalla = SDL_SetVideoMode(ANCHO, ALTO, 0,  SDL_SWSURFACE|
            SDL_RESIZABLE);
19      if (pantalla == NULL){
20          printf("Error al inicializar el modo de video: '%s'\n",
                SDL_GetError());
21          exit(1);
22      }
23
24      SDL_WM_SetCaption("Prueba de redimensionamiento", NULL);
25
26      while(corriendo) {
27
28          while(SDL_PollEvent(&evento)) {
29              switch(evento.type){
30                  /*
31                      Consultar el archivo:
32                      /usr/include/SDL/SDL_events.h
33                      Allí se encuentra la definición de la unión '
                            SDL_Event'.
34                  */
35                  case SDL_VIDEORESIZE:
```

```
36                      //Redimensionar la pantalla, no hace falta liberar la
                            anterior.
37                      pantalla = SDL_SetVideoMode(evento.resize.w, evento.
                            resize.h, PROFUNDIDAD_COLOR, SDL_SWSURFACE|
                            SDL_RESIZABLE);
38                      if(pantalla == NULL){
39                          printf("Error al redimensionar la pantalla: '%s'\
                                n", SDL_GetError());
40                          exit(1);
41                      }
42                      printf("La ventana se ha redimensionado de %dx%d
                            pixeles a %dx%d\n",
43                          ANCHO, ALTO, evento.resize.w, evento.resize.h);
44                      ANCHO = evento.resize.w; ALTO = evento.resize.h;
45                      //Aquí habría que redibujar lo que se estaba
                            mostrando
46                      //ya que 'pantalla' aparece borrada (en negro).
47                      break;
48
49                  case SDL_QUIT:
50                      corriendo = 0;
51                      break;
52              }
53          }
54      }
55
56      return 0;
57  }
```

1.3.7. Facilitar la compilación

Para facilitar el proceso de compilación para programas que usen SDL que consistan de un solo archivo fuente (sin contar el archivo de cabecera), puede crearse en el mismo directorio el siguiente archivo `Makefile`:

Listing 1.7: Makefile para programas SDL de un archivo fuente

```
1  # c01/Makefile
2  LDFLAGS = $(shell sdl-config --cflags)
3  LDLIBS  = $(shell sdl-config --libs)
4  CC = gcc
5
6  #Existiendo este archivo en el directorio,
7  #se pueden compilar archivos *.c que usen SDL
8  #con sólo ejecutar:
9  #'$ make <ejecutable>'
10 #(claro que tiene que existir un archivo '<ejecutable>.c')
11
```

```
12 #Si este archivo Makefile no existiera, para compilar habría que
       ejecutar:
13 #'$ gcc -o <ejecutable> <ejecutable>.c $(sdl-config --libs --cflags)'
```

Con este archivo en el mismo directorio, podemos usar la utilería `make` de los sistemas basados en Unix. De tal manera que sólo haya que ejecutar la siguiente instrucción para compilar un programa fuente:

```
$ make ejemplo-01
```

Con esta instrucción y el archivo mencionado, se buscará el archivo `ejemplo-01.c` y se compilará en el archivo `ejemplo-01`.

1.3.8. Compilación desde varios archivos fuente

Lo anterior no aplica para un programa que consista de varios archivos fuente (sin contar sus archivos de cabecera). Sin embargo, siempre nos podemos apoyar en el comando `make` para automatizar el proceso de compilación. Veamos un ejemplo sencillo (se recomienda echarle un vistazo al capítulo 13 en la página 279 antes de continuar aquí):

Listing 1.8: Archivo de cabecera de otro archivo fuente

```
1 /* c01/ejemplo-07/otro.h
2  * */
3 #include <stdio.h>
4 #define CONSTANTE 5
5 void funcionOtro(int entero, FILE *f);
```

Listing 1.9: Otro archivo fuente

```
1 /* c01/ejemplo-07/otro.c
2  * */
3 #include "otro.h"
4
5 void funcionOtro(int entero, FILE *f){
6     fprintf(f, "Código␣en␣'otro.c',\nparámetro:␣%d\nconstante:␣%d\n",
          entero, CONSTANTE);
7 }
```

A continuación se presentan los archivos fuente que continen la implementanción de un par de funciones de primitivas gráficas:

Listing 1.10: Archivo de cabecera principal

```
1 /* c01/ejemplo-07/primitivas.h
2  * */
3 #include <SDL/SDL.h>
```

```
4
5  /*
6   * Enciende el pixel (x, y) con el color dado.
7   *
8   * SDL no contiene de forma nativa una función para
9   * encender pixeles, a diferencia de otras bibliotecas gráficas.
10  * Esto en sí mismo es un tema de discución, pero típicamente
11  * el código de esta función aplica en SDL.
12  */
13 void ponerPixel(SDL_Surface *s, int x, int y, Uint32 color);
14
15 /*
16     Dibuja un cuadrado vacío dado por 'rectangulo'
17 */
18 void dibujar_rectangulo(SDL_Surface *s, SDL_Rect rectangulo, Uint32 color
       );
```

Listing 1.11: Código fuente de primitivas gráficas

```
1  /* c01/ejemplo-07/primitivas.c
2   * */
3  #include "primitivas.h"
4
5  void dibujar_rectangulo(SDL_Surface *s, SDL_Rect rectangulo, Uint32 color
       ){
6      int i;
7      for(i=rectangulo.x; i<rectangulo.x+rectangulo.w; i++){
8          ponerPixel(s, i, rectangulo.y,              color);
9          ponerPixel(s, i, rectangulo.y+rectangulo.h-1, color);
10     }
11     for(i=rectangulo.y; i<rectangulo.y+rectangulo.h; i++){
12         ponerPixel(s, rectangulo.x,              i, color);
13         ponerPixel(s, rectangulo.x+rectangulo.w-1, i, color);
14     }
15 }
16
17 void ponerPixel(SDL_Surface *surface, int x, int y, Uint32 color){
18     int bpp = surface->format->BytesPerPixel;
19     // Aquí p es la dirección del pixel al que queremos ponerle color
20     Uint8 *p = (Uint8 *)surface->pixels + y * surface->pitch + x * bpp;
21
22     if((x>surface->w)||(y>surface->h)||(x<0)||(y<0)) return;
23
24     switch(bpp) {
25     case 1:
26         *p = color;
27         break;
28
29     case 2:
30         *(Uint16 *)p = color;
```

```
31          break;
32
33      case 3:
34          if(SDL_BYTEORDER == SDL_BIG_ENDIAN) {
35              p[0] = (color >> 16) & 0xff;
36              p[1] = (color >> 8) & 0xff;
37              p[2] = color & 0xff;
38          }
39          else {
40              p[0] = color & 0xff;
41              p[1] = (color >> 8) & 0xff;
42              p[2] = (color >> 16) & 0xff;
43          }
44          break;
45
46      case 4:
47          *(Uint32 *)p = color;
48          break;
49      }
50  }
```

Listing 1.12: Código principal del ejemplo

```
1   /* c01/ejemplo-07/main.c
2    * Programa principal
3    * */
4   #include "otro.h"
5   #include "primitivas.h"
6   #include <stdio.h>
7
8   int main(int argc, char *argv[]){
9       SDL_Surface *pantalla = NULL;
10      SDL_Event evento;
11      Uint32 color_fondo, color1, color2;
12      SDL_Rect rect1, rect2;
13      int ANCHO = 400, ALTO = 400;
14      int corriendo = 1;
15
16      if (SDL_Init(SDL_INIT_VIDEO) < 0){
17          printf("Error␣al␣iniciar␣SDL:␣%s\n", SDL_GetError());
18          exit(1);
19      }
20      atexit(SDL_Quit);
21
22      pantalla = SDL_SetVideoMode(ANCHO, ALTO, 0,  SDL_SWSURFACE|
            SDL_RESIZABLE);
23      if (pantalla == NULL){
24          printf("Error␣al␣inicializar␣el␣modo␣de␣video:␣'%s'\n",
                SDL_GetError());
25          exit(1);
```

```
26      }
27
28      //Esta es la forma básica de construir un color en SDL:
29      color_fondo = SDL_MapRGB (pantalla->format,0,0,0);
30      color1 = SDL_MapRGB (pantalla->format,255,0,0);
31      color2 = SDL_MapRGB (pantalla->format,0,255,0);
32
33      SDL_WM_SetCaption("Código␣distribuido", NULL);
34
35      //Hacer primer dibujo
36      //Dibujar un rectángulo rojo y otro verde
37      rect1.x = 0;
38      rect1.y = 0;
39      rect1.w = ANCHO/2;
40      rect1.h =ALTO/2;
41      dibujar_rectangulo(pantalla, rect1, color1);
42      rect2.x = ANCHO/2;
43      rect2.y = ALTO/2;
44      rect2.w = ANCHO/2-1;
45      rect2.h =ALTO/2-1;
46      dibujar_rectangulo(pantalla, rect2, color2);
47
48      //Volcar el buffer en la pantalla
49      SDL_Flip (pantalla);
50
51      //llamar una función que se encuentra en otro archivo de código
52      funcionOtro(10, stdout);
53
54      while(corriendo) {
55          while(SDL_PollEvent(&evento)) {
56              switch(evento.type){
57                  case SDL_VIDEORESIZE: {
58                      ANCHO = evento.resize.w;
59                      ALTO = evento.resize.h;
60
61                      //Redimensionar la pantalla, no hace falta liberar la
                            anterior.
62                      pantalla =
63                              SDL_SetVideoMode(ANCHO, ALTO, 0,
64                      SDL_SWSURFACE|SDL_RESIZABLE);
65                      if(pantalla == NULL){
66                          printf("Error␣al␣redimensionar␣la␣pantalla:␣'%s'\
                                n", SDL_GetError());
67                          exit(1);
68                      }
69
70                      //La pantalla nueva aparece en blanco
71                      //o mejor dicho, en negro,
72                      //por lo que hay que dibujar de nuevo:
73                      rect1.x = 0;
```

```
74                      rect1.y = 0;
75                      rect1.w = ANCHO/2;
76                      rect1.h =ALTO/2;
77                      dibujar_rectangulo(pantalla, rect1, color1);
78                      rect2.x = ANCHO/2;
79                      rect2.y = ALTO/2;
80                      rect2.w = ANCHO/2-1;
81                      rect2.h =ALTO/2-1;
82                      dibujar_rectangulo(pantalla, rect2, color2);
83
84                      //Vuelca el buffer en la pantalla:
85                      SDL_Flip (pantalla);
86                  }
87                  break;
88
89                  case SDL_QUIT:
90                      corriendo = 0;
91                  break;
92              }
93          }
94      }
95
96      return 0;
97  }
```

Finalmente, para poder compilar fácilmente todo este código, requerimos de un archivo `Makefile` como el siguiente:

Listing 1.13: Archivo Makefile para varios fuentes usando SDL

```
1  # c01/ejemplo-07/Makefile
2  LDFLAGS = $(shell sdl-config --cflags)
3  LDLIBS  = $(shell sdl-config --libs)
4  RM      = /bin/rm -f
5
6  #Esto indica que los siguientes identificadores,
7  #no son archivos, sino comandos de make:
8  .PHONY: limpiar
9  .PHONY: limpiartodo
10 .PHONY: all
11
12 #Nombre del programa ejecutable:
13 PROG = ejemplo-07
14
15 #Un '*.o' por cada '*.c'
16 OBJ = otro.o main.o primitivas.o
17
18 #Cuando se ejecuta '$ make', se ejecuta esto:
19 all: $(PROG) limpiar
20
```

```
21 #Esto compila todo el código y lo enlaza:
22 $(PROG): $(OBJ)
23         gcc -o $(PROG) $(OBJ) $(LDLIBS) $(LDFLAGS)
24
25 #Borra todos los archivos intermedios y de copia de seguridad
26 limpiar:
27         $(RM) *~ $(OBJ)
28
29 #Borra todos los archivos intermedios, de copia de seguridad
30 # y el programa ejecutable, si es que existe
31 limpiartodo:
32         make limpiar
33         $(RM) $(PROG)
```

La función `ponerPixel` implementada en el archivo 1.11 es la forma típica de "encender" o "colorear" pixeles en SDL.

1.4. Dibujo de primitivas gráficas con SDL

Desafortunadamente, la biblioteca base de SDL no incluye funciones para encender pixeles ni para dibujar líneas, rectángulos, círculos, polígonos, etc (es decir, primitivas gráficas[3]). Esto es debido a que sus desarrolladores pretenden que se mantenga como una biblioteca multimedia de bajo nivel para que sea ampliada al gusto (y habilidad) del programador que la use, dando la libertad/responsabilidad de construirlas uno mismo.

Pero por cuestiones prácticas, en este libro vamos a preferir dibujar primitivas gráficas con la ayuda de la biblioteca *gfxPrimitives* del paquete `sdl-gfx` mencionada en la subsección 1.1.2 en la página 24.

Su uso se ilustra en el siguiente programa:

Listing 1.14: Ejemplo de uso de gfxPrimitives

```
1  /* c01/ejemplo-08/primitivas.c
2   *
3   */
4  #include <SDL/SDL.h>
5  #include <SDL/SDL_gfxPrimitives.h>
6  #include <stdio.h>
7
8  #define ANCHO 800
9  #define ALTO 640
10
11 Uint32 rojo, verde, blanco;
```

[3] La definición de *Primitiva Gráfica* aparece en el capítulo 3 en la página 81, pero no es otra cosa que líneas rectas, círculos, rectángulos, polígonos, etc.

```
12
13  /*
14      Las funciones ____Color de SDL_gfx,
15      indicadas en el archivo
16      /usr/include/SDL/SDL_gfxPrimitives.h
17      requieren el color en un entero de 32 bis
18      así: 0xRRGGBBAA, no acepta el color
19      en ningún otro formato.
20
21      Además, utiliza las transparencias por defecto,
22      siempre hay que especificar la opacidad del color.
23  */
24  Uint32 color(Uint8 r, Uint8 g, Uint8 b){
25      return
26          r << 24 |
27          g << 16 |
28          b << 8  |
29          255;     //este valor es la opacidad del color
30                   //y debe ser máxima para que sea sólido
31  }
32
33  /*
34      Ver todas las funciones disponibles
35      en el archivo de cabecera:
36      /usr/include/SDL/SDL_gfxPrimitives.h
37
38      Hay unas ____Color y otras _____RGBA.
39  */
40  void dibujar(SDL_Surface *pantalla){
41      //Borrar la pantalla
42      SDL_FillRect (pantalla, NULL, SDL_MapRGB (pantalla->format,0,0,0));
43      /*Las operaciones fuera de SDL_gfx deben ser tratadas con 'SDL_MapRGB
            ' y 'SDL_MapRGBA',
44      no con 'Uint32 color(Uint8,Uint8,Uint8)'*/
45
46      //Dibujar un rectángulo rojo y otro verde
47      boxColor(pantalla, 0,0,ANCHO/2, ALTO/2, rojo);
48      boxColor(pantalla, ANCHO/2, ALTO/2, ANCHO, ALTO, verde);
49
50      //Lineas (notar la diferencia visual entre ambas):
51      lineColor(pantalla, 0,0,ANCHO, ALTO , blanco);
52      aalineColor(pantalla, 0,ALTO,ANCHO, 0 , blanco);
53
54      //Línea horizontal amarilla
55      hlineColor(pantalla, 0, ANCHO, ALTO/2, color(255, 255, 0));
56
57      stringColor (pantalla, ANCHO/2,ALTO/2,"HOLA", color(255,0,255));
58      stringRGBA (pantalla, 3*ANCHO/4,ALTO/4,"BIENVENIDOS␣A␣SDL_gfx",
            255,255,255,255);
59  }
```

```
60
61 int main(int argc, char *argv[]){
62     SDL_Surface *pantalla = NULL;
63     SDL_Event evento;
64     int corriendo = 1;
65
66     if (SDL_Init(SDL_INIT_VIDEO) < 0){
67         printf("Error al iniciar SDL: %s\n", SDL_GetError());
68         exit(1);
69     }
70     atexit(SDL_Quit);
71
72     pantalla = SDL_SetVideoMode(ANCHO, ALTO, 0,  SDL_SWSURFACE);
73     if (pantalla == NULL){
74         printf("Error al inicializar el modo de video: '%s'\n",
               SDL_GetError());
75         exit(2);
76     }
77
78     rojo = color(255,0,0);
79     verde = color(0,255,0);
80     //verde = SDL_MapRGBA (pantalla->format,0,255,0, 255);  //ASÍ NO
           FUNCIONA
81     blanco = color(255, 255, 255);
82
83     SDL_WM_SetCaption("Ejemplo de gfx", NULL);
84
85     //Hacer primer dibujo
86     dibujar(pantalla);
87
88     //Volcar el buffer en la pantalla
89     SDL_Flip(pantalla);
90
91     //Esperar que se cierre la ventana
92     while(corriendo) {
93         while(SDL_PollEvent(&evento)) {
94             switch(evento.type){
95
96                 case SDL_QUIT:
97                     corriendo = 0;
98                 break;
99             }
100        }
101    }
102
103    return 0;
104 }
```

Listing 1.15: Makefile necesario para usar gfxPrimitives

```
# c01/ejemplo-08/Makefile

# Nótese que se incluye un parámetro adicional por gfx:
LINEA = $(shell sdl-config --cflags --libs) -lSDL_gfx

RM      = /bin/rm -f

.PHONY: limpiar
.PHONY: limpiartodo
.PHONY: all

PROG = gfx

OBJ = primitivas.o

all: $(PROG) limpiar

$(PROG): $(OBJ)
        gcc -o $(PROG) $(OBJ) $(LINEA)

limpiar:
        $(RM) *~ $(OBJ)

limpiartodo:
        make limpiar
        $(RM) $(PROG)
```

1.5. Instalación de PyGAME

PyGAME es un conjunto de módulos del lenguaje Python diseñados para programar juegos y programas multimedia. PyGAME agrega funcionalidad sobre la biblioteca SDL y se puede interpretar como SDL para Python.

En distribuciones basadas en Debian y openSuSE, el paquete en cuestión se llama *python-pygame*, por lo que basta con instalarlo, en Debian, con:
`# apt-get install python-pygame`
o en OpenSuSE con:
`# yast -i python-pygame` (para la interfaz de texto)
o
`# yast2 -i python-pygame` (para la interfaz gráfica de alto nivel)
o
`# zypper install python-pygame` (en línea de comandos)

En OpenSuse, existe también el paquete `python-pygame-doc` que instala documentación y ejemplos de PyGAME.

En distribuciones Fedora y derivados de RedHat, el paquete se llama simplemente *pygame* y se puede instalar con:

```
# yum install pygame
```

Al igual que con el caso de las bibliotecas de SDL, se recomienda usar las interfaces (gráficas o de linea de comandos) para la instalación desde los repositorios de cada distribución.

Si esto no es posible o simplemente no se desea, se pueden descargar los instaladores del sitio de PyGAME www.pygame.org en diversos formatos y diferentes versiones, y además, también se pueden descargar los archivos fuentes[4] para compilar los paquetes, *in situ*.

1.6. Ejemplos básicos en Python

Pygame tiene un diseño y un estilo muy parecido al típico de los módulos de Python. Por lo cual, hay marcadas diferencias en la forma de programar con SDL en C y con PyGAME en Python. Sin embargo, la nomenclatura de las funciones, constantes y "objetos" es muy parecida o igual a la de SDL en C.

1.6.1. Inicialización básica del sistema de video (el *Hola Mundo* de pygame):

Listing 1.16: Hola Mundo en Pygame

```
# coding: utf-8
'''c01/ejemplo-01.py
Hola Mundo en pygame
'''
import pygame

#Inicializar el Sistema de pygame
pygame.init()

tam = ancho, alto = 323, 400

#Abrir la ventana para gráficos
pantalla = pygame.display.set_mode(tam)

#Cambiar el título de la ventana
pygame.display.set_caption("¡Hola Mundo!")

#Cargar una imagen:
```

[4] El principal intérprete de Python y los paquetes de pygame están programados en lenguaje C estándar.

```
19 imagen = pygame.image.load("logo_uca.bmp")
20
21 #Mostrar la imagen:
22 pantalla.blit(imagen, (0,0))
23
24 #Volcar el buffer a la memoria de video:
25 pygame.display.flip()
26
27 #Esperar 5000 milisegundos
28 pygame.time.delay(5000)
29 #y terminar el programa.
30
31 #El cerrado de los subsistemas de pygame es automático
```

1.6.2. Inicialización de los subsistemas

Pygame cuenta con 5 subsistemas nativos. Estos son:

pygame.cdrom Módulo para control de cdrom de audio.

pygame.display Módulo de video.

pygame.font Módulo de renderización de fuentes TrueType.

pygame.joystick Módulo de interacción con palancas de mando o controles de juego.

pygame.mixer Módulo de carga y reproducción de pistas de audio.

Listing 1.17: Inicialización de subsistemas en Pygame

```
1  # coding: utf-8
2  '''c01/ejemplo-02.py
3  Inicialización␣de␣subsistemas␣en␣pygame.
4
5  Los␣subsistemas/módulos␣disponibles␣son:
6  .-␣pygame.cdrom
7  .-␣pygame.display
8  .-␣pygame.font
9  .-␣pygame.joystick
10 .-␣pygame.mixer
11 .-␣pygame.scap
12 '''
13 import pygame
14
15 def mensaje(nombre, valor_de_verdad):
16     '''Esta␣función␣simplemente␣sirve␣para␣no␣escribir␣la␣misma␣cadena␣
           varias␣veces.
17
18 ␣␣␣␣La␣combinación␣'and-or'␣en␣Python␣funciona␣como␣el␣operador␣ternario␣
       de␣C
```

```
    siempre y cuando el valor entre el 'and' y el 'or' se evalúe a
        verdadero.'''
        print("El subsistema de" + nombre + " " + (valor_de_verdad and "sí"
            or "no") + " está encendido")

'''La función 'init' inicializa todos los subsistemas
disponibles y devuelve el número de subsistemas
que pudieron ser inicializados y los que no'''
funcionando, no_funcionando = pygame.init()

print("El número de submódulos de pygame funcionando es: " + str(
    funcionando))
print("El número de submódulos de pygame que no están funcionando es: " +
     str(no_funcionando))

mensaje("palanca de mandos", pygame.joystick.get_init())
mensaje("cdrom", pygame.cdrom.get_init())

#Apagar el módulo de fuentes y de video:
pygame.font.quit()
mensaje("fuentes", pygame.font.get_init())
pygame.display.quit()
mensaje("video", pygame.display.get_init())

#Encender el módulo de fuentes:
pygame.font.init()
mensaje("fuentes", pygame.font.get_init())

print("Apagando pygame...")

'''Esta función apaga todos los módulos,
pero es llamada automáticamente cuando
el programa termina, por lo que sólo es necesario
en el caso que un programa deba seguir corriendo
sin pygame.
'''
pygame.quit()
```

1.6.3. Modos de video

Los modos de video disponibles son:

`pygame.FULLSCREEN` Crea una superficie de dibujo que ocupa toda la pantalla.

`pygame.DOUBLEBUF` Activa doble buffer en la memoria de video, recomendado con `HWSURFACE` y con `OPENGL`.

`pygame.HWSURFACE` Activa aceleración de hardware, disponible sólo con `FULLSCREEN` (en pantalla completa).

`pygame.OPENGL` Crea una superficie renderizable con opengl.

`pygame.RESIZABLE` Crea una ventana de dibujo redimensionable.

`pygame.NOFRAME` Crea una ventana de dibujo sin borde.

Listing 1.18: Modos de video en pygame

```
# coding: utf-8
'''c01/ejemplo-03.py
Modos de video en Pygame
'''
import pygame

def mostrarImagen():
    #Copiar la imagen al buffer temporal en 'pantalla':
    pantalla.blit(imagen, (0,0))

    #Volcar el buffer a la memoria de video:
    pygame.display.flip()

    #Espera un tiempo de 10 segundos
    pygame.time.delay(10000)

    #Apaga manualmente el sistema de video
    pygame.display.quit()

    print("Tamaño de la pantalla: {0}x{1}".format(pantalla.get_width,
        pantalla.get_height))

pygame.init()

tam = ancho, alto = 400, 400
titulo = "¡Modos de video! - "
imagen = pygame.image.load("logo_uca.bmp")

raw_input("Ventana normal de tamaño fijo :" + str(tam))
pantalla = pygame.display.set_mode(tam)
pygame.display.set_caption(titulo + "Ventana normal de tamaño fijo :" +
    str(tam))
mostrarImagen()

raw_input("Ventana normal de tamaño variable")
pantalla = pygame.display.set_mode(tam, pygame.RESIZABLE)
pygame.display.set_caption(titulo + "Ventana normal de tamaño variable")
mostrarImagen()

raw_input("Ventana normal de tamaño fijo maximizada (tamaño igual a
    pantalla completa)")
pantalla = pygame.display.set_mode()
mostrarImagen()

```

```
raw_input("Ventana sin borde de tamaño fijo: " + str(tam))
pantalla = pygame.display.set_mode(tam, pygame.NOFRAME)
mostrarImagen()

raw_input("Pantalla completa con resolución fija: " + str(tam))
pantalla = pygame.display.set_mode(tam, pygame.FULLSCREEN)
mostrarImagen()

raw_input("Pantalla completa con resolución máxima")
pantalla = pygame.display.set_mode((0,0), pygame.FULLSCREEN)
mostrarImagen()
```

1.6.4. Eventos de ratón

Listing 1.19: Eventos de ratón

```
# coding: utf-8
'''c01/ejemplo-04.py
Eventos del ratón en pygame
'''

import pygame

pygame.init()

tam = ancho, alto = 400, 400
titulo = "Eventos del ratón"
botones = ["izquierdo", "medio", "derecho", "rueda arriba", "rueda abajo"
    ]

pantalla = pygame.display.set_mode(tam)
pygame.display.set_caption(titulo)
corriendo = True

while corriendo:
    for evento in pygame.event.get():
        if evento.type == pygame.MOUSEBUTTONDOWN:
            print("Botón de ratón '{0}' presionado en {1}".format(\
                botones[evento.button-1], evento.pos))
        elif evento.type == pygame.MOUSEBUTTONUP:
            print("Botón de ratón '{0}' liberado en {1}".format(\
                botones[evento.button-1], evento.pos))
        elif evento.type == pygame.MOUSEMOTION:
            print("El ratón se movió {0} pixeles hasta {1}".format(\
                evento.rel, evento.pos))
            print("Los botones presionados son: {0}.".format(evento.
                buttons))
            print("\tBotón izq: {0}\n\tBotón cen: {1}\n\tBotón der: {2}".
                format(\
```

```
31                 (evento.buttons[0] and "sí" or "no"),
32                 (evento.buttons[1] and "sí" or "no"),
33                 (evento.buttons[2] and "sí" or "no") ))
34
35         elif evento.type == pygame.QUIT:
36             corriendo = False
```

1.6.5. Eventos de teclado

Listing 1.20: Eventos de teclado

```
1  # coding: utf-8
2  '''c01/ejemplo-05.py
3  Eventos del teclado en pygame
4  '''
5
6  import pygame
7
8  def devolverModificadores(m):
9      c = '<<'
10     c += m & pygame.KMOD_NUM and "NUMLOCK " or ""
11     c += m & pygame.KMOD_CAPS and "CAPSLOCK " or ""
12     c += m & pygame.KMOD_MODE and "MODE " or ""
13     c += m & pygame.KMOD_LCTRL and "LCTRL " or ""
14     c += m & pygame.KMOD_RCTRL and "RCTRL " or ""
15     c += m & pygame.KMOD_LSHIFT and "LSHIFT " or ""
16     c += m & pygame.KMOD_RSHIFT and "RSHIFT " or ""
17     c += m & pygame.KMOD_LALT and "LALT " or ""
18     c += m & pygame.KMOD_RALT and "RALT " or ""
19     c += m & pygame.KMOD_LMETA and "LMETA " or ""
20     c += m & pygame.KMOD_RMETA and "RMETA " or ""
21     c += '>> '
22     return c
23
24 def devolverTecla(tecla):
25     return "Código {0}, nombre: '{1}' ".format(tecla, pygame.key.name(
           tecla))
26
27 pygame.init()
28
29 tam = ancho, alto = 400, 400
30 titulo = "Eventos del teclado"
31
32 pantalla = pygame.display.set_mode(tam)
33 pygame.display.set_caption(titulo)
34 corriendo = True
35
36 while corriendo:
```

```
37      for evento in pygame.event.get():
38          if evento.type == pygame.KEYDOWN:
39              mensaje = "Presionada "
40              mensaje += devolverModificadores(evento.mod)
41              mensaje += devolverTecla(evento.key)
42              print(mensaje)
43          elif evento.type == pygame.KEYUP:
44              mensaje = "Liberada    "
45              mensaje += devolverModificadores(evento.mod)
46              mensaje += devolverTecla(evento.key)
47              print(mensaje)
48
49          elif evento.type == pygame.QUIT:
50              corriendo = False
```

1.6.6. Redimensionamiento de la ventana

Listing 1.21: Redimensionamiento

```
1   # coding: utf-8
2   '''c01/ejemplo-06.py
3   Redimensionamiento de la ventana en pygame
4   '''
5
6   import pygame
7
8   pygame.init()
9
10  tam = ancho, alto = 400, 400
11  titulo = "Redimensionamiento de la ventana"
12
13  pantalla = pygame.display.set_mode(tam, pygame.RESIZABLE)
14  pygame.display.set_caption(titulo)
15  corriendo = True
16
17  while corriendo:
18      for evento in pygame.event.get():
19          if evento.type == pygame.VIDEORESIZE:
20              pantalla = pygame.display.set_mode(evento.size, pygame.
                    RESIZABLE)
21              print("La ventana se ha redimensionado de {0} pixeles a {1}"
                    .\
22                  format(tam, evento.size))
23              tam = evento.size
24              #Aquí habría que redibujar lo que se estaba mostrando
25              #ya que 'pantalla' aparece borrada.
26
27          elif evento.type == pygame.QUIT:
```

```
28              corriendo = False
```

1.7. Dibujo de primitivas gráficas con pygame

En pygame, existe un módulo asociado al módulo pygame.display, que contiene las funciones para dibujar primitivas gráficas. Este módulo es pygame.draw.

Su uso se ilustra en el siguiente programa:

Listing 1.22: Ejemplo de uso de pygame.draw

```
1  # coding: utf-8
2  '''c01/ejemplo-08/primitivas.py
3  Primitivas␣gráficas
4  '''
5
6  import pygame, random
7
8  ancho, alto = 400, 400
9  titulo = "Primitivas␣gráficas"
10 rojo, verde, blanco, negro = (255,0,0), (0,255,0), (255,255,255), (0,0,0)
11
12 def dibujar():
13     #Borrar la pantalla
14     pantalla.fill(negro)
15
16     #Dibujar un rectángulo rojo y otro verde
17     pygame.draw.rect(pantalla, rojo, pygame.Rect(0,0,ancho/2, alto/2) )
18     pygame.draw.rect(pantalla, verde, pygame.Rect(ancho/2, alto/2, ancho
           /2, alto/2) )
19
20     #Lineas (notar la diferencia visual entre ambas):
21     pygame.draw.line(pantalla, blanco, (0,0), (ancho, alto) )
22     pygame.draw.aaline(pantalla, blanco, (0,alto), (ancho, 0) );
23
24     #Línea horizontal amarilla
25     pygame.draw.line(pantalla, (255, 255, 0), (0, alto/2), (ancho, alto
           /2) )
26
27     #Inicializar la fuente TrueType por defecto
28     fuente = pygame.font.Font(None, alto/20)
29     '''Las␣fuentes␣de␣sistema␣pueden␣encontrarse␣típicamente␣en
30 ␣␣␣␣/usr/share/fonts/...
31 ␣␣␣␣y␣dependen␣de␣cada␣sistema/distribución.'''
32
33     #Crear el texto
34     saludo      = fuente.render("HOLA", True, (255,0,255) )
```

```
35      bienvenida = fuente.render("Bienvenidos a pygame", False,
            (255,255,255), (128,128,128) )
36
37      #Copiar el texto
38      pantalla.blit(saludo, (ancho/2,alto/2) )
39      pantalla.blit(bienvenida, (3*ancho/4,alto/4) )
40
41      #Redibujar el buffer
42      pygame.display.flip()
43
44  pygame.init()
45
46  pantalla = pygame.display.set_mode((ancho, alto), pygame.RESIZABLE)
47  pygame.display.set_caption(titulo)
48  corriendo = True
49
50  #Hacer primer dibujo
51  dibujar()
52
53  while corriendo:
54      for evento in pygame.event.get():
55          if evento.type == pygame.VIDEORESIZE:
56              pantalla = pygame.display.set_mode(evento.size, pygame.
                    RESIZABLE)
57              print("La ventana se ha redimensionado de {0} pixeles a {1}"
                    .\
58                  format((ancho, alto), evento.size))
59              ancho, alto = evento.size
60              dibujar()
61
62          elif evento.type == pygame.QUIT:
63              corriendo = False
```

1.8. Ejercicios

1. Construya una aplicación gráfica que muestre en pantalla un display de 7 segmentos según las teclas numéricas que presione el usuario. Por ejemplo, si el usuario presiona el número 2 (ya sea del teclado numérico o del teclado normal) debería ver algo parecido a lo que aparece en la figura 1.1.

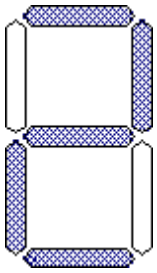


Figura 1.1: Display de 7 segmentos

2 Introducción a la Graficación por Computadora

2.1. Marco conceptual para la graficación interactiva

En general, para desarrollar aplicaciones que incluyan graficación interactiva, se deben atender los siguientes puntos:

2.1.1. El Modelado

¿Cómo se modelan las cosas en la computadora?

Tiene que describirse, de algún modo, cómo es aquello que queremos representar en la computadora y hasta qué nivel de detalle lo necesitamos representar. Esta descripción, es el *modelo*.

Obviamente, la forma de modelar las cosas, está supeditada al hecho que debe facilitar el proceso de dibujado de tal modelo y de su respectiva manipulación por parte del usuario (si es que esto último es parte del propósito de la aplicación).

Por ejemplo, piénsese en cómo modelar una esfera... **(a)** Una forma es por su ecuación esférica $r = R$ (asumiendo que está centrada en el origen), **(b)** por su ecuación rectangular: $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = R$ (en este caso puede estar descentrada), **(c)** por las coordenadas de la mayor cantidad posible de puntos que la conforman (para formar una nube densa de puntos), **(d)** etc.

En el primer caso, el modelo es sumamente simple (nada más es el radio de la esfera), pero el proceso de dibujo se vuelve bastante complejo debido a que hay que recalcular los puntos o líneas a dibujar en cada ocasión que necesitemos generar una imagen de la esfera. Lo cual no es precisamente lo más eficiente si se necesita dibujar la esfera con mucha frecuencia.

En el segundo caso, la situación es igual, excepto que el modelo es más flexible, porque permite describir esferas descentradas, pero nada más.

En el tercer caso, el proceso de dibujado será más rápido debido a que los puntos (que se calculan sólo la primera vez) ya estarán calculados, y bastará con volverlos a dibujar

cuando se necesite. Pero la alteración de la posición de la esfera, por ejemplo, será más lento, debido a que habrá que cambiar la posición de cada punto. Considérese en contraste el cambio de posición con el segundo caso en el que basta con cambiar tres variables.

Para que los objetos del mundo real puedan ser representados en la memoria (primaria y/o secundaria) de la computadora, necesitamos primero pensar en algún modelo matemático que describa esos objetos. Y además, en el caso de necesitar representarlos en la memoria secundaria, transformarlos a un modelo estrictamente lineal.

2.1.2. La Presentación

¿Cómo se dibuja el modelo?

Dado el *modelo*, hay que describir las técnicas necesarias (o al menos tenerlas muy claras) para poder *proyectar* o representar el modelo en la pantalla de la computadora (o en algún otro tipo de dispositivo de salida gráfica). Usualmente esa proyección es hacia dos dimensiones (y eso será así, al menos, mientras no tengamos monitores tridimensionales).

Se espera que una buena presentación describa el modelo lo suficientemente bien como para que el usuario lo comprenda y pueda manipularlo (o al menos imaginárselo) sin necesidad de entender los mecanismos internos de representación de dicho modelo.

Hay que tener en cuenta que la rapidez de los algoritmos empleados para este proceso debe ser, en general, lo más alta posible para que la aplicación pueda presentar una interacción con rapidez humana. Es decir, que el proceso de dibujo del modelo no debe ser muy lento. Tampoco debe ser demasiado rápido, pero eso es un problema menor, la mayoría de las veces.

2.1.3. La Interacción

¿Cómo interactúa el usuario con el modelo?

El *modelo* no necesariamente tiene una estructura idéntica a lo que el usuario ve por medio de la presentación descrita en la subsección anterior. Por ello también hay que definir la forma en que el usuario interactuará con el modelo (rotar un objeto tridimensional, desplazarlo o cambiarle el tamaño).

Dependiendo de la naturaleza de la aplicación, el usuario final, no necesariamente debe comprender cómo es almacenado y manipulado el modelo internamente, sino que debería poder manipularlo únicamente a través de lo que ve en la pantalla.

Sobre este tema se hablará más en el capítulo 8 en la página 163.

2.2. Ciclo de interacción

En general existen dos formas de implementar el ciclo de interacción principal de una aplicación. Estas dos formas son "por ciclo de eventos" y "por ciclo de juego" (conocido en inglés como *gameloop*).

Ambos tienen un propósito diferente, ya que el ciclo de eventos permite escrutar y procesar todos los eventos relevantes para la aplicación, mientras que el ciclo de juego hace un muestreo del estado de los dispositivos relevantes para la aplicación en diferentes momentos.

Con un ciclo por eventos, se tienen que procesar todos los eventos, teniendo que decidir si analizarlos o ignorarlos, pudiendo provocar una sobrecarga en el algoritmo de procesamiento, llegando a tener que procesar muchos eventos que sucedieron *hace mucho tiempo*[1]. Con un ciclo de juego, no hay forma de garantizar que todas las acciones del usuario serán percibidas y procesadas por la aplicación, pero el procesamiento tiende menos a retrasarse con respecto a las acciones del usuario.

El ciclo de eventos itera a través de la secuencia de eventos generados, pudiendo estos ocurrir en diferentes lapsos de tiempo, mientras que el ciclo de juego itera tan rápido como la computadora es capaz, o itera aproximadamente cada cierto tiempo, definido por el programador.

De manera esquemática, un ciclo de eventos tiene la siguiente estructura:

Listing 2.1: Ciclo de ventos

```
hayQueEjecutar = Verdadero
.
.
.
MIENTRAS (hayQueEjecutar){
    MIENTRAS (hayEventos()){
        evento = siguienteEvento()
        procesarEvento(evento);
    }
}
```

La estructura de un ciclo de juego suele parecerse al siguiente esquema:

Listing 2.2: Ciclo de juego

```
hayQueEjecutar = Verdadero
.
.
.
```

[1] *"Hace mucho tiempo"* para una aplicación gráfica podría significar un par de segundos.

```
MIENTRAS (hayQueEjecutar){
    t = ahora();

    estado = leerEstadoDeDispositivosRelevantes();
    actualizarObjetosYPosiciones(estado);
    redibujarObjetosVisibles();

    dt = ahora() - t;
    SI (dt < TiempoPorCiclo){
        hacerPausa(TiempoPorCiclo - dt);
    }
}
```

2.2.1. Ciclo de eventos con SDL

Como ya se ilustró en los ejemplos de la sección 1.3, para procesar los eventos de los dispositivos soportados por SDL (ratón, teclado, joystick y el manejador de ventanas), se extraen los eventos de una cola de eventos de la aplicación. Pueden utilizarse las funciones:

- `int SDL_PollEvent(SDL_Event *evento);` que verifica si hay eventos pendientes de procesar en la cola de eventos y retorna 1 si hay alguno. Retorna 0 si no hay ninguno en el momento de ser llamada. Siempre retorna inmediatamente.
- `int SDL_WaitEvent(SDL_Event *evento);` que espera indefinidamente hasta que haya otro evento en la cola de eventos y retornando 1 cuando llega. Retorna 0 si hubo algún error mientras se esperaba el evento. Si ya había eventos en la cola al momento de la llamada, retorna inmediatamente.

Ambas colocan en la dirección `evento` el siguiente evento extraído de la cola de eventos para esta aplicación.

En el caso de utilizar la primera, el ciclo principal del programa, el ciclo de eventos debería tener una forma parecida a esta:

Listing 2.3: Ciclo de ventos con `SDL_PollEvent`

```
{
    SDL_Event evento;
    int corriendo = 1;
    .
    .
    .
    while(corriendo){
        while(SDL_PollEvent(&evento)) {
            switch(evento.type){
                case SDL_MOUSEBUTTONDOWN:
```

```
                    .
                    .
                    .
                        break;
                case ...:
                    .
                    .
                    .
                case SDL_QUIT:
                    corriendo = 0;
                    break;
            }
        }
    }
}
```

En el caso de utilizar la segunda, el ciclo de eventos debería parecerse a:

Listing 2.4: Ciclo de ventos con `SDL_WaitEvent`

```
{
    SDL_Event evento;
    .
    .
    .
    while(SDL_WaitEvent(&evento)) {
        switch(evento.type){
            case SDL_MOUSEBUTTONDOWN:
                .
                .
                .
                break;
            case ...:
                .
                .
                .
            case SDL_QUIT:
                break;
        }
    }
}
```

2.2.2. Ciclo de juego con SDL

Para implementar un ciclo de juego con SDL, se dispone de las siguientes funciones:

- `Uint32 SDL_GetTicks(void);` Devuelve el número de milisegundos desde la inicialización de SDL. Esta función es útil para el control del tiempo que dura cada

iteración del ciclo de juego.

- `void SDL_Delay(Uint32 milisegundos);` Espera durante una cantidad especificada de milisegundos antes de continuar. Esta función esperará *al menos* el tiempo indicado, pero es posible que lleve más tiempo debido al proceso de asignación y cambio de procesos (multitarea) del Sistema Operativo en ese momento particular.
- `Uint8 *SDL_GetKeyState(int *numteclas);` Devuelve el estado instantaneo del teclado. El estado es representado por un arreglo cuya dimensión se coloca en la dirección `numteclas` si no es `NULL`. El arreglo se puede indexar por las constantes de la forma `SDLK_*`. Cada casilla puede contener un 1 que indica que la tecla está presionada o un 0 que indica que no está presionada. Para poder usar esta función, es necesario llamar primero a la función `void SDL_PumpEvents(void);` para actualizar el estado del arreglo. Hay que tener en cuenta que las funciones `SDL_PollEvent` y `SDL_WaitEvent` llaman internamente a `SDL_PumpEvents`, por lo que si se utilizan las primeras, no es necesario llamarla manualmente.
- `SDLMod SDL_GetModState(void);` Devuelve el estado actual de las teclas modificadoras (CTRL, ALT, CAPS-LOCK, etc.). El estado (de tipo `SDLMod`) es una variable entera que contiene las banderas de la forma `KMOD_*` combinadas con disyunción a nivel de bits.
- `Uint8 SDL_GetMouseState(int *x, int *y);` Devuelve el estado de los botones del ratón en una variable entera. El estado contiene las banderas `SDL_BUTTON_(L/M/R)MASK` combinadas con disyunción a nivel de bits. Si las direcciones `x` y `y` no son `NULL`, se almacena allí la coordenada del ratón en el momento de la llamada. Al igual que `SDL_GetKeyState`, esta función requiere una llamada previa a `SDL_PumpEvents`, que puede ser ejecutada a través de `SDL_PollEvent` o `SDL_WaitEvent`.
- `Uint8 SDL_GetRelativeMouseState(int *dx, int *dy);` Devuelve el estado de los botones del ratón igual que la función anterior, pero almacena en `dx` y `dy` (si no son `NULL`) los cambios de posición del cursor desde la última llamada a esta misma función o desde la inicialización de SDL.

La plantilla para un ciclo de eventos con SDL podría ser:

Listing 2.5: Ciclo de juego con SDL

```
1 #define MARCOS_POR_SEGUNDO 20
2 {
3     SDL_Event evento;
4     Uint8 *teclas;
5     int corriendo = 1;
6     Uint32 t, dt;
7     Uint32 TiempoCiclo = (int) (1000.0 / MARCOS_POR_SEGUNDO);
8     .
9     .
```

```
    .
    while(corriendo){
        t = SDL_GetTicks();

        SDL_PollEvent(&evento);
        if(evento.type == SDL_QUIT){
            corriendo = 0;
            break;
        }

        actualizarPosiciones(); //revisar colisiones, mover cosas, etc.

        teclas = SDL_GetKeyState(NULL);
        if (teclas[SDLK_ESCAPE]){
            corriendo=0;
            break;
        }

        if (teclas[SDLK_LEFT]){...}
        .
        .
        .

        redibujarObjetosVisibles();

        dt = SDL_GetTicks() - t;
        if (dt<TiempoCiclo)
            SDL_Delay(TiempoCiclo-dt);
    }
}
```

A continuación se incluye un pequeño programa de ejemplo, sencillo, sobre ciclo de juego con SDL:

Listing 2.6: Juego de tenis para dos jugadores con SDL

```
// c02/tenis/tenis.c

/**
 * Programa de ejemplo para ciclos de juego
 * */
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>

#define ANCHO 640
#define ALTO  480

#define INCREMENTO_DESPLAZAMIENTO 10
#define INCREMENTO_DESPLAZAMIENTO_PELOTA 4
```

```
15
16 #define MARCOS_POR_SEGUNDO 25
17 #define FONDO "fondo-trabajar.bmp"
18 #define PELOTA "pelotita.bmp"
19 #define TITULO  "¡Tenis! - Partida %d"
20
21 Uint32 color_fondo, color1, color2;
22 SDL_Surface *imgPelota, *imgFondo;
23 SDL_Rect jugador1, jugador2, pelota;
24 int velx, vely;
25
26 void reiniciarPosiciones(void){
27     jugador1.x = jugador2.x = ANCHO/2 - jugador1.w/2;
28     jugador1.y = 25;
29     jugador2.y = ALTO - 25 - jugador2.h;
30
31     pelota.x = (ANCHO/2) - (pelota.w/2);
32     pelota.y = (ALTO/2) - (pelota.h/2);
33     velx = ((rand() % 2) ? -1: 1) * INCREMENTO_DESPLAZAMIENTO_PELOTA;
34     vely = ((rand() % 2) ? -1: 1) * INCREMENTO_DESPLAZAMIENTO_PELOTA;
35 }
36
37 int puntoEnRectangulo(int x, int y, SDL_Rect *r){
38     return (x > r->x && x < r->x+r->w) &&
39            (y > r->y && y < r->y+r->h);
40 }
41
42 int main(int argc, char *argv[]){
43     SDL_Surface *pantalla = NULL;
44     SDL_Event evento;
45     Uint8 *teclas;
46
47     Uint32 t, dt;
48     Uint32 TiempoCiclo = (int) (1000.0 / MARCOS_POR_SEGUNDO);
49
50     int corriendo = 1;
51     int marcador1 = 0, marcador2 = 0;
52     int partida = 1;
53     char titulo[50];
54
55     /*Inicialización*/
56     {
57         srand(time(NULL));
58
59         if(SDL_Init(SDL_INIT_VIDEO) < 0 ) {
60             fprintf(stderr, "No se puede iniciar SDL: %s\n", SDL_GetError
                    ());
61             exit(1);
62         }
63         atexit(SDL_Quit);
```

```
64
65          //Cargar el fondo:
66          imgFondo = SDL_LoadBMP(FONDO);
67          if (imgFondo == NULL){
68              printf("Error al cargar el fondo: '%s'\n", SDL_GetError());
69              exit(1);
70          }
71          //Cargar la Pelota:
72          imgPelota = SDL_LoadBMP(PELOTA);
73          if (imgPelota == NULL){
74              printf("Error al cargar la pelota: '%s'\n", SDL_GetError());
75              exit(1);
76          }
77          SDL_SetColorKey(imgPelota, SDL_SRCCOLORKEY, SDL_MapRGB(imgPelota
                ->format,255,255,255));
78
79          jugador1.w = jugador2.w = 40;
80          jugador1.h = jugador2.h = 10;
81          pelota.w = imgPelota->w; pelota.h = imgPelota->h;
82
83          reiniciarPosiciones();
84
85          SDL_WM_SetIcon(imgPelota, NULL);
86          pantalla = SDL_SetVideoMode(ANCHO, ALTO, 0, SDL_SWSURFACE);
87          if(pantalla == NULL) {
88              fprintf(stderr, "No se puede establecer el modo de video %dx%
                    d: %s\n",
89                  ANCHO, ALTO, SDL_GetError());
90              exit(1);
91          }
92          sprintf(titulo, TITULO, partida);
93          SDL_WM_SetCaption(titulo, NULL);
94
95          color_fondo = SDL_MapRGB (pantalla->format,255,255,255);
96          color1 = SDL_MapRGB (pantalla->format,255,0,0);
97          color2 = SDL_MapRGB (pantalla->format,0,0,255);
98      }
99
100     while(corriendo) {
101         t = SDL_GetTicks();
102
103         /*Actualización*/
104         {
105             SDL_PollEvent(&evento);
106             if(evento.type == SDL_QUIT){
107                 corriendo = 0;
108                 break;
109             }
110
111             //Movimiento de la pelota
```

```
112             pelota.x += velx;
113             pelota.y += vely;
114
115             //Rebotes laterales
116             if (pelota.x < 0 || pelota.x + pelota.w > ANCHO)
117                 velx *= -1;
118
119             //Detección de choque vertical
120             if (pelota.y < 0 || pelota.y + pelota.h > ALTO){
121                 if (pelota.y < 0){
122                     printf("¡Jugador␣2␣gana!\n");
123                     marcador2++;
124                 }else{
125                     printf("¡Jugador␣1␣gana!\n");
126                     marcador1++;
127                 }
128                 sprintf(titulo, TITULO, ++partida);
129                 SDL_WM_SetCaption(titulo, NULL);
130                 reiniciarPosiciones();
131                 continue;
132             }
133
134             //Rebotes verticales
135             if (puntoEnRectangulo(pelota.x + pelota.w/2, pelota.y,
                            &jugador1) ||
136                 puntoEnRectangulo(pelota.x + pelota.w/2, pelota.y +
                        pelota.h, &jugador2)){
137                 vely *= -1;
138                 continue;
139             }
140
141             //Movimiento de los jugadores
142             teclas = SDL_GetKeyState(NULL);
143
144             if (teclas[SDLK_ESCAPE]){
145                 corriendo=0;
146                 break;
147             }
148
149             if (teclas[SDLK_LEFT]){
150                 jugador2.x -= INCREMENTO_DESPLAZAMIENTO;
151                 if (jugador2.x < 0) jugador2.x = 0;
152             }
153
154             if (teclas[SDLK_RIGHT]){
155                 jugador2.x += INCREMENTO_DESPLAZAMIENTO;
156                 if (jugador2.x + jugador2.w > pantalla->w)
157                     jugador2.x = pantalla->w-jugador2.w;
158             }
159
```

```
160             if (teclas[SDLK_a]){
161                 jugador1.x -= INCREMENTO_DESPLAZAMIENTO;
162                 if (jugador1.x < 0) jugador1.x = 0;
163             }
164
165             if (teclas[SDLK_d]){
166                 jugador1.x += INCREMENTO_DESPLAZAMIENTO;
167                 if (jugador1.x + jugador1.w > pantalla->w)
168                     jugador1.x = pantalla->w-jugador1.w;
169             }
170         }
171
172         /*Redibujar*/
173         {
174
175             //Borra la pantalla
176             SDL_BlitSurface(imgFondo, NULL, pantalla, NULL);
177             //SDL_FillRect(pantalla, NULL, color_fondo);
178
179             //Dibujo de los jugadores
180             SDL_FillRect(pantalla, &jugador1, color1);
181             SDL_FillRect(pantalla, &jugador2, color2);
182
183             //Dibujo de la bola
184             SDL_BlitSurface(imgPelota, NULL, pantalla, &pelota);
185
186             //Vuelca el buffer en la memoria de video
187             SDL_Flip (pantalla);
188         }
189
190         /* Sincronización de tiempo */
191         dt = SDL_GetTicks() - t;
192         if (dt<TiempoCiclo) SDL_Delay(TiempoCiclo-dt);
193     }
194
195     printf("\n<<<<<<<<<<<Marcador␣final:>>>>>>>>>>>\nJugador␣1:␣%d\
        nJugador␣2:␣%d\n\n", marcador1, marcador2);
196
197     return 0;
198 }
```

Se juega con las teclas “a” y “d” para el jugador 1 y con los cursores izquierda y derecha para el jugador 2. Cuando un jugador gana, se inicia una nueva partida.

2.2.3. Ciclo de eventos con pygame

Al igual que SDL, pygame dispone de las funciones para extraer eventos de la cola de eventos:

- `pygame.event.poll()` que verifica si hay eventos pendientes de procesar en la cola de eventos y retorna el primero de ella. Retorna `pygame.NOEVENT` si no hay ninguno en el momento de ser llamada. Siempre retorna inmediatamente.
- `pygame.event.wait()` que espera indefinidamente hasta que haya otro evento en la cola de eventos y devuelve el primero, cuando llega. Si ya había eventos en la cola al momento de la llamada, retorna inmediatamente.
- `pygame.event.get()` que extrae todos los eventos existentes en la cola de eventos en ese momento específico.

En el caso de utilizar la primera, el ciclo principal del programa, el ciclo de eventos debería tener una forma parecida a esta[2]:

Listing 2.7: Ciclo de ventos con `pygame.event.poll()`

```
corriendo = True
.
.
.
while corriendo:
    evento = pygame.event.poll()
    if evento == pygame.NOEVENT:
        continue
    if evento.type == pygame.MOUSEBUTTONDOWN:
        .
        .
        .
    elif evento.type == ...:
        .
        .
        .
    elif evento.type == pygame.QUIT:
        corriendo = False
```

En el caso de utilizar la segunda, el ciclo de eventos debería parecerse a[3]:

Listing 2.8: Ciclo de ventos con `pygame.event.wait()`

```
corriendo = True
.
.
.
while corriendo:
    evento = pygame.event.wait()
    if evento.type == pygame.MOUSEBUTTONDOWN:
        .
```

[2] puede consultarse el archivo `c02/ejemplo-raton-poll.py`
[3] puede consultarse el archivo `c02/ejemplo-raton-wait.py`

```
        .
        .
    elif evento.type == ...:
        .
        .
        .
    elif evento.type == pygame.QUIT:
        corriendo = False
```

En todo caso, la forma usada (porque tiende más al estilo de programación de Python) es con la función `pygame.event.get()`. Con esta función, el ciclo de eventos debería tener una forma similar a:

Listing 2.9: Ciclo de ventos con `pygame.event.get()`

```
corriendo = True
.
.
.
while corriendo:
    for evento in pygame.event.get():
        if evento.type == pygame.MOUSEBUTTONDOWN:
            .
            .
            .
        elif evento.type == ...:
            .
            .
            .
        elif evento.type == pygame.QUIT:
            corriendo = False
            break
```

2.2.4. Ciclo de juego con pygame

Para implementar un ciclo de juego con pygame, se dispone de las siguientes funciones:

- `pygame.time.get_ticks()` Devuelve el número de milisegundos desde la inicialización de pygame.
- `pygame.time.delay(milisegundos)` Realiza una espera activa durante un número de milisegundos especificados. Es bastante preciso ya que usa el procesador durante ese tiempo en lugar de pasar a inactividad.
- `pygame.time.wait(milisegundos)` Realiza una espera pasiva durante un número de milisegundos especificados. Es ligeramente menos preciso que `pygame.time.delay` ya que depende del sistema operativo para ser despertado.

- `pygame.key.get_pressed()` Devuelve una secuencia de booleanos representando el estado (presionado o no) de cada tecla del teclado. Se usan las constantes de tecla de la forma `pygame.K_*` para indexar la secuencia. Esta función requiere una llamada previa a `pygame.event.pump`, que se realiza automáticamente cuando se usan otras funciones del paquete `pygame.event`, como `pygame.event.get`, `pygame.event.wait` o `pygame.event.poll`.
- `pygame.key.get_mods()` Devuelve un valor entero que representa el estado actual de las teclas modificadoras (CTRL, ALT, CAPS-LOCK, etc.). El estado es una variable entera que contiene las banderas de la forma `pygame.KMOD_*` combinadas con disyunción a nivel de bits.
- `pygame.mouse.get_pressed()` Devuelve una tupla de tres booleanos indicando el estado (presionado o no) de los tres botones del ratón. El primero siempre es el botón izquierdo, el segundo el botón central y el tercero es el botón derecho. Al igual que `pygame.key.get_pressed`, esta función requiere una llamada previa a `pygame.event.pump`, que se realiza automáticamente cuando se usan las otras funciones del paquete `pygame.event`.
- `pygame.mouse.get_pos()` Devuelve una tupla de la forma `(x,y)` con las coordenadas del cursor (dentro de la ventana).
- `pygame.mouse.get_rel()` Devuelve una tupla de la forma `(dx,dy)` con la cantidad de movimiento del cursor desde la última llamada a esta misma función.

La plantilla para un ciclo de eventos con pygame podría ser:

Listing 2.10: Ciclo de juego con pygame

```
MARCOS_POR_SEGUNDO = 20
t=0; dt=0
TiempoCiclo = int(1000.0 / MARCOS_POR_SEGUNDO);
corriendo = True
.
.
.
while corriendo:
    t = pygame.time.get_ticks()

    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            pygame.display.quit() #Cierra la ventana de inmediato
            corriendo = False
            break
    if not corriendo:
        break

    actualizarPosiciones() #revisar colisiones, mover cosas, etc.

```

```
21      teclas = pygame.key.get_pressed()
22      if teclas[pygame.K_ESCAPE]:
23          corriendo=False
24          break
25
26      if teclas[pygame.K_LEFT]: ...
27          .
28          .
29          .
30
31      redibujarObjetosVisibles()
32
33      dt = pygame.time.get_ticks() - t
34      if dt<TiempoCiclo:
35          pygame.time.delay(TiempoCiclo-dt)
```

A continuación se incluye un pequeño programa de ejemplo, sencillo, sobre ciclo de juego con pygame:

Listing 2.11: Juego de tenis para dos jugadores con pygame

```
1  #!/usr/bin/env python
2  # coding: utf-8
3  """c02/tenis/tenis.py - Programa de ejemplo para ciclos de juego
4  """
5  import pygame, random, sys
6
7  ANCHO, ALTO = 640, 480
8
9  INCREMENTO_DESPLAZAMIENTO = 10
10 INCREMENTO_DESPLAZAMIENTO_PELOTA = 4
11
12 X, Y = 0, 1
13
14 MARCOS_POR_SEGUNDO = 25
15 FONDO = "fondo-trabajar.bmp"
16 PELOTA = "pelotita.bmp"
17 TITULO = "¡Tenis! - Partida {0}"
18
19 def reiniciarPosiciones():
20     jugador1.x = jugador2.x = ANCHO/2 - jugador1.w/2;
21     jugador1.y = 25;
22     jugador2.y = ALTO - 25 - jugador2.h;
23
24     pelota.x = (ANCHO/2) - (pelota.w/2);
25     pelota.y = (ALTO/2) - (pelota.h/2);
26     vel[X] = (random.randint(0,1) and -1 or 1) *
           INCREMENTO_DESPLAZAMIENTO_PELOTA
27     vel[Y] = (random.randint(0,1) and -1 or 1) *
           INCREMENTO_DESPLAZAMIENTO_PELOTA
```

```
28
29
30  # Inicialización:
31
32  t=0; dt=0
33  TiempoCiclo = int(1000.0 / MARCOS_POR_SEGUNDO);
34
35  marcador1 = 0; marcador2 = 0
36  partida = 1
37  corriendo = True
38
39  pygame.init()
40
41  # Cargar el fondo:
42  imgFondo = pygame.image.load(FONDO)
43  # Cargar la Pelota:
44  imgPelota = pygame.image.load(PELOTA)
45  imgPelota.set_colorkey((255,255,255))
46
47  jugador1 = pygame.Rect((0,0), (40,10))
48  jugador2 = pygame.Rect((0,0), (40,10))
49  pelota = imgPelota.get_rect()
50  vel = [0,0]
51
52  reiniciarPosiciones()
53
54  pygame.display.set_icon(imgPelota)
55  pantalla = pygame.display.set_mode((ANCHO,ALTO))
56  pygame.display.set_caption(TITULO.format(partida))
57
58  color_fondo = (255,255,255)
59  color1 = (255,0,0)
60  color2 = (0,0,255)
61
62  while corriendo:
63      t = pygame.time.get_ticks()
64
65      # Actualización: ######################
66      for evento in pygame.event.get():
67          if evento.type == pygame.QUIT:
68              pygame.display.quit()
69              corriendo = False
70              break
71
72      if not corriendo:
73          break
74      # Movimiento de la pelota
75      pelota.x += vel[X];
76      pelota.y += vel[Y];
77
```

```
78      # Rebotes laterales
79      if pelota.left < 0 or pelota.right > ANCHO:
80          vel[X] *= -1;
81
82      # Detección de choque vertical
83      if pelota.top < 0 or pelota.bottom > ALTO:
84          if pelota.top < 0:
85              print("¡Jugador␣2␣gana!\n")
86              marcador2 += 1
87          else:
88              print("¡Jugador␣1␣gana!\n")
89              marcador1 += 1
90          partida += 1
91          pygame.display.set_caption(TITULO.format(partida))
92          reiniciarPosiciones()
93          continue
94
95      # Rebotes verticales
96      if jugador1.collidepoint(pelota.midtop) or \
97         jugador2.collidepoint(pelota.midbottom):
98          vel[Y] *= -1
99          continue
100
101     # Movimiento de los jugadores
102     teclas = pygame.key.get_pressed()
103
104     if teclas[pygame.K_ESCAPE]:
105         corriendo=False
106         break
107
108     if teclas[pygame.K_LEFT]:
109         jugador2.x -= INCREMENTO_DESPLAZAMIENTO
110         if jugador2.left < 0: jugador2.left = 0;
111
112     if teclas[pygame.K_RIGHT]:
113         jugador2.x += INCREMENTO_DESPLAZAMIENTO
114         if jugador2.right > pantalla.get_width():
115             jugador2.x = pantalla.get_width()-jugador2.w
116
117     if teclas[pygame.K_a]:
118         jugador1.x -= INCREMENTO_DESPLAZAMIENTO
119         if jugador1.x < 0: jugador1.x = 0
120
121     if teclas[pygame.K_d]:
122         jugador1.x += INCREMENTO_DESPLAZAMIENTO
123         if jugador1.x + jugador1.w > pantalla.get_width():
124             jugador1.x = pantalla.get_width()-jugador1.w
125
126
127     # Redibujar: ##############################
```

```
128
129     # Borra la pantalla
130     pantalla.blit(imgFondo, (0,0))
131     #pantalla.fill(color_fondo)
132
133     # Dibujo de los jugadores
134     pantalla.fill(color1, jugador1)
135     pantalla.fill(color2, jugador2)
136
137     # Dibujo de la bola
138     pantalla.blit(imgPelota, pelota)
139
140     # Vuelca el buffer en la memoria de video
141     pygame.display.flip()
142
143     # Sincronización de tiempo ###################
144     dt = pygame.time.get_ticks() - t
145     if dt<TiempoCiclo: pygame.time.delay(TiempoCiclo-dt)
146
147 print("\n<<<<<<<<<<<Marcador␣final:>>>>>>>>>>>\nJugador␣1:␣{0}\nJugador␣
    2:␣{1}\n\n".format(marcador1, marcador2))
```

Se juega con las teclas “a” y “d” para el jugador 1 y con los cursores izquierda y derecha para el jugador 2. Cuando un jugador gana, se inicia una nueva partida.

2.3. Tipos de representación de gráficos

Fundamentalmente existen dos formas de representar y almacenar objetos gráficos en la memoria de una computadora:

2.3.1. Gráficos de barrido o raster

Esta representación establece la manipulación directa de los pixeles que conforman las imágenes. Estos objetos gráficos son conocidos típicamente como *mapas de bits*.

Muchas aplicaciones utilizan este esquema, principalmente usadas para la manipulación de imágenes a nivel de pixel.

Algunas aplicaciones típicas que funcionan así son aquellas tipo Paint, aplicaciones de retoque de fotos como Picasa de Google, y otras más conocidas como PhotoShop, GIMP, etc.

Los formatos típicos son los .bmp, .gif, .jpg, .png, etc.

2.3.2. Gráficos vectoriales

Esta representación establece la manipulación de las características descriptivas de los objetos. Por ejemplo: Se almacenan los puntos de inicio y fin de un segmento de recta y su ancho, en lugar de los pixeles que la conformarían; Se almacena el punto de la esquina superior izquierda de un cuadrado y su ancho y alto, en lugar de almancenar los pixeles de sus cuatro lados; Se almacena el centro y radio de un círculo y el grueso de su circunferencia, en lugar de almacenar los pixeles de toda la circunferencia; etc.

Las aplicaciones que utilizan este esquema generalmente están orientados a la ingeniería y en general al diseño asistido por computadora (Computer-Aided Design, CAD). Algunas aplicaciones típicas que funcionan así son: AutoCAD de AutoDesk, Google SketchUp, OO.org Draw, Inkscape, QCad, Dia, etc.

Los formatos específicos varían mucho dependiendo de la aplicación, pero algunos más conocidos son los .dia, .svg, .dwg, .dxf, .odg, etc.

2.3.3. Representación híbrida

Existen, sin embargo, aplicaciones con funcionalidades híbridas. Por ejemplo, en aplicaciones típicamente de barrido como GIMP (o Photoshop), el texto puede almacenarse como imagen, pero también como una cadena de caracteres inmersos en un rectángulo descrito vectorialmente, dentro del área de trabajo.

Así mismo, en aplicaciones primariamente vectoriales como OO.org Draw, se pueden insertar archivos de imágen de barrido (png, gif, jpg) y cambiarles, por ejemplo, el contraste a dicha imágen.

2.4. Paletas de colores

Para representar el color de un objeto gráfico, ya sea un pixel o un objeto vectorizado, es necesario utilizar una cantidad finita de memoria para ello. Y en los tiempos iniciales de la computación gráfica, la memoria era un recurso sumamente costoso y demasiado valioso como para desperdiciarlo. Por ello surgió la necesidad de utilizar poca memoria para representar los colores a utilizar por una aplicación gráfica.

Por ejemplo, se puede utilizar una paleta de 8 bits (un byte), para permitirle a una aplicación, usar hasta $2^8 = 256$ colores. De estos, hay que especificar, *arbitrariamente*, que el primero, el 00_H sea negro, el 01_H sea blanco,. . ., el 09_H sea azul, el $0A_H$ sea verde, etc.

Esa asignación *arbitraria* puede realizarse con ayuda de la biblioteca gráfica a usar (generalmente traen paletas predefinidas) o directamente con código ensamblador, en función de las capacidades del dispositivo de graficación (monitor, impresor, etc.).

2.5. Paletas estándares actuales

Con el advenimiento de *dispositivos de graficación* cada vez más avanzados (que pueden desplegar más colores y tienen mayor resolución), con el abaratamiento exponencial de la *memoria primaria* para computadoras personales y con *tarjetas gráficas* cada vez más sofisticadas, el uso de paletas de colores ha ido poco a poco cayendo en desuso entre los programadores gráficos. En su lugar, suele especificarse (y almancenarse) el color directamente como lo entienden los dispositivos de graficación modernos a través de estándares internacionales.

Estos nuevos estándares tienen, de hecho, la forma de paletas de colores realmente grandes, que no se pueden cambiar y que permiten variaciones de color *aparentemente* continuas para el ojo humano[4].

Las más usadas, soportadas en casi todos los lenguajes de programación, son los de mezcla aditiva de luz (*RGB* y *RGBA*) y los de mezcla sustractiva de luz (*CMY(K)*).

2.5.1. RGB

[Red, Green, Blue]

Un color en este formato se suele espeficicar en la forma de un conjunto de tres bytes continuos (24 bits), donde los primeros ocho bits, de derecha a izquierda, representan la intensidad de luz azul. Los segundos ocho bits, de derecha a izquierda, la intensidad de luz verde, y los últimos ocho, de derecha a izquierda, la intensidad de luz roja. En la figura 2.1 podemos ver el típo diagrama de colores en el modelo RGB.

Este modelo representa un color mediante la mezcla por adición de los tres colores primarios de la luz, rojo, verde y azul. De tal manera que en una superficie negra, su suma a nula intensidad, resulta en negro; y su suma a máxima intensidad, resulta en blanco.

Eventualmente, y dependiendo de la arquitectura del procesador, el orden en que se indican estas intensidades puede ser al revés de explicado arriba[5]. Este es un detalle de poca o nula importancia cuando que utilizan APIs de alto nivel.

[4] es decir, la diferencia entre un "color" y otro, es menor que el umbral diferencial de la luz, en los humanos.

[5] Para mayor información, véanse los artículos: http://es.wikipedia.org/wiki/Big-endian y http://en.wikipedia.org/wiki/Endianness

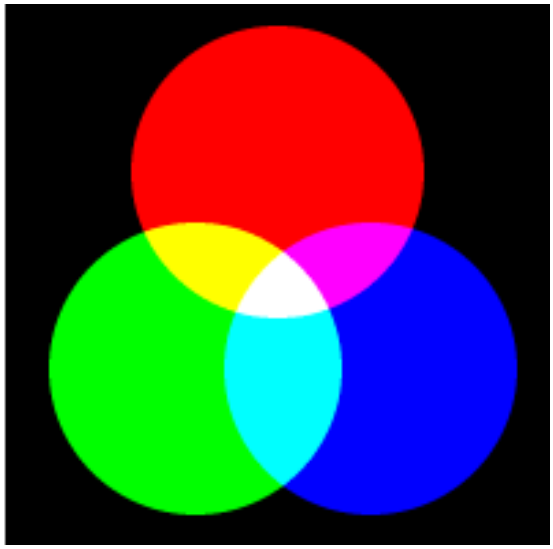


Figura 2.1: Modelo de color RGB

2.5.2. RGBA

[Red, Green, Blue, Alpha]

Este otro modo de especificación de color, es similar al anterior, excepto que usa 32 bits. Los ocho que se han agregado (normalmente al final), representan la medida de opacidad/transparencia (en el caso de SDL, 00_H es transparencia total y FF_H es opacidad total, pero en otras bibliotecas gráficas, es al revés). E igual que en el caso del RGB, su representación binaria varía dependiendo del hardware.

2.5.3. CMY(K)

[Cyan, Magenta, Yellow, Black (o Key)]

Este modelo representa un color mediante la mezcla sustractiva de los colores cyan, magente y amarillo. De tal manera que impresos sobre un papel blanco, su mezcla a mínima intensidad no absorbe nada y la luz se refleja completamente blanca, pero si se mezclan a máxima intensidad, absorben la luz completamente, por lo que no vemos luz reflejada (eso es el color negro).

Esta es la forma en que se especifican los colores para los dispositivos de impresión con algún tipo de tinta. Y la inclusión de tinta de color negro directamente, se debe a razones técnicas específicas[6].

[6] se les llama sistemas de impresión de cuatro tintas

Figura 2.2: Modelo de color CMY(K)

2.6. Espacios de Colores (gamuts)

En la teoría del color, *el gamut de un dispositivo o proceso usado para la creación de un color, es la porción del espacio de color de la luz visible que se puede representar con ese dispositivo o proceso.*

Es obvio que existen limitaciones físicas en todos los dispositivos y procesos, que les impiden mostrar la gama completa del espacio de color de la luz visible (que es bastante grande). Por eso se habla de una porción del espacio de color de la luz visible.

También se podría definir como *el lugar geométrico de los puntos del plano matiz-saturación que se pueden representar mediante el dispositivo o proceso en cuestión.* (ver figura 2.3).

El color visto en el monitor de una computadora es diferente al color del mismo objeto en una impresión en papel, pues los modelos CMYK y RGB tienen diferentes *gamuts*. Por ejemplo, el azul puro (rgb: 0,0,100 %) es imposible de reproducir en CMYK. El equivalente más cercano en CMYK es un tono azulvioláceo.

En general, en los materiales impresos, las combinaciones de luz RGB no pueden ser reproducidas directamente, por lo que las imágenes generadas en los ordenadores, cuando se usa un programa de edición, dibujo vectorial, o retoque fotográfico se debe convertir a su equivalente en el modelo CMYK que es el adecuado cuando se usa un dispositivo que usa tintas, como las impresoras o los ploters.

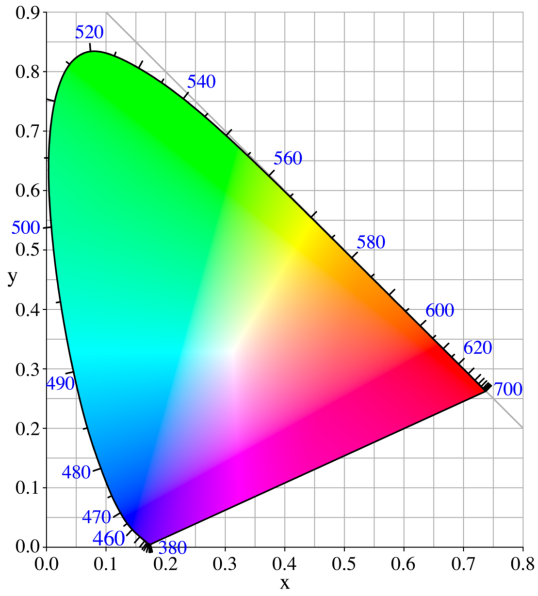


Figura 2.3: Plano Matiz-Saturación de la luz visible

En la figura 2.4 se presentan (sin pretender ofrecer una gran precisión) las diferencias de los gamuts de RGB y de CMYK.

El gamut del modelo RGB es el triángulo y la otra figura es el gamut del modelo CMYK.

2.7. Ejercicios

1. Describa las características de los gráficos de barrido y los gráficos vectoriales.
2. Suponga que una aplicación modela dos círculos, A y B. El círculo A es modelado como gráfico de barrido y el B como gráfico vectorial. Describa cómo podría representar dicha aplicación ambos círculos.
3. Explique con sus propias palabras, qué es el Gamut de un impresor laser.
4. Investigue cómo hacer para detectar combinaciones del tipo `clic izquierdo+shift`, `clic derecho+ctrl`, etc.

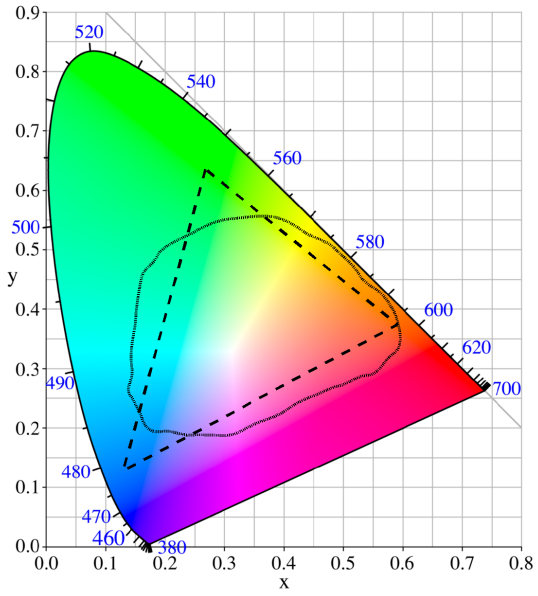


Figura 2.4: Comparación del Gamut de RGB y CMYK

3 Discretización de Primitivas Gráficas

Antes de iniciar este capítulo, vale la pena aclarar el concepto de Primitiva Gráfica: Se refiere a las figuras básicas de dibujo vectorial, a partir de las cuales se realizan todas las demás. Estas figuras básicas son los puntos, los segmentos de rectas, círculos, elipses y polígonos.

Por otro lado, antes de comenzar con este capítulo, se sugiere hacer un repaso de los temas de geometría analítica básica. Aquí sólo se mencionarán las formas de las ecuaciones usadas en el resto del libro, sin describirlas formalmente ya que se asume que el lector ha aprobado algún curso básico de geometría analítica vectorial.

3.1. Recordatorio básico

Recuérdese la ecuación *pendiente-intersecto* de las líneas rectas:

$$y = mx + B \tag{3.1}$$

Recuérdese la ecuación *punto-pendiente* de las líneas rectas:

$$y - y_0 = m(x - x_0) \tag{3.2}$$

La equación *dos puntos* de las líneas rectas no verticales:

$$y - y_0 = \left(\frac{y_1 - y_0}{x_1 - x_0}\right)(x - x_0) \tag{3.3}$$

También se dispone de la ecuación canónica o implícita de las líneas rectas:

$$ax + by + c = 0 \tag{3.4}$$

Recuérdese la ecuación paramétrica de las circunferencias centradas en un punto dado:

$$(x - x_0)^2 + (y - y_0)^2 = R^2 \tag{3.5}$$

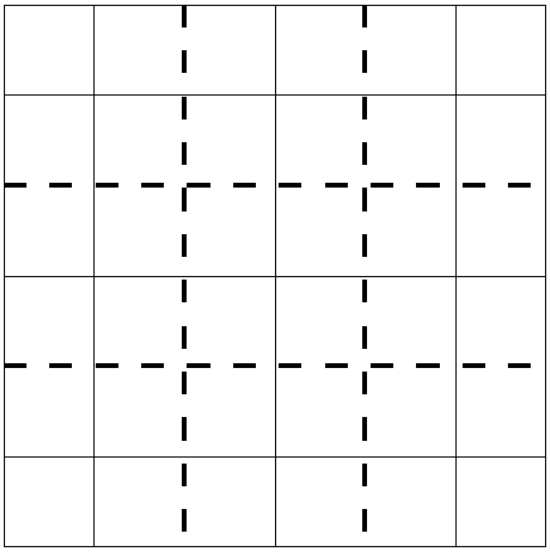


Figura 3.1: Representación abstracta de nueve pixeles

3.2. Simbología

Antes que nada, consideremos que los dispositivos principales sobre los que vamos a trabajar, son monitores cuyas pantallas están compuestas por pixeles. Los pixeles están dispuestos en forma de una matriz rectangular de puntos *casi* perfectamente cuadrados y cada uno puede brillar de un color independiente de los demás.

La figura 3.1 es la imágen que se usará para esquematizar en este capítulo cómo funcionan los algoritmos presentados

La cuadrícula de la figura 3.1 representa un grupo de nueve pixeles de una computadora. Las líneas gruesas de guiones representan las fronteras entre dichos pixeles y las intersecciones de las líneas continuas delgadas representan los centros geométricos de ellos.

Por simplicidad, vamos a suponer, en las deducciones, que la numeración de los pixeles se extiende desde la esquina inferior izquierda, el pixel $(0, 0)$ de las pantallas, hasta la esquina superior derecha. En realidad no es así, pero así es más parecido a los textos básicos de matemática, con los cuales el lector está familiarizado.

Se dice también que, respecto del pixel central, el pixel que está a su derecha, es su pixel «ESTE» o simplemente «E». El pixel de la esquina superior derecha, es el pixel «NOR-ESTE» o «NE» respecto del mismo pixel central. El pixel de la esquina inferior izquierda, es el pixel «SUR-OESTE» o «SO» respecto del pixel central, etc.

3.3. Algoritmo incremental básico

En esta sección, se procederá a describir un algoritmo para dibujar líneas rectas que es sencillo y eficiente desde el punto de vista matemático.

Analicemos el problema de dibujar una línea del punto (x_0,y_0), al (x_1,y_1).

Lo primero a realizar es la definición del modelo: Sea $y_i = mx_i + B$ la ecuación que define la linea, y los pixeles a encender (o a colorear) serán los pares ordenados: $(x_i, redondear(y_i))$. Evidentemente, $m = \frac{y_1-y_0}{x_1-x_0}$ y $B = y_0 - \left(\frac{y_1-y_0}{x_1-x_0}\right) x_0$.

Replanteando el problema en términos de un proceso iterativo, consideramos que $x_{i+1} - x_i = \Delta x$ para iterar sobre el eje horizontal, de izquierda a derecha. Entonces podemos plantear lo siguiente:

$$\begin{aligned} y_{i+1} &= mx_{i+1} + B \\ &= m(x_i + \Delta x) + B \\ &= mx_i + m\Delta x + B \\ &= m\Delta x + (mx_i + B) \\ y_{i+1} &= y_i + m\Delta x \end{aligned}$$

y si asumimos que $\Delta x = 1$ ya que avanzamos de columna en columna de pixeles, entonces:

$$\begin{aligned} y_{i+1} &= y_i + m \\ x_{i+i} &= x_i + 1 \end{aligned} \tag{3.6}$$

Esta idea es la base para el algoritmo que se conoce como Analizador Diferencial Digital (DDA, *Digital Differential Analyzer*). Que está restringido al caso en que $-1 \leq m \leq 1$. Para otros valores de m, se puede usar simetría para modificar el algoritmo.

Esta es su especificación en lenguaje c:

Listing 3.1: Algoritmo Analizador Diferencial Digital

```
1  void dda(int x0, int y0, int x1, int y1){
2      /*
3          Supone que -1 <= m <= 1, x0 < x1.
4          x varía de x0 a x1 en incrementos unitarios
5      */
6      int x;
7      float y, m;
8      m = (float)(y1 - y0) / (x1 - x0);
9      y = y0;
10     for(x = x0; x <= x1, x++){
11         marcarPixel(x, (int) y);
```

```
12              y += m;
13          }
14      }
```

A pesar de que esta solución funciona y se puede adaptar a cualquier pendiente, tiene un detalle indeseable: *Utiliza artimética de coma flotante*. Por ello, en las siguientes secciones se presenta una elegante solución que utiliza únicamente aritmética entera.

3.4. Algoritmo de línea de punto medio

A continuación, se procederá a describir uno de los mejores algoritmos de dibujo de líneas rectas: El *algoritmo de línea de punto medio* (también conocido como *algoritmo de línea de Bresenham*).

Antes de abordar la resolución general del dibujo de segmentos de recta, se abordará un caso particular más sencillo que permite describir la lógica del algoritmo.

> El problema consiste en dibujar un segmento de recta, desde el pixel inferior izquierdo (x_0, y_0), hasta el pixel superior derecho (x_1, y_1) usando únicamente aritmética entera.

Considere la línea que se representa en la figura 3.2, donde el pixel previamente seleccionado aparece sombreado y los dos pixeles candidatos a ser seleccionados en el siguiente paso, son el pixel a la derecha del marcado, el pixel "ESTE" E, y el pixel que está arriba del anterior, el pixel "NOR-ESTE" NE. El punto M es el punto medio entre los centros geométricos de E y de NE.

Acabamos de marcar el pixel en cuestión, el pixel P, en las coordenadas (x_0, y_0) y ahora tenemos que elegir entre el pixel E y el NE. Eso lo hacemos averiguando si la recta ideal pasa más cerca del centro geométrico de E o de NE.

Si el punto medio M, está por encima de la línea, el pixel E es el más cercano a la línea. Si el punto medio está debajo, el pixel NE es el más cercano. En cualquier caso, el error es siempre menor o igual a $\frac{1}{2}$ de pixel.

En el caso de la figura 3.2, el pixel escogido como siguiente es el NE, y en el caso de la figura 3.3 en la página siguiente es el E.

Lo que se necesita entonces, es una manera de averiguar de qué lado de la línea está M. Para ello se usará la siguiente ecuación de la línea recta, la ecuación implícita: $ax + by + c = 0$.

Con ella, podemos construir la siguiente función:

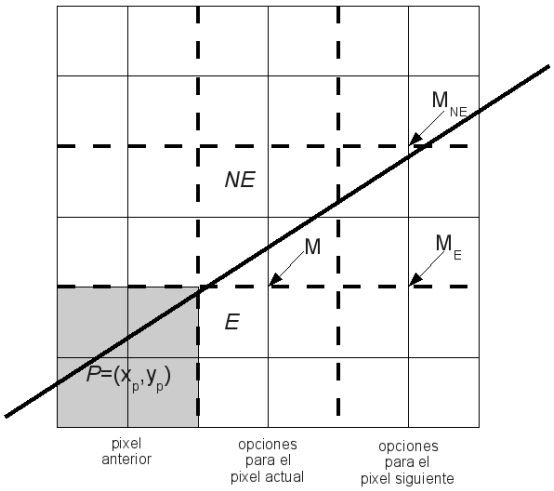


Figura 3.2: Justificación del algoritmo de línea de punto medio básico - 1

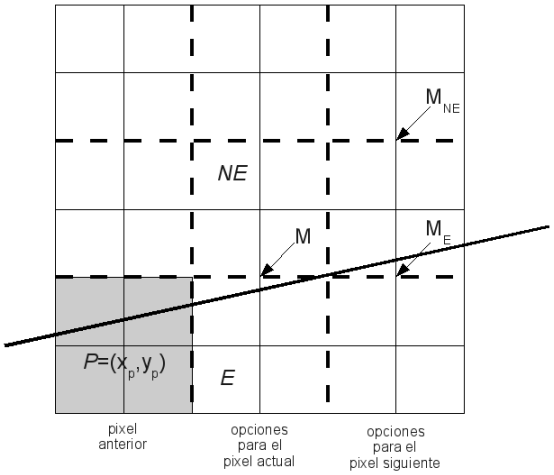


Figura 3.3: Justificación del algoritmo de línea de punto medio básico - 2

$$F(x,y) = ax + by + c \tag{3.7}$$

Esta tiene la siguiente característica: El valor de F es cero para los puntos que pertenecen a la línea, positivo si el punto evaluado está debajo de la línea, y es negativo si el punto evaluado está encima de la línea, siempre y cuando el coeficiente b, el coeficiente de y sea negativo.

$$\ldots\ldots\ldots\ldots F(x,y) < 0 \Longrightarrow \underline{(x,y)} \ldots\ldots\ldots\ldots F(x,y) > 0 \Longrightarrow \overline{(x,y)} \ldots\ldots\ldots\ldots$$

Entonces, tenemos que evaluar el signo de $F(M) = F\left(x_p + 1, y_p + \frac{1}{2}\right)$ para saber si M está arriba o abajo de la recta ideal.

Para ordenar un poco el panorama, definamos una nueva variable, nuestra variable de decisión:

$$d = F\left(x_p + 1, y_p + \frac{1}{2}\right) = a\left(x_p + 1\right) + b\left(y_p + \frac{1}{2}\right) + c \tag{3.8}$$

Con esta variable, podemos establecer el siguiente convenio: Si $d > 0$, se elige el pixel NE; si $d \leq 0$, se elige el pixel E (si $d = 0$, podríamos elegir cualquiera, pero por convención, eligimos E).

Una vez tomada la decisión y marcado el pixel, se pasa al siguiente punto y se hace lo mismo. $d_{nuevo} = F(M_E) = F\left((x_p + 1) + 1, y_p + \frac{1}{2}\right)$ si se eligió el pixel E, y $d_{nuevo} = F(M_{NE}) = F\left((x_p + 1) + 1, (y_p + 1) + \frac{1}{2}\right)$ si se eligió el pixel NE.

Ahora bien, desconocemos los valores de a, b y c. ¿Qué hacemos?. Hagamos una pausa y analicemos el siguiente fenómeno:

Sea $P(x) = mx + B$ y sea $x_{i+1} = x_i + 1$ (ya que consideramos que los sucesivos x_i son las coordenadas de pixeles). Entonces:

$$\begin{aligned} P(x_{i+1}) &= mx_{i+1} + B \\ &= m(x_i + 1) + B \\ &= (mx_i + B) + m \\ P(x_{i+1}) &= P(x_i) + m \\ P(x_{i+1}) - P(x_i) &= m \end{aligned}$$

Ahora veamoslo aplicado a nuestro problema de F (ver la ecuación 3.7):

- En el caso de elegir el pixel E:

$$\begin{aligned} F(x_{i+1}, y_i) &= a(x_i + 1) + by_i + c \\ &= ax_i + a + by_i + c \\ &= (ax_i + by_i + c) + a \\ F(x_{i+1}, y_i) &= F(x_i, y_i) + a \\ F(x_{i+1}, y_i) - F(x_i, y_i) &= a \\ d^E_{nuevo} - d_{viejo} &= a = \Delta_E \end{aligned}$$

- Veamos el caso en que se elige el pixel NE:

$$\begin{aligned} F(x_{i+1}, y_{i+1}) &= a(x_i + 1) + b(y_i + 1) + c \\ &= ax_i + a + by_i + b + c \\ &= (ax_i + by_i + c) + a + b \\ F(x_{i+1}, y_{i+1}) &= F(x_i, y_i) + a + b \\ F(x_{i+1}, y_{i+1}) - F(x_i, y_i) &= a + b \\ d^{NE}_{nuevo} - d_{viejo} &= a + b = \Delta_{NE} \end{aligned}$$

A estas diferencias recién introducidas, las llamaremos Δ_E y Δ_{NE} respectivamente.

Resta entonces el problema del valor inicial de d (equivale al valor d_{viejo} del primer paso):

$$\begin{aligned} d_{inicial} &= F(M) \\ &= F(x_0 + 1, y_0 + \frac{1}{2}) = a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c \\ &= ax_0 + a + by_0 + \frac{1}{2}b + c \\ &= ax_0 + by_0 + c + a + \frac{1}{2}b \\ &= (ax_0 + by_0 + c) + a + \frac{1}{2}b \\ &= F(x_{0,}y_0) + a + \frac{1}{2}b;\ F(x_{0,}y_0) = 0 \\ d_{inicial} &= a + \frac{1}{2}b \end{aligned}$$

Se deben conseguir los parámetros a y b (c no es necesario). Para ello, partimos de $y = \frac{\Delta y}{\Delta x}x + B$, así:

$$\begin{aligned} y &= \frac{\Delta y}{\Delta x}x + B \\ \Delta x \cdot y &= \Delta y \cdot x + \Delta x \cdot B \\ 0 &= \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot B \\ F(x,y) &= \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot B \end{aligned}$$

por lo que:

$$\begin{aligned} a &= \Delta y \\ b &= -\Delta x \\ c &= \Delta x \cdot B \end{aligned}$$

Note que el coeficiente b resulta negativo. Note además que el valor de B es desconocido (se puede calcular, porque conocemos al menos dos puntos de la recta), pero como no necesitamos c, no representa ningún problema para nuestro propósito.

Tenemos entonces:

$$\begin{aligned} \Delta_E = &\quad F(x_{i+1}, y_i) - F(x_i, y_i) = a &&= \Delta y \\ \Delta_{NE} = &\quad F(x_{i+1}, y_{i+1}) - F(x_i, y_i) = a + b &&= \Delta y - \Delta x \\ d_{inicial} = &\quad a + \tfrac{1}{2}b &&= \Delta y - \frac{1}{2}\Delta x \end{aligned}$$

Pero esto nos fuerza a usar coma flotante sólo por el valor inicial de d. Todos los demás valores sucesivos son enteros...

Entonces, replanteamos la ecuación inicial, de la siguiente manera:

$$\begin{aligned} y &= \frac{\Delta y}{\Delta x}x + B \\ 2y &= 2\frac{\Delta y}{\Delta x}x + 2B \\ 2\Delta x \cdot y &= 2\Delta y \cdot x + 2\Delta x \cdot B \\ F(x,y) &= 2\Delta y \cdot x - 2\Delta x \cdot y + 2\Delta x \cdot B \end{aligned} \tag{3.9}$$

por lo que ahora:

$$\begin{aligned} a &= 2\Delta y \\ b &= -2\Delta x \\ c &= 2\Delta x \cdot B \end{aligned}$$

y volvemos a calcular los valores que nos interesan[1]. Llegamos a:

$$\begin{aligned} \Delta_E = &\quad F(x_{i+1}, y_i) - F(x_i, y_i) = a &&= 2\Delta y \\ \Delta_{NE} = &\quad F(x_{i+1}, y_{i+1}) - F(x_i, y_i) = a + b &&= 2\Delta y - 2\Delta x \\ d_{inicial} = &\quad a + \tfrac{1}{2}b &&= 2\Delta y - \Delta x \end{aligned}$$

Con esto hemos resuelto[2] el problema del valor inicial flotante para d.

Ahora, el algoritmo usa exclusivamente artimética entera, sin el costoso tiempo para hacer operaciones en coma flotante.

El código en lenguaje C es:

Listing 3.2: Algoritmo de línea de punto medio en la mitad de un cuadrante

```
1  void linea_punto_medio_1(int x0, int y0, int x1, int y1){
2      int delta_x, delta_y, delta_E, delta_NE, d, x, y;
3      /*
4          Supone que x0 < x1, y0 < y1 y que delta_y <= delta_x.
5          x varía de x0 a x1 en incrementos unitarios
6      */
7      delta_x = x1 - x0;
8      delta_y = y1 - y0;
9      d = delta_y * 2 - delta_x;
10     delta_E = delta_y * 2;
11     delta_NE = (delta_y - delta_x) * 2;
12     x = x0;
13     y = y0;
14     marcarPixel(x, y); //marcar el primero
15     while(x < x1){
16         if(d <= 0){
17             d += delta_E;
18         }else{
19             d += delta_NE;
20             y++;
21         }
22         x++;
```

[1] Basta con repetir el proceso con la nueva ecuación F

[2] Note que el valor de b sigue siendo negativo

```
23          marcarPixel(x, y);
24      }
25  }
```

3.4.1. Simetría del algoritmo de línea de punto medio

El problema de generalizar el algoritmo de punto medio para pendientes arbitrarias tiene dos componentes: **(a)** permitir pendientes mayores que 1, y **(b)** poder hacer el recorrido *hacia atrás*, para poder aceptar los puntos en cualquier orden sin tener que intercambiar las variables de entrada.

El primer problema se resuelve, averiguando la relación de orden de Δx y Δy. En base a eso, se decide hacer el recorrido sobre x o sobre y. Así:

Listing 3.3: Algoritmo de línea de punto medio de un cuadrante (o dos mitades de cuadrante)

```
1  void linea_punto_medio_2(int x0, int y0, int x1, int y1){
2
3      int delta_x, delta_y, delta_E, delta_NE, d, x, y;
4      delta_x=x1-x0; delta_y=y1-y0;
5      x=x0, y=y0;
6
7      // Cuando 'horizontal' es verdadero, el ciclo iterará a través de x,
           si no, lo hará a través de y
8      int horizontal = delta_y < delta_x;
9
10     // Inicializaciones
11     if(horizontal){
12         d = (delta_y*2)-delta_x;
13         delta_E = delta_y*2;
14         delta_NE = (delta_y-delta_x)*2;
15     }else{
16         d = (delta_x*2)-delta_y;
17         delta_E = delta_x*2;
18         delta_NE = (delta_x-delta_y)*2;
19     }
20
21     // Dibujar el primer pixel
22     marcarPixel(x,y);
23     if(horizontal)
24         while(x < x1){ // Iterar a través de x
25             if(d<=0)
26                 d+=delta_E;
27             else{
28                 y++;
29                 d+=delta_NE;
30             }
```

```
31              x++;
32              marcarPixel(x,y);
33          }
34      else
35          while(y < y1){ // Iterar a través de y
36              if(d<=0)
37                  d+=delta_E;
38              else{
39                  x++;
40                  d+=delta_NE;
41              }
42              y++;
43              marcarPixel(x,y);
44          }
45  }
```

Ahora bien, el otro problema es un poco más complicado. Consideremos la figura 3.4 y repitamos el análisis a partir de la ecuación 3.9, pero haciendo las modificaciones pertinentes.

Llegamos a lo siguiente[3]:

$$\begin{aligned} d' = \quad & F\left(x_p - 1, y_p - \tfrac{1}{2}\right) && = a\left(x_p - 1\right) + b\left(y_p - \frac{1}{2}\right) + c \\ \Delta_O = \quad & -2\Delta y && = -\Delta_E \\ \Delta_{SO} = \quad & -2\Delta y + 2\Delta x && = -\Delta_{NE} \\ d'_{inicial} = \quad & -2\Delta y + \Delta x && = -d_{inicial} \end{aligned}$$

Lo que significa que el convenio en este caso es: Si $d' \geq 0$, se elige el pixel O; si $d' < 0$, se elige el pixel SO.

Aunque esto parezca bastante complicado, no hace falta darle muchas vueltas al asunto[4]. Basta con tomar conciencia del siguiente razonamiento:

Si el valor inicial de d y todos sus incrementos posibles son del mismo valor absoluto, pero de signo opuesto, y el criterio se basa en el signo de dicha variable, significa que se pueden sustituir los valores Δ_O, Δ_{SO} y $d'_{inicial}$ por Δ_E, Δ_{NE} y $d_{inicial}$ respectivamente sin afectar gravemente el algoritmo. Basta con invertir el criterio de incremento, de tal manera que quede así: Si $d' > 0$, se elige el pixel SO (sumar Δ_{NE}); si $d' \leq 0$, se elige el pixel O (sumar Δ_E). De tal manera que con una ligera modificación al código 3.2 se puedan considerar ambos casos, así:

[3] La demostración de esto se deja como ejercicio
[4] tal como lo hizo el autor de este libro

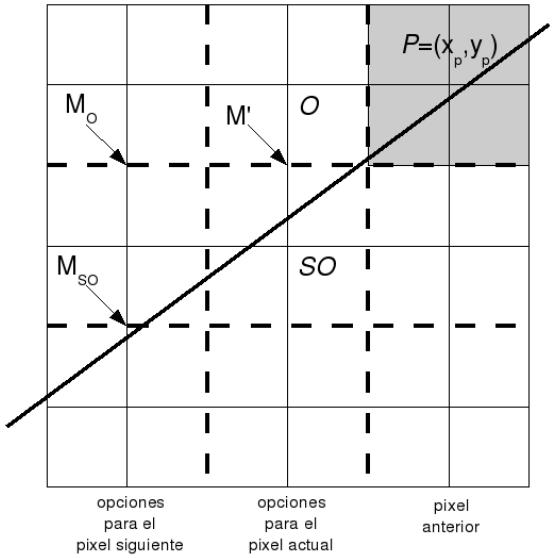


Figura 3.4: Análisis inverso de punto medio

Listing 3.4: Algoritmo de línea de punto medio de dos mitades de cuadrantes opuestos

```
void linea_punto_medio_3(int x0, int y0, int x1, int y1){
    int delta_x, delta_y, delta_E, delta_NE, d, x, y, dir_x, dir_y;
    /*
        x varía de x0 a x1, o de x1 a x0 en incrementos unitarios
    */
    delta_x = abs(x1 - x0);
    delta_y = abs(y1 - y0);
    x = x0;
    y = y0;
    d = delta_y * 2 - delta_x;
    delta_E = delta_y * 2;
    delta_NE = (delta_y - delta_x) * 2;

    // Indican la dirección en que la línea debe seguir
    dir_x = (x1-x0)>0 ? 1 : -1;
    dir_y = (y1-y0)>0 ? 1 : -1;

    marcarPixel(x, y); //marcar el primero
    while(x != x1){
        if(d <= 0){
            d += delta_E;
        }else{
            d += delta_NE;
            y+=dir_y;
        }
```

```
26          x+=dir_y;
27          marcarPixel(x, y);
28      }
29  }
```

Finalmente, el código en lenguaje C para el algoritmo de línea de punto medio genérico (que considera avance hacia atrás como el 3.4 y pendientes arbitrarias como el 3.3) es:

Listing 3.5: Algoritmo de línea de punto medio genérico

```
1  void linea_punto_medio(int x0, int y0, int x1, int y1){
2
3      int delta_x, delta_y, delta_E, delta_NE, d, x, y, dir_x, dir_y;
4      delta_x=abs(x1-x0); delta_y=abs(y1-y0);
5      x=x0, y=y0;
6
7      // Cuando 'horizontal' es verdadero, el ciclo iterará a través de x,
           sino, lo hará a través de y
8      int horizontal = delta_y < delta_x;
9
10     // Indican la dirección en que la línea debe seguir
11     dir_x = (x1-x0)>0 ? 1 : -1;
12     dir_y = (y1-y0)>0 ? 1 : -1;
13
14     // Inicializaciones
15     if(horizontal){
16         d=(delta_y*2)-delta_x;
17         delta_E=delta_y*2;
18         delta_NE=(delta_y-delta_x)*2;
19     }else{
20         d=(delta_x*2)-delta_y;
21         delta_E=delta_x*2;
22         delta_NE=(delta_x-delta_y)*2;
23     }
24
25     // Dibujar el primer pixel
26     marcarPixel(x,y);
27     if(horizontal)
28         while(x != x1){
29             if(d <= 0){
30                 d += delta_E;
31             }else{
32                 d += delta_NE;
33                 y+=dir_y;
34             }
35             x+=dir_x;
36             marcarPixel(x, y);
37         }
38     else
```

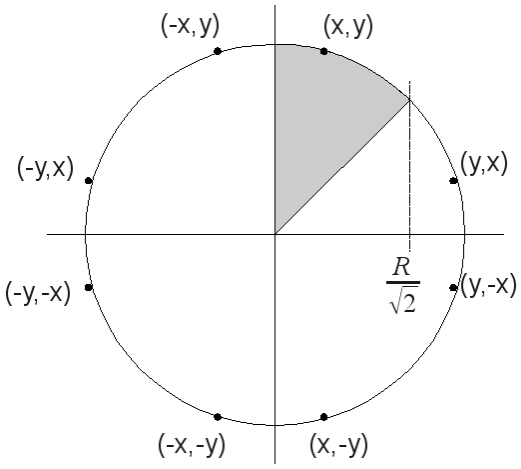


Figura 3.5: Simetría de las circunferencias

```
39          while(y != y1){
40              if(d <= 0){
41                  d += delta_E;
42              }else{
43                  d += delta_NE;
44                  x+=dir_x;
45              }
46              y+=dir_y;
47              marcarPixel(x, y);
48          }
49  }
```

3.5. La simetría de la circunferencia

El círculo se divide en ocho partes iguales, y si se calculan los puntos de un octante, se pueden calcular, por simetría, los puntos de los demás octantes, como puede apreciarse en la figura 3.5

Podría implementarse un procedimiento como el siguiente para aprovechar este fenómeno:

Listing 3.6: Función de aprovechamiento de la simetría de las circunferencias

```
1  void punto_circunferencia_simetria(int x, int y){
2      marcarPixel( x, y);
3      marcarPixel( y, x);
4      marcarPixel( y,-x);
```

```
5       marcarPixel( x,-y);
6       marcarPixel(-x,-y);
7       marcarPixel(-y,-x);
8       marcarPixel(-y, x);
9       marcarPixel(-x, y);
10  }
```

3.6. Algunas ideas sobre circunferencias

Existen varias alternativas para dibujar un círculo. Entre ellas, dibujarla a partir de la ecuación:

$$x^2 + y^2 = R^2 \Rightarrow y = \pm\sqrt{R^2 - x^2} \tag{3.10}$$

Pero esta alternativa provoca algunos efectos no deseables cuando $x \rightarrow R$, además del enorme consumo de recursos para ejecutar las multiplicaciones y la raíz cuadrada.

Otra alternativa, es dibujarla en su forma paramétrica: $(r\cos\theta, r\sin\theta), 0 \leq \theta \leq 2\pi$. Esta es menos ineficiente, pero aún demasiado, debido al cálculo de las funciones trigonométricas (que se realizan con series de potencias).

3.7. Algoritmo de circunferencia de punto medio

La alternativa más eficiente es el *algoritmo de punto medio para circunferencia* o *algoritmo de círculo de Bresenham*. A continuación se presentará la solución básica en el caso de circunferencias centradas en el origen para comprender la idea general.

En la figura 3.6 aparece un diagrama para explicar la idea, que es la misma del algortimo de punto medio para líneas.

Para construir el código se procederá de manera similar. Se usará la siguiente ecuación del círculo, la ecuación implícita:

$$F(x, y) = x^2 + y^2 - R^2 = 0 \tag{3.11}$$

El valor de F es cero en el círculo, positivo fuera, y negativo dentro, siempre y cuando los coeficientes de x^2 y y^2 sean positivos.

Como se hizo con las líneas, la decisión se basa en la variable d:

$$d_{viejo} = F(M) = F(x_p + 1, y_p - \frac{1}{2}) = (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2$$

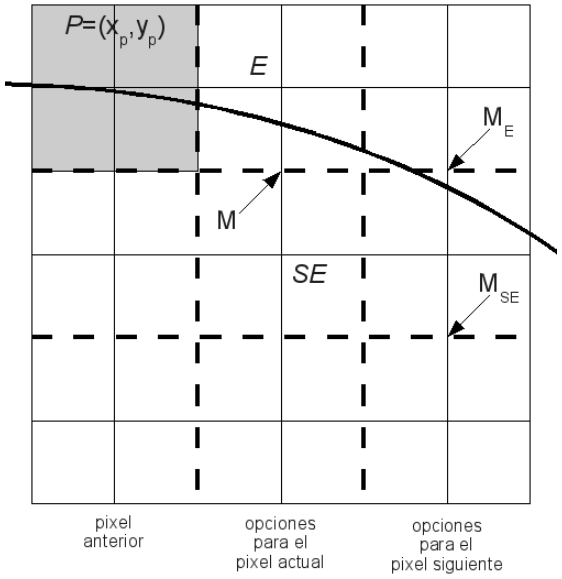


Figura 3.6: Algoritmo de circunferencia de punto medio

Si $d_{viejo} < 0$, elegimos avanzar al pixel E:

$$
\begin{aligned}
d^E_{nuevo} &= F\left((x_p+1)+1, y_p - \frac{1}{2}\right) \\
d^E_{nuevo} &= d_{viejo} + \Delta_E \\
\Delta_E &= d^E_{nuevo} - d_{viejo} \\
&= F(M_E) - F(M) \\
&= F(x_p+2, y_p - \frac{1}{2}) - F(x_p+1, y_p - \frac{1}{2}) \\
&= (x_p+2)^2 + (y_p - \frac{1}{2})^2 - R^2 \\
&\quad -(x_p+1)^2 - (y_p - \frac{1}{2})^2 + R^2 \\
\Delta_E &= 2x_p + 3
\end{aligned}
$$

Si $d_{viejo} \geq 0$, elegimos el pixel SE:

$$
\begin{aligned}
d_{nuevo}^{SE} &= F\left((x_p+1)+1,(y_p-1)-\frac{1}{2}\right)\\
d_{nuevo}^{SE} &= d_{viejo}+\Delta_{SE}\\
\Delta_{SE} &= d_{nuevo}^{SE}-d_{viejo}\\
&= F(M_{SE})-F(M)\\
&= F(x_p+2,y_p-\frac{3}{2})-F(x_p+1,y_p-\frac{1}{2})\\
&= (x_p+2)^2+(y_p-\frac{3}{2})^2-R^2\\
&\quad -(x_p+1)^2-(y_p-\frac{1}{2})^2+R^2\\
&= (2x_p+3)+(-2y_p+2)\\
\Delta_{SE} &= 2x_p-2y_p+5
\end{aligned}
$$

Lo que falta es entonces, calcular la condición inicial. Sabemos, que el punto inicial es $(0,R)$ y por ende, el siguiente punto medio es $(1,R-\frac{1}{2})$:

$$
\begin{aligned}
d_{inicial} &= F(1,R-\frac{1}{2})\\
&= 1+(R-\frac{1}{2})^2-R^2\\
&= 1+(R^2-R+\frac{1}{4})-R^2\\
d_{inicial} &= \frac{5}{4}-R
\end{aligned}
$$

El algoritmo resultante en lenguaje C es:

Listing 3.7: Primera propuesta de algoritmo de círculo de punto medio

```
1 void circunferencia_punto_medio_1(int radio){
2     int x, y;
3     float d;
4 
5     x = 0;
6     y = radio;
7     d = 5.0 / 4 - radio;
8     marcarPixel( x, y);
9     marcarPixel( y, x);
```

```
10      marcarPixel( y,-x);
11      marcarPixel( x,-y);
12      marcarPixel(-x,-y);
13      marcarPixel(-y,-x);
14      marcarPixel(-y, x);
15      marcarPixel(-x, y);
16
17      while(y > x){
18          if(d < 0){
19              d += x * 2.0 + 3;
20          }else{
21              d += (x - y) * 2.0 + 5;
22              y--;
23          }
24          x++;
25          marcarPixel( x, y);
26          marcarPixel( y, x);
27          marcarPixel( y,-x);
28          marcarPixel( x,-y);
29          marcarPixel(-x,-y);
30          marcarPixel(-y,-x);
31          marcarPixel(-y, x);
32          marcarPixel(-x, y);
33      }
34  }
```

Sin embargo, persiste el desagradable asunto de tener que operar con una variable de coma flotante sólo por el valor inicial de d. Eso puede solucionarse de la siguiente manera:

Definimos una nueva variable de decisión: $h = d - \frac{1}{4}$ y sustituimos este valor en lugar de d en el código 3.7. El valor es ahora $h = 1 - R$ y la comparación $d < 0$ es $h < -\frac{1}{4}$. Pero como h es incrementada únicamente en valores enteros, se puede dejar la comparación como $h < 0$ sin afectar el resultado. Y por razones de consistencia, se dejará la variable d en lugar de h.

El algoritmo resultante, usando sólo aritmética entera para dibujar círculos en lenguaje C es:

Listing 3.8: Algoritmo de circunferencia de punto medio sólo con aritmética entera

```
1   void circunferencia_punto_medio_2(int radio){
2       int x, y, d;
3
4       x = 0;
5       y = radio;
6       d = 1 - radio;
7       marcarPixel( x, y);
8       marcarPixel( y, x);
9       marcarPixel( y,-x);
```

```
10      marcarPixel( x,-y);
11      marcarPixel(-x,-y);
12      marcarPixel(-y,-x);
13      marcarPixel(-y, x);
14      marcarPixel(-x, y);
15
16      while(y > x){
17          if(d < 0){
18              d += x * 2 + 3;
19          }else{
20              d += (x - y) * 2 + 5;
21              y--;
22          }
23          x++;
24          marcarPixel( x, y);
25          marcarPixel( y, x);
26          marcarPixel( y,-x);
27          marcarPixel( x,-y);
28          marcarPixel(-x,-y);
29          marcarPixel(-y,-x);
30          marcarPixel(-y, x);
31          marcarPixel(-x, y);
32      }
33  }
```

3.7.1. Versión sin multiplicaciones

El algoritmo 3.8 se puede agilizar aún más si se considera que así como se han calculado *diferencias parciales de primer* orden para d, se pueden calcular diferencias de primer orden para los incrementos de d, es decir, las *diferencias parciales de segundo orden* de d. Así:

Si elegimos E en la iteración actual, el punto de evaluación se mueve de (x_p, y_p) a $(x_p + 1, y_p)$. Tenemos entonces el siguiente panorama:

$$\Delta_{Eviejo} = 2x_p + 3$$
$$\Delta_{Enuevo} = 2(x_p + 1) + 3$$
$$\Delta_{Enuevo} - \Delta_{Eviejo} = 2$$

$$\Delta_{SEviejo} = 2x_p - 2y_p + 5$$

y tenemos además: $\Delta_{SEnuevo} = 2(x_p + 1) - 2y_p + 5$.

$$\Delta_{SEnuevo} - \Delta_{SEviejo} = 2$$

Pero si elegimos SE en la iteración actual, el punto de evaluación se mueve de (x_p, y_p) a $(x_p + 1, y_p - 1)$. Tenemos lo siguiente:

$$\Delta_{Eviejo} = 2x_p + 3$$
$$\Delta_{Enuevo} = 2(x_p + 1) + 3$$
$$\Delta_{Enuevo} - \Delta_{Eviejo} = 2$$

$$\Delta_{SEviejo} = 2x_p - 2y_p + 5$$

y tenemos además: $\Delta_{SEnuevo} = 2(x_p + 1) - 2(y_p - 1) + 5.$

$$\Delta_{SEnuevo} - \Delta_{SEviejo} = 4$$

De este análisis, surge una tercera versión del algoritmo (mucho más rápida que las anteriores, ya que sólo contiene una multiplicación):

Listing 3.9: Algoritmo de línea de punto medio sin multiplicaciones

```
void circunferencia_punto_medio_3(int radio){
    int x, y, d, delta_E, delta_SE;

    x = 0;
    y = radio;
    d = 1 - radio;
    delta_E = 3;
    delta_SE = 5 - radio * 2;
    marcarPixel( x, y);
    marcarPixel( y, x);
    marcarPixel( y,-x);
    marcarPixel( x,-y);
    marcarPixel(-x,-y);
    marcarPixel(-y,-x);
    marcarPixel(-y, x);
    marcarPixel(-x, y);

    while(y > x){
        if(d < 0){
            d += delta_E;
            delta_E += 2;
            delta_SE += 2;
        }else{
            d += delta_SE;
            delta_E += 2;
            delta_SE += 4;
            y--;
        }
        x++;
        marcarPixel( x, y);
        marcarPixel( y, x);
        marcarPixel( y,-x);
        marcarPixel( x,-y);
        marcarPixel(-x,-y);
        marcarPixel(-y,-x);
        marcarPixel(-y, x);
```

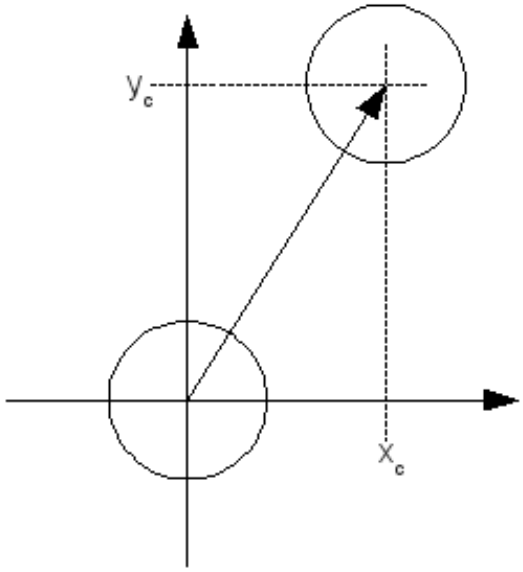


Figura 3.7: Esquema para algorito de circunferencias con centro arbitrario

```
37            marcarPixel(-x, y);
38        }
39    }
```

3.7.2. Circunferencias con centro arbitrario

Para construir un algoritmo de dibujo de circunferencias con centro arbitrario, a partir del algoritmo de punto medio de esta sección, básicamente hay que hacer un desplazamiento vectorial de cada uno de los puntos a marcar.

Consideremos la figura 3.7. En ella aparecen dos circunferencias del mismo radio. La diferencia vectorial entre cada uno de los puntos de la circunferencia centrada en el origen y la centrada en (x_c, y_c), es precísamente el vector $\overrightarrow{C} = (x_c, y_c)$.

> Otra forma de plantearlo es así: Sean (x', y') los puntos de la circunferencia con centro en (x_c, y_c). Sean (x, y) los puntos de la circunferencia con centro en el origen.
>
> Entonces $(x', y') = (x + x_c, y + y_c)$.

En base a este sencillo análisis, presentamos el siguiente algoritmo de dibujo de circunferencias con centro arbitrario, basado en el algoritmo 3.8:

Listing 3.10: Algoritmo de circunferencia de punto medio para centros arbitrarios

```
// Función auxiliar
void marcarPixelesCicunferencia(int x, int y, int xc, int yc){
    marcarPixel( x+xc, y+yc);
    marcarPixel( x+xc,-y+yc);
    marcarPixel(-x+xc, y+yc);
    marcarPixel(-x+xc,-y+yc);
    marcarPixel( y+xc, x+yc);
    marcarPixel( y+xc,-x+yc);
    marcarPixel(-y+xc, x+yc);
    marcarPixel(-y+xc,-x+yc);
}
void circunferencia_punto_medio(int xc, int yc, int radio){
    int x,y,d;
    x=0;
    y=radio;
    d=1-radio;
    marcarPixelesCicunferencia(x,y, xc,yc);
    while(y>x){
        if(d<0){
            d += x * 2 + 3;
        }else{
            d += (x - y) * 2 + 5;
            y--;
        }
        x++;
        marcarPixelesCicunferencia(x,y, xc,yc);
    }
}
```

3.8. Relleno de rectángulos

Veamos primero un par de algoritmos de dibujo de rectángulos huecos antes de pasar al de relleno:

```
void dibujar_rectangulo_1(int x1, int y1, int x2, int x2){
    /* Se asume que x1 < x2 y y1 < y2
    */
    int i;
    if(x1>x2){
        i=x1;
        x1=x2;
        x2=x1;
    }
    for(i=x1; i<=x2; i++){
        marcarPixel(i, y1);
        marcarPixel(i, y2);
```

```
13      }
14      if(y1>y2){
15          i=y1;
16          y1=y2;
17          y2=y1;
18      }
19      for(i=y1; i<=y2; i++){
20          marcarPixel(x1, i);
21          marcarPixel(x2, i);
22      }
23  }
```

```
1   void dibujar_rectangulo_2(int x1, int y1, int ancho, int alto){
2       int i;
3       for(i=x1; i< x1+ancho; i++){
4           marcarPixel(i, y1,       );
5           marcarPixel(i, y1+alto-1);
6       }
7       for(i=y1; i< y1+alto; i++){
8           marcarPixel(x1,          i);
9           marcarPixel(x1+ancho-1, i);
10      }
11  }
```

El primer algoritmo recibe como parámetros dos puntos opuestos del rectángulo (nótese que no importa el orden en que se especifiquen). El segundo recibe el punto más cercano al origen y el ancho y alto del rectángulo (nótese que tanto `ancho` como `alto`, deben ser positivos).

Una de las alternativas más usuales entre las bibliotecas gráficas, es ofrecer al programador una función de relleno de rectángulos que requiere del punto inicial del rectángulo y su ancho y su alto. Esta es una propuesta de implementación en lenguaje C:

Listing 3.11: Relleno de rectángulos

```
1   void dibujar_rectangulo_relleno(int x1, int y1, int ancho, int alto){
2       int i, j;
3       for(i=x1; i< x1+ancho; i++)
4           for(j=y1; j< y1+alto; j++)
5               marcarPixel(i, j);
6   }
```

3.9. Relleno de circunferencias

Para el relleno de circunferencias, también existen diferentes propuestas. Una de ellas es aprovechar el algoritmo de punto medio para circunferencias y la simetría propia de

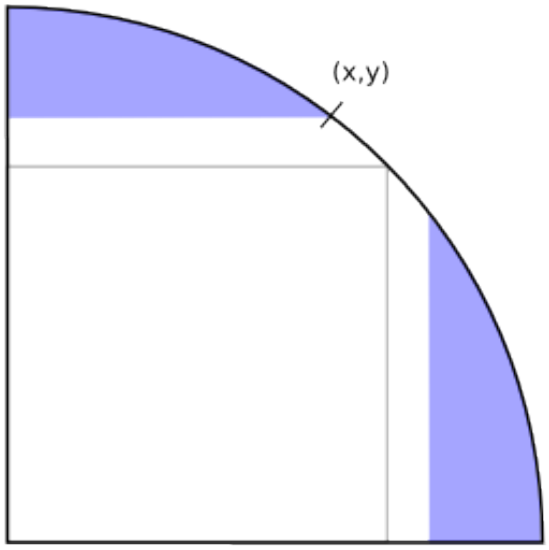


Figura 3.8: Relleno de circunferencias

esta hermosa figura geométrica.

La idea básica es hacer una modificación al algoritmo de punto medio para circunferencias de la siguiente manera: Cuando se haga un avance hacia el pixel SE, aprovechar para marcar todos los pixeles desde $(0, y)$ hasta (x, y) (y al mismo tiempo aprovechar para marcar los pixeles correspondientes con la misma idea de la simetría de ocho octantes). A continuación se presenta el código fuente de dicha solución[5]:

Listing 3.12: Algoritmo de relleno de circunferencias de centro arbitrario (basado en el código 3.10 y 3.11)

```
1  void circunferencia_rellena(int x0, int y0, int radio){
2      int r_x, r_y, r_ancho, r_alto;
3      int x,y,d,i;
4      x=0;
5      y=radio;
6      d=1-radio;
7      marcarPixelesCicunferencia(x,y, x0,y0);
8      while(y>x){
9          if(d<0){
10             d += x * 2 + 3;
11             x++;
12             marcarPixelesCicunferencia(x,y, x0,y0);
13         }else{
14             d += (x - y) * 2 + 5;
```

[5] comparar con el código de la función `circunferencia_punto_medio` de la subsección en la página 101

```
15              y--;
16              x++;
17              for(i=0;i<=x; i++){
18                  marcarPixelesCicunferencia(i,y, x0,y0);
19                  marcarPixelesCicunferencia(y,i, x0,y0);
20              }
21          }
22      }
23      r_x = x0-x;
24      r_y = y0-y;
25      r_ancho = 2*x;
26      r_alto = 2*y;
27      dibujar_rectangulo_relleno(r_x, r_y, r_ancho, r_alto);
28  }
```

3.10. Ejercicios

1. Construya un algoritmo que use la idea del DDA (código 3.1 en la página 83) para pendientes arbitrarias.
2. Describa cómo funciona el algoritmo de dibujo de líneas conocido como algoritmo de línea de punto medio.
3. Implementar un programa de dibujo usando la ecuación 3.10 en la página 95.
4. En la cuadrícula de la figura 3.9, marque (sombree, tache, repinte, etc.) los pixeles que encendería el algoritmo de línea 3.5, al unir a los puntos (2,2) y (9,5) y a los puntos (0,2) y (3,9). El centro geométrico de los pixeles está en el centro de los cuadritos negros. Las líneas grices son sólo guías.
5. En la cuadrícula de la figura 3.9, marque (sombree, tache, repinte, etc.) los pixeles que encendería el algoritmo de círculo 3.10 en la página 102, con centro (4,4) y radio 4. El centro geométrico de los pixeles está en el centro de los cuadritos negros. Las líneas grices son sólo guías.
6. Modificar el algoritmo 3.9 en la página 100 para dibujar círculos con centro arbitrario.
7. Modificar el algoritmo del ejercicio anterior para implementar el dibujo de círculos rellenos.
8. Considere los códigos siguientes para rellenar círculos y explique por qué el primero es preferible frente al segundo:

```
1  void marcarPixelesCicunferencia(int x, int y){
2      marcarPixel( x, y);
```

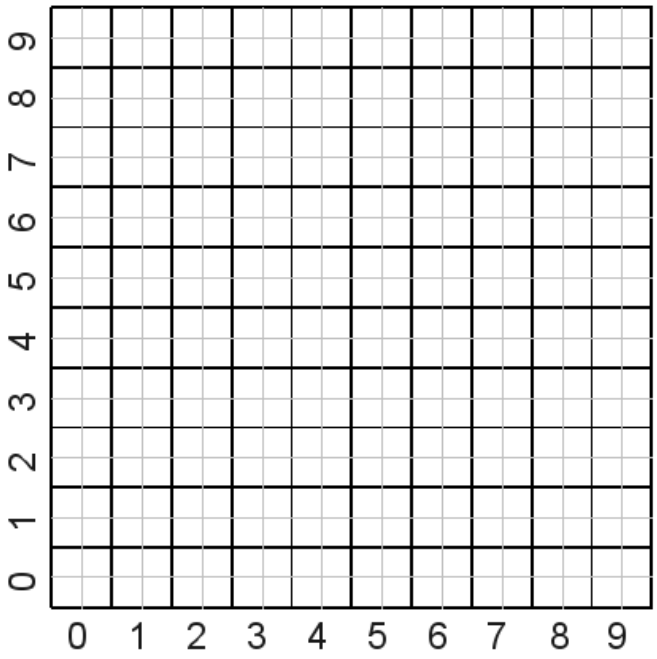


Figura 3.9: Cuadrícula de práctica de primitivas gráficas

```
3       marcarPixel( x,-y);
4       marcarPixel(-x, y);
5       marcarPixel(-x,-y);
6       marcarPixel( y, x);
7       marcarPixel( y,-x);
8       marcarPixel(-y, x);
9       marcarPixel(-y,-x);
10  }
11
12  void dibujarCirculoRelleno1(int radio){
13      int x,y,d,i;
14      x=0;
15      y=radio;
16      d=1-radio;
17      marcarPixelesCicunferencia(x,y);
18      while(y>x){
19          if(d<0){
20              d += x * 2 + 3;
21              x++;
22              marcarPixelesCicunferencia(x,y);
23          }else{
24              d += (x - y) * 2 + 5;
25              y--;
```

```
26              x++;
27              for(i=0;i<=x; i++){
28                  marcarPixelesCicunferencia(i,y);
29                  marcarPixelesCicunferencia(y,i);
30              }
31          }
32      }
33      dibujar_rectangulo_relleno(-x,-y, 2*x,2*y);
34  }
35
36  void dibujarCirculoRelleno2(int radio){
37      int i;
38      for(i=0; i<=radio; i++)
39          dibujarCircunferencia(i);
40  }
```

9. Considere los códigos siguientes y explique por qué el primero es más eficiente que el segundo:

```
1   void rectanguloRelleno1(int x, int y, int ancho, int alto){
2       int i, j, xmax=x+ancho, ymax=y+alto;
3       for(i=x; i< xmax; i++){
4           for(j=y; j< ymax; j++)
5               marcarPixel(i, j);
6   }
7
8   void rectanguloRelleno2(int x, int y, int ancho, int alto){
9       int i, xmax=x+ancho, ymax=y+alto;
10      for(i=x; i< xmax; i++)
11          linea_punto_medio(i, y, i, ymax);
12  }
```

10. Implementar una función de dibujo de rectángulos que reciba uno de los vértices del rectángulo y su ancho y alto. Pero el alto y el ancho debe aceptarlos con signo negativo, lo que significa que el rectángulo debe dibujarse *hacia atrás* en esa dimensión.

11. Implementar una función de relleno de rectángulos que reciba como parámetros, dos puntos opuestos del mismo.

12. Implemente la siguiente función en lenguaje C estándar
`void dibujarPoligono(int x[],int y[],int nPuntos);`
Esta función debe dibujar un polígono cerrado, formado por los puntos dados. Por *polígono cerrado*, nos referimos a que el último punto se une con el primero aunque no sean iguales.

13. Construir una función de dibujo de elipses, dados su centro y sus dos radios.

14. Construir una función de relleno de elipses, dados su centro y sus dos radios.
15. Construir una función de dibujo de elipses, dado el rectángulo que la circunscribe.

4 Marcos de Referencia y Cambio de Coordenadas

Es usual que se trabaje en los programas de aplicación, con marcos de referencia diferentes del marco de referencia establecido por la biblioteca gráfica a utilizar, principalmente para proveer al código la abstracción necesaria para que la aplicación sea portable y coherente al modelo que se esté graficando. Además, ayuda mucho a la comprensión por parte de otros programadores.

Este capítulo contempla una introducción al tema de cambio de coordenadas entre diversos marcos de referencia. Y la herramienta fundamental que se usará será el álgebra vectorial, por lo que se recomienda hacer un buen repaso de dichos temas, si es que no se tienen ya a la mano.

4.1. Notación

Para recorrer el presente capítulo con tranquilidad, vamos a establecer la siguiente nomenclatura:

Sea $\overrightarrow{p}^{(R)}$ el vector del punto p en el marco de referencia R, y se lee "el vector p en R".

Las componentes rectangulares de $\overrightarrow{p}^{(R)}$ se representarán en este libro como $\overrightarrow{p}_x^{(R)}$ y $\overrightarrow{p}_y^{(R)}$, y se leen "la componente x del vector p en R" y "la componente y del vector p en R" respectivamente.

Sea $\Delta^{(R)}$ una distancia Δ en la escala del marco de referencia R, y se lee "Δ en R".

Recordemos que un punto cualquiera, puede existir simultaneamente en una cantidad infinita de marcos de referencia. Por ejemplo, veamos la figura 4.1.

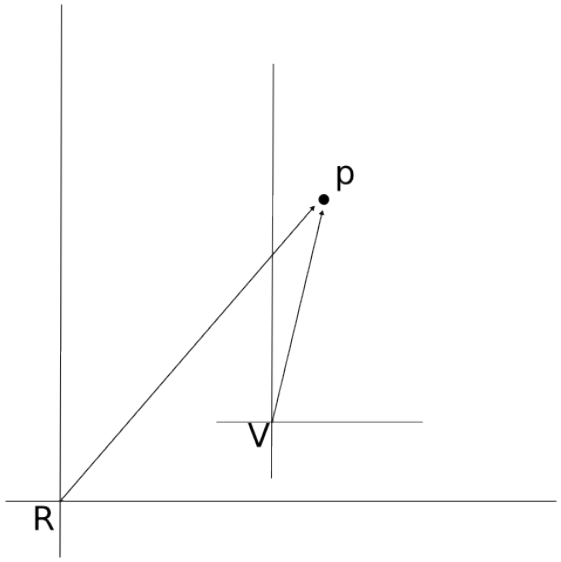


Figura 4.1: Un mismo punto en dos marcos de referencia diferentes

En ella aparece el punto p en el marco de referencia V, y en el marco de referencia R. En cada uno de ellos, el punto tendrá sus coordenadas medidas en una escala diferente, apropiada para cada marco.

Usualmente trabajaremos con dos marcos de referencia, uno *Virtual*, propio de la lógica de nuestra aplicación, y uno *Real*, propio de la API gráfica que estemos utilizando (típicamente coincidente con los pixeles físicos del monitor).

El problema a analizar en este capítulo es básicamente el siguiente:

> Disponemos de $\overrightarrow{p}^{(V)}$, y deseamos conocer $\overrightarrow{p}^{(R)}$ (y/o viceversa) para que nuestra manipulación programática sea congruente y nuestro código fuente sea ordenado y claro.

4.2. Análisis vectorial del cambio de coordenadas

Habiendo hecho la respectiva presentación de la nomenclatura, pasamos al tema que nos interesa: *Cómo cambiar las coordenadas de un punto (u objeto) de un marco de referencia a otro.*

Bueno, el problema puede abordarse de muy diversas maneras. Una de ellas es la perspectiva vectorial.

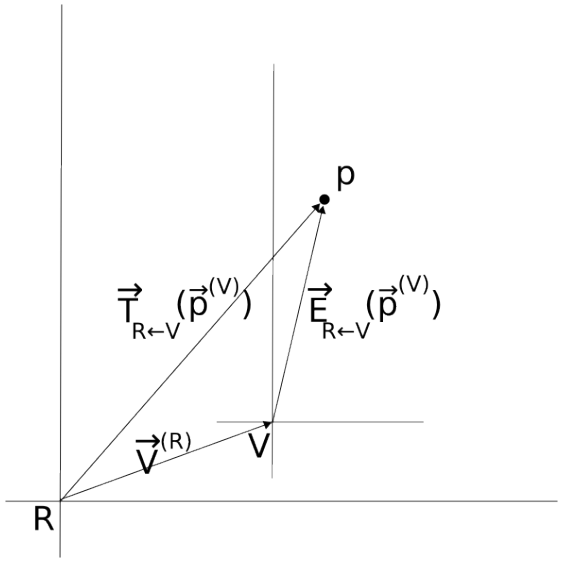


Figura 4.2: Transformación vectorial de coordenadas

Veamos la figura 4.2. En ella aparece el mismo punto p, visto de otra manera:

$$\overrightarrow{p}^{\,(R)} = \overrightarrow{T}_{R\leftarrow V}\left(\overrightarrow{p}^{\,(V)}\right) = \overrightarrow{V}^{\,(R)} + \overrightarrow{E}_{R\leftarrow V}\left(\overrightarrow{p}^{\,(V)}\right)$$

.

Veamos estos nuevos elementos: $\overrightarrow{T}_{R\leftarrow V}$ es una función vectorial que transforma el vector de entrada, del marco de referencia V al R (esta es la que buscamos definir). $\overrightarrow{V}^{\,(R)}$ es el vector del origen de V en R. $\overrightarrow{E}_{R\leftarrow V}$ es una función vectorial que cambia la escala de las componentes del vector de entrada, del marco de referencia V al R.

El vector $\overrightarrow{V}^{\,(R)}$ usualmente se conoce, o puede calcularse de acuerdo a la propia distribución gráfica de nuestra aplicación. La función $\overrightarrow{E}_{R\leftarrow V}$ no es algo dado, normalmente, sin embargo puede definirse de la siguiente manera:

Antes que nada, necesitamos las coordenadas de dos puntos en ambos marcos de referencia. Llamémoslos A y B. Consideremos la figura 4.3.

Nótese que $\Delta_x^{(R)} = B_x^{(R)} - A_x^{(R)}$, $\Delta_x^{(V)} = B_x^{(V)} - A_x^{(V)}$, del mismo modo que $\Delta_y^{(R)} = B_y^{(R)} - A_y^{(R)}$ y $\Delta_y^{(V)} = B_y^{(V)} - A_y^{(V)}$. Es importante mantener los signos tal cuales. Con este panorama, es fácil deducir que:

$$\overrightarrow{E}_{R\leftarrow V}\left(\overrightarrow{p}^{\,(V)}\right) = \left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}}\,\overrightarrow{p}_x^{\,(V)}, \frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\,\overrightarrow{p}_y^{\,(V)}\right)$$

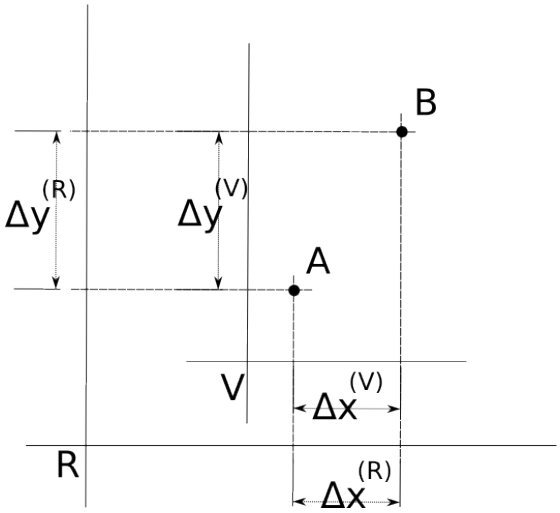


Figura 4.3: Cambio de escala de vectores

Así, la forma completa para la ecuación vectorial de transformación de marcos de referencia es:

$$\overrightarrow{p}^{\,(R)} = \overrightarrow{T}_{R \leftarrow V}\left(\overrightarrow{p}^{\,(V)}\right) = \overrightarrow{V}^{\,(R)} + \left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}}\,\overrightarrow{p}_x^{\,(V)}, \frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\,\overrightarrow{p}_y^{\,(V)}\right) \tag{4.1}$$

Si necesitaramos invertir el proceso, es decir, calcular $\overrightarrow{p}^{\,(V)}$, dado un $\overrightarrow{p}^{\,(R)}$, procedemos de forma inversa:

$$\overrightarrow{p}^{\,(V)} = \overrightarrow{T}_{V \leftarrow R}\left(\overrightarrow{p}^{\,(R)}\right) = \overrightarrow{R}^{\,(V)} + \left[\frac{\Delta_x^{(V)}}{\Delta_x^{(R)}}\,\overrightarrow{p}_x^{\,(R)}, \frac{\Delta_y^{(V)}}{\Delta_y^{(R)}}\,\overrightarrow{p}_y^{\,(R)}\right] \tag{4.2}$$

4.2.1. Ejemplo

Planteemos un escenario en el que es necesario el cambio de marco de referencia: Tenemos una aplicación gráfica como en la figura 4.4. La aplicación tiene su propio marco de referencia establecido por la API gráfica que estamos utilizando. Esta API establece (y la mayoría lo hace de la misma manera) que el origen está en la esquina superior izquierda del interior de la ventana. Todas las coordenadas delimitadas por paréntesis pertenecen al sistema de coordenadas de la API. Este marco será R.

Dentro de nuestra aplicación gráfica tenemos un cuadro (en este caso, punteado) a través del cual mostramos un juego que tiene su propio marco de referencia, que establece que

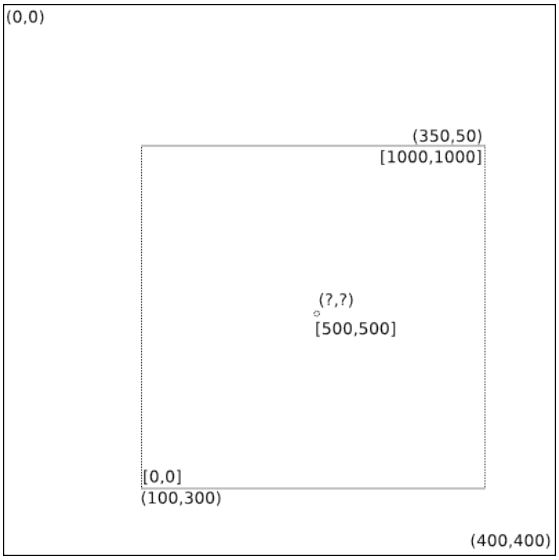


Figura 4.4: Situación de ejemplo

el origen está en la esquina inferior izquierda del cuadro, y que las coordenadas visibles para el usuario se extienden hacia la esquina superior derecha hasta la coordenada $[1000, 1000]$, independientemente del tamaño en pixeles del subcuadro. Todas las coordendadas delimitadas por corchetes están en el marco de referencia del juego, que llamaremos V.

Nótese que los ejes verticales avanzan en dirección opuesta, uno del otro. Y que por convención en este texto, las coordenadas en el marco de referencia virtual se delimitan con corchetes: $V \rightarrow [\ ]$. Y que las coordenadas en el marco de referencia real, se delimitan con paréntesis: $R \rightarrow (\ )$.

Se nos plantea entonces un problema: El juego necesita marcar el punto $\overrightarrow{p}^{(V)} = [500, 500]$. Pero nuestra API gráfica no funciona con ese marco de referencia. Necesitamos indicarle el pixel específico que deseamos marcar. ¿Cómo hacerlo?

El problema se resuelve de la siguiente manera: Tenemos $\overrightarrow{p}^{(V)}$ y necesitamos $\overrightarrow{p}^{(R)}$. Podemos ver que $\overrightarrow{V}^{(R)} = (100, 300)$. Procedemos entonces a buscar nuestros puntos A y B. Podemos tomar estos de las esquinas inferior izquierda y superior derecha del cuadro del juego. Entonces $A^{(R)} = (100, 300)$, $B^{(R)} = (350, 50)$, $A^{(V)} = [0, 0]$ y $B^{(V)} = [1000, 1000]$. Ahora sustituímos:

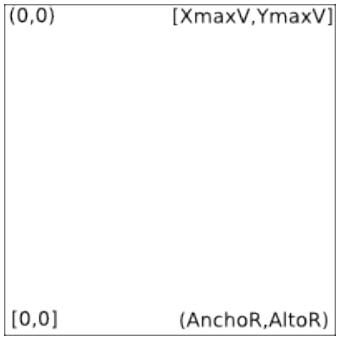


Figura 4.5: Caso particular de pantalla completa

$$
\begin{aligned}
\overrightarrow{p}^{(R)} &= \overrightarrow{T}_{R\leftarrow V}\left(\overrightarrow{p}^{(V)}\right) \\
&= \overrightarrow{V}^{(R)} + \left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}}\,\overrightarrow{p}_x^{(V)}, \frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\,\overrightarrow{p}_y^{(V)}\right) \\
&= (100, 300) + \left(\frac{350-100}{1000-0}\times 500, \frac{50-300}{1000-0}\times 500\right) \\
&= (100, 300) + (125, -125) \\
\overrightarrow{p}^{(R)} &= (225, 175)
\end{aligned}
$$

Ahora sabemos que si en el juego se necesita marcar el punto $[500, 500]$ en el marco de referencia interno, debemos marcar el pixel $(225, 175)$ de la aplicación.

4.3. Simplificación escalar para ventana completa

Hay un caso particular que es muy usual en aplicaciones de corte académico: Aquel en el el que toda la pantalla (o al menos toda la ventana) de la aplicación, se usa para el despliegue gráfico en un marco de referencia con origen en la esquina inferior izquierda, tal como puede verse en la figura 4.5.

En este caso, tal como en el ejemplo de la sección 4.4, los puntos a elegir como referencias, son las esquinas inferior izquierda y superior derecha del marco virtual en el marco real.

Podemos ver que $\overrightarrow{V}^{(R)} = (0, AltoR)$. Además $A^{(R)} = (0, AltoR)$, $B^{(R)} = (AnchoR, 0)$, $A^{(V)} = [0, 0]$ y $B^{(V)} = [XmaxV, YmaxV]$ y sustituimos:

$$
\begin{aligned}
\overrightarrow{p}^{(R)} &= \overrightarrow{T}_{R\leftarrow V}\left(\overrightarrow{p}^{(V)}\right) \\
&= \overrightarrow{V}^{(R)} + \left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}}\,\overrightarrow{p}_x^{(V)}, \frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\,\overrightarrow{p}_y^{(V)}\right) \\
&= (0, AltoR) + \left(\frac{AnchoR - 0}{XmaxV - 0}\cdot\overrightarrow{p}_x^{(V)}, \frac{0 - AltoR}{YmaxV - 0}\cdot\overrightarrow{p}_y^{(V)}\right) \\
&= \left(\frac{AnchoR}{XmaxV}\cdot\overrightarrow{p}_x^{(V)},\ AltoR - \frac{AltoR}{YmaxV}\cdot\overrightarrow{p}_y^{(V)}\right) \\
\overrightarrow{p}^{(R)} &= \left(\frac{AnchoR}{XmaxV}\cdot\overrightarrow{p}_x^{(V)},\ AltoR\left(1 - \frac{\overrightarrow{p}_y^{(V)}}{YmaxV}\right)\right)
\end{aligned}
$$

Lo que significa que:

$$\frac{\overrightarrow{p}_x^{(R)}}{AnchoR} = \frac{\overrightarrow{p}_x^{(V)}}{XmaxV}$$

y

$$\frac{\overrightarrow{p}_y^{(R)}}{AltoR} = 1 - \frac{\overrightarrow{p}_y^{(V)}}{YmaxV}$$

4.4. Transformación de distancias

Eventualmente aparece también el problema de transformar no sólo coordenadas, sino también *distancias* entre diferentes marcos de referencia.

El problema se resuelve intuitivamente haciendo una regla de tres con las distancias así: $\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}} = \frac{d_x^{(R)}}{d_x^{(V)}}$ para las magnitudes en x y $\frac{\Delta_y^{(R)}}{\Delta_y^{(V)}} = \frac{d_y^{(R)}}{d_y^{(V)}}$ para las magnitudes en y. De hecho, es el mismo razonamiento usado para definir la ecuación 4.1 en la página 112.

Es de hacer notar que de esta manera, las distancias quedan direccionadas. Es decir, no sólo se traduce la magnitud de la distancia, sino también su sentido. Si el propósito es traducir sólo la magnitud de la distancia, habrá que ignorar (o eliminar) el signo resultante.

4.5. Aplicación: Simulador de campo eléctrico bidimensional

A continuación se expone brevemente una sencila aplicación gráfica interactiva que simula el campo eléctrico producido por un conjunto de cargas puntuales en un espacio

continuo bidimensional[1].

4.5.1. Campo Eléctrico

Si tenemos una carga puntual q en el punto Q, entonces el campo eléctrico $\overrightarrow{E}$ en el punto P es:

$$\overrightarrow{E} = \frac{1}{4\pi\epsilon_0}\frac{q}{\left|\overrightarrow{QP}\right|^3}\overrightarrow{QP} \tag{4.3}$$

Si tenemos una serie de cargas puntuales q_i en los puntos Q_i respectivamente, con $i = 0, 1, \ldots, n$, entonces el campo eléctrico $\overrightarrow{E}$ en el punto P es:

$$\overrightarrow{E} = \frac{1}{4\pi\epsilon_0}\sum_{i=0}^{n}\frac{q_i}{\left|\overrightarrow{Q_iP}\right|^3}\overrightarrow{Q_iP} \tag{4.4}$$

Los siguientes dos archivos proporcionan la funcionalidad básica para manipular campo eléctrico.

Listing 4.1: Archivo de cabecera de funciones de campo eléctrico

```
1  /* c04/campos vectoriales/campo.h
2   * */
3  #define EPSILON_0 8.854e-12
4  #define _1_4PIe0 8.988e9
5
6  /*
7  Vector genérico bidimensional,
8  se puede usar para posición, velocidad, etc.
9  */
10 typedef struct{
11     double x,y;
12 } Vector;
13
14 /*
15 <valor> puede ser negativo.
16 */
17 typedef struct{
```

[1] Para más detalles del dicho fenómeno físico, consúltese cualquier texto introductorio de Electromagnetismo

```
18      Vector posicion;
19      double valor;
20  } CargaElectrica;
21
22  /*
23  Calcula el vector de Campo Eléctrico
24  en el punto <punto>, por efecto de la
25  carga <carga>.
26  El resultado se almacena en <campo>.
27  */
28  void campoElectrico(CargaElectrica *carga, Vector *punto, Vector* campo);
29
30  /*
31   * Calcula el vector de Campo Eléctrico
32   * en el punto <punto>, por efecto del
33   * arreglo de cargas <cargas>.
34   * El resultado se almacena en <campo>
35   * */
36  void campoElectricoCargas(CargaElectrica *cargas, int cantidad, Vector *
        punto, Vector* campo);
37
38  /*
39   * Calcula la magnitud de un vector
40   * */
41  double magnitudVector(Vector *v);
```

Listing 4.2: Código fuente de funciones de campo eléctrico

```
1   /* c04/campos vectoriales/campo.c
2    * */
3   #include <math.h>
4   #include "campo.h"
5
6   void campoElectrico(CargaElectrica *carga, Vector *punto, Vector* campo){
7       Vector distancia;
8       double magnitudDistancia3;
9       distancia.x = punto->x - carga->posicion.x;
10      distancia.y = punto->y - carga->posicion.y;
11      magnitudDistancia3 = pow(magnitudVector(&distancia),3.0);
12      campo->x = _1_4PIe0 * carga->valor * distancia.x / magnitudDistancia3
            ;
13      campo->y = _1_4PIe0 * carga->valor * distancia.y / magnitudDistancia3
            ;
14  }
15
16  void campoElectricoCargas(CargaElectrica *cargas, int cantidad, Vector *
        punto, Vector* campo){
17      int k;
18      Vector temp;
19      campo->x = campo->y = 0.0;
```

```
20      for(k=0;k<cantidad;k++){
21          campoElectrico(cargas+k, punto, &temp);
22          campo->x += temp.x;
23          campo->y += temp.y;
24      }
25  }
26
27  double magnitudVector(Vector *v){
28      return sqrt(v->x*v->x + v->y*v->y);
29  }
```

Nótese que el código de este último archivo es altamente suceptible de ser optimizado para mejorar el rendimiento general de la aplicación.

4.5.2. Uso de colores para SDL_gfxPrimitives

Los siguientes dos archivos proporcionan funcionalidad básica para manipular colores en el formato que la biblioteca SDL_gfxPrimitives requiere para funcionar correctamente.

Listing 4.3: Archivo de cabecera para el formato de color de gfx

```
1   /* c04/campos vectoriales/colores.h
2    * */
3   #include <SDL/SDL.h>
4
5   //color sólido
6   #define ALFA  0x000000FF
7
8   #define ROJO  (0xFF000000|ALFA)
9   #define VERDE (0x00FF0000|ALFA)
10  #define AZUL  (0x0000FF00|ALFA)
11  #define BLANCO (ROJO|VERDE|AZUL)
12  #define NEGRO 0x000000FF
13
14  /*
15      Las funciones ____Color de SDL_gfx
16      requieren el color en un entero de 32 bis
17      así: 0xRRGGBBAA, no acepta el color
18      en ningún otro formato.
19
20      Además, utiliza las transparencias por defecto,
21      siempre hay que especificar la opacidad del color.
22  */
23  Uint32 colorGfx(Uint8 r, Uint8 g, Uint8 b);
24
25  Uint32 colorGfxA(Uint8 r, Uint8 g, Uint8 b, Uint8 a);
26
27
```

```
28 //Borra la pantalla rellenando con el color especificado
29 void borrarPantalla(SDL_Surface *pantalla, Uint32 relleno);
```

Listing 4.4: Código fuente de colores para gfx

```
1  /* c04/campos vectoriales/colores.c
2   * */
3  #include "colores.h"
4
5  Uint32 colorGfx(Uint8 r, Uint8 g, Uint8 b){
6      return
7          r << 24 |
8          g << 16 |
9          b << 8  |
10         255;    //este valor es la opacidad del color
11                 //y debe ser máxima para que el color
12                 //sea sólido
13 }
14
15 Uint32 colorGfxA(Uint8 r, Uint8 g, Uint8 b, Uint8 a){
16     return
17         r << 24 |
18         g << 16 |
19         b << 8  |
20         a;
21 }
22
23 void borrarPantalla(SDL_Surface *pantalla, Uint32 relleno){
24     SDL_FillRect(pantalla, NULL, SDL_MapRGB (pantalla->format,
25         (relleno&ROJO)>>24,
26         (relleno&VERDE)>>16,
27         (relleno&AZUL)>>8));
28 }
```

4.5.3. Las escalas y sus conversiones

Los siguientes dos archivos proporcionan la capa de abstracción que independiza las dos escalas usadas: La escala interna del modelo, que un universo bidimensional continuo, virtualmente ilimitado donde sólo existen cargas eléctricas puntuales fijas. Y la escala de la biblioteca gráfica usada, que, como suele suceder, es una matriz discreta y finita de pixeles.

Listing 4.5: Archivo de cabecera de funciones de escala

```
1 /* c04/campos vectoriales/escala.h
2  * Esta es una versión particular del problema
3  * de la transformación de escalas.
```

```
 * No es una solución general para cualquier caso.
 * */
#include <stdio.h>

/*
 * Estructura para manipular escala
 * de pantalla completa
 * */
typedef struct {
    int AnchoR, AltoR;
    double XminV, YminV;
    double XmaxV, YmaxV;
} Escala;

/*
 * Funciones de transformación vectorial
 * entre el marco de referencia Real y el Virtual
 *
 * La primera función se lee:
 * "Transformación a Real, dado el Virtual en x"
 * */
int tR_Vx(double xV, Escala *e);
int tR_Vy(double yV, Escala *e);
double tV_Rx(int xR, Escala *e);
double tV_Ry(int yR, Escala *e);

/*
 * Funciones de tranformación de distancias
 * entre el marco de referencia Real y el Virtual
 *
 * La primera función se lee:
 * "Magnitud en Real, dado el Virtual en x"
 * */
int mR_Vx(double deltaxV, Escala *e);
int mR_Vy(double deltayV, Escala *e);
double mV_Rx(int deltaxR, Escala *e);
double mV_Ry(int deltayR, Escala *e);

void imprimirEscala(FILE *salida, Escala *e);
```

Listing 4.6: Código fuente de funciones de escala

```
/* c04/campos vectoriales/escala.c
 * */
#include "escala.h"

int tR_Vx(double xV, Escala *e){
    return (int) ((xV - e->XminV) * e->AnchoR / (e->XmaxV - e->XminV));
}

```

```
9  int tR_Vy(double yV, Escala *e){
10     return (int) ((yV - e->YmaxV) * -e->AltoR / (e->YmaxV - e->YminV));
11 }
12
13 double tV_Rx(int xR, Escala *e){
14     return e->XminV + xR * (e->XmaxV-e->XminV)/e->AnchoR;
15 }
16
17 double tV_Ry(int yR, Escala *e){
18     return e->YmaxV - (e->YmaxV-e->YminV)*yR/e->AltoR;
19 }
20
21 int mR_Vx(double deltaxV, Escala *e){
22     return deltaxV*e->AnchoR/(e->XmaxV-e->XminV);
23 }
24
25 int mR_Vy(double deltayV, Escala *e){
26     return deltayV * -e->AltoR/(e->YmaxV-e->YminV);
27 }
28
29 double mV_Rx(int deltaxR, Escala *e){
30     return deltaxR*(e->XmaxV-e->XminV)/e->AnchoR;
31 }
32
33 double mV_Ry(int deltayR, Escala *e){
34     return deltayR*(e->YmaxV-e->YminV) / -e->AltoR;
35 }
36
37 void imprimirEscala(FILE *salida, Escala *e){
38     fprintf(salida, "Datos de escala:\n");
39     fprintf(salida, "AnchoR:\t%d\nAltoR:\t%d\n", e->AnchoR, e->AltoR);
40     fprintf(salida, "[%f,%f]-[%f,%f]\n", e->XminV, e->YminV, e->XmaxV, e
           ->YmaxV);
41 }
```

4.5.4. Programa principal

A continuación se presenta la aplicación principal que controla los eventos producidos por el usuario.

El programa tiene la funcionalidad de que el tamaño de la ventana gráfica es independiente de la escala del modelo.

Los comandos de interacción son los siguientes:

clic izquierdo Sobre una carga no tiene efecto. Sobre el campo tiene el efecto de imprimir en consola el vector de campo y su magnitud medido en N/C.

clic izquierdo sostenido Sobre una carga tiene el efecto desplazar esa carga a otro punto del plano. Sobre el campo no tiene ningún efecto.

clic derecho Agrega una nueva carga eléctrica puntual de $1\,C$ (dicho valor está indicado en la macro `CARGA_INICIAL`) si es que aún no se ha alcanzado el máximo número de cargas permitidas (indicado en la macro `MAX_CARGAS`).

clic medio Sobre una carga, tiene el efecto de cambiarle el signo a su valor. Sobre el campo no tiene ningún efecto.

rueda del mouse arriba Sobre una carga tiene el efecto de aumentar su valor en $1\,C$ cada vez (el valor de incremento está definido en la macro `INCREMENTO_CARGA`). Sobre el campo provoca un alejamiento del $10\,\%$ del plano virtual.

rueda del mouse abajo Sobre una carga tiene el efecto de decrementar su valor en $1\,C$ cada vez (el valor de decremento está definido en la macro `INCREMENTO_CARGA`). Sobre el campo provoca un acercamiento del $10\,\%$ del plano virtual.

Shift + flecha arriba Aumenta el número de flechas de campo en la pantalla sin afectar la escala del modelo ni el tamaño de la ventana.

Shift + flecha abajo Disminuye el número de flechas de campo en la pantalla sin afectar la escala del modelo ni el tamaño de la ventana.

Listing 4.7: Programa principal del simulador de campo eléctrico

```
1  /* c04/campos vectoriales/main.c
2   * */
3  #include <SDL/SDL.h>
4  #include <SDL/SDL_gfxPrimitives.h>
5  #include <stdio.h>
6  #include <math.h>
7  #include "colores.h"
8  #include "campo.h"
9  #include "escala.h"
10
11 #define MAX_CARGAS 20
12 #define CARGA_INICIAL 1.0
13 #define INCREMENTO_CARGA 1.0
14 #define MIN_CANT_CUADROS 5
15
16 struct configuracion{
17     int numCuadrosH;
18     int numCuadrosV;
19
20     Uint32 colorFondo;
21     Uint32 colorFlecha;
22     Uint32 colorPuntaFlecha;
23     Uint32 colorCargaPositiva;
24     Uint32 colorCargaNegativa;
25
```

```
26      CargaElectrica cargas[MAX_CARGAS];
27      int cantidadActualCargas;
28
29      Escala e;
30      double radioCarga; //en el marco virtual
31  } conf;
32
33  int profundidad_color;
34  const SDL_VideoInfo *info;
35
36  void configurar(void){
37      conf.numCuadrosH = conf.numCuadrosV = 30;
38      conf.colorFondo = BLANCO;
39      conf.colorFlecha = VERDE;
40      conf.colorPuntaFlecha = ROJO|AZUL;
41      conf.colorCargaPositiva = ROJO;
42      conf.colorCargaNegativa = NEGRO;
43      conf.e.AnchoR = 600;
44      conf.e.AltoR = 600;
45      conf.e.XminV = conf.e.YminV = -10.0;
46      conf.e.XmaxV = conf.e.YmaxV = 10.0;
47      conf.radioCarga = 1.0;
48      conf.cantidadActualCargas = 0;
49  }
50
51  //Dibuja los campos en el buffer, no en la pantalla directamente
52  void actualizar(SDL_Surface *pantalla){
53      int i,j;
54      Vector campo;
55      Vector punto;
56
57      borrarPantalla(pantalla, conf.colorFondo);
58
59      //dibujar las cargas
60      for(i=0;i<conf.cantidadActualCargas; i++){
61          filledEllipseColor(pantalla,
62          tR_Vx(conf.cargas[i].posicion.x,&conf.e),
63          tR_Vy(conf.cargas[i].posicion.y,&conf.e),
64          abs(mR_Vx(conf.radioCarga,&conf.e)),
65          abs(mR_Vy(conf.radioCarga,&conf.e)),
66          (conf.cargas[i].valor>0.0)? conf.colorCargaPositiva: conf.
                colorCargaNegativa);
67      }
68
69      //dibujar el campo
70      if(conf.cantidadActualCargas>0)
71      for(i=0; i<conf.numCuadrosV; i++){
72
73          //el centro de cada cuadrito
74          punto.y = tV_Ry((conf.e.AltoR/conf.numCuadrosV)*(2*i+1)/2, &conf.
```

```
            e);

        for(j=0;j<conf.numCuadrosH;j++){
            punto.x = tV_Rx((conf.e.AnchoR/conf.numCuadrosH)*(2*j+1)/2, &
                conf.e);

            campoElectricoCargas(conf.cargas, conf.cantidadActualCargas,
                &punto, &campo);

            //dibujar las lineas
            aalineColor(pantalla,
            tR_Vx(punto.x, &conf.e),
            tR_Vy(punto.y, &conf.e),
            //**********
            tR_Vx(punto.x, &conf.e) + (int)((campo.x/magnitudVector(&
                campo))*(conf.e.AnchoR/conf.numCuadrosH -1)),
            tR_Vy(punto.y, &conf.e) - (int)((campo.y/magnitudVector(&
                campo))*(conf.e.AltoR /conf.numCuadrosV -1)),
            conf.colorFlecha);

            //dibujar puntas
            filledCircleColor(pantalla,
            tR_Vx(punto.x, &conf.e) + (int)((campo.x/magnitudVector(&
                campo))*(conf.e.AnchoR/conf.numCuadrosH -1)),
            tR_Vy(punto.y, &conf.e) - (int)((campo.y/magnitudVector(&
                campo))*(conf.e.AltoR /conf.numCuadrosV -1)),
            1,
            conf.colorPuntaFlecha);
        }
    }
}

void configurarVideo(SDL_Surface **pantalla){
    if (SDL_Init(SDL_INIT_VIDEO) < 0){
        printf("Error al iniciar SDL: %s\n", SDL_GetError());
        exit(1);
    }
    atexit(SDL_Quit);

    //Este if es importante para poder usar SDL_gfx
    info = SDL_GetVideoInfo();
    if ( info->vfmt->BitsPerPixel > 8 ) {
        profundidad_color = info->vfmt->BitsPerPixel;
        //printf("%d\n", profundidad_color);
    } else {
        profundidad_color = 16;
    }

    *pantalla = SDL_SetVideoMode(conf.e.AnchoR, conf.e.AltoR,
        profundidad_color,  SDL_SWSURFACE|SDL_RESIZABLE);
```

```
117     if (*pantalla == NULL){
118         printf("Error al inicializar el modo de video: '%s'\n",
                SDL_GetError());
119         exit(2);
120     }
121 
122     SDL_WM_SetCaption("Simulador de campos eléctricos", NULL);
123 }
124 
125 /*
126  * Dado un arreglo de cargas eléctricas,
127  * y una coordenada en el marco Real,
128  * retorna el índice del arreglo que se corresponde
129  * con la carga seleccionada.
130  * Si no coincide con ninguna, retorna -1
131  *
132  * */
133 int seleccionarCarga(CargaElectrica *cargas, int cantidad, int xR, int yR
        ){
134     int i;
135     int x1,x2,y1,y2;
136     for (i=0; i<cantidad; i++){
137         x1 = tR_Vx(cargas[i].posicion.x,&conf.e) - mR_Vx(conf.radioCarga
                ,&conf.e);
138         x2 = tR_Vx(cargas[i].posicion.x,&conf.e) + mR_Vx(conf.radioCarga
                ,&conf.e);
139         y1 = tR_Vy(cargas[i].posicion.y,&conf.e) + mR_Vy(conf.radioCarga
                ,&conf.e);
140         y2 = tR_Vy(cargas[i].posicion.y,&conf.e) - mR_Vy(conf.radioCarga
                ,&conf.e);
141         if(x1 <= xR && xR <= x2 && y1 <= yR && yR <= y2)
142             return i;
143     }
144     return -1;
145 }
146 
147 int main(int argc, char *argv[]){
148     SDL_Surface *pantalla = NULL;
149     SDL_Event evento;
150 
151     int corriendo = 1;
152     int seleccion = -1;
153 
154     configurar();
155     configurarVideo(&pantalla);
156 
157     //hacer primer dibujo
158     stringColor(pantalla,0,conf.e.AltoR/2,"Haga clic derecho para agregar
            una carga eléctrica", conf.colorFlecha);
159 
```

```
160     //volcar el buffer en la pantalla
161     SDL_Flip (pantalla);
162 
163     while(corriendo) {
164     while(SDL_PollEvent(&evento)) {
165         switch(evento.type){
166         case SDL_VIDEORESIZE: {
167             conf.e.AnchoR = evento.resize.w;
168             conf.e.AltoR  = evento.resize.h;
169 
170             //redimensionar la pantalla, no hace falta liberar la
                    anterior.
171             pantalla =
172                     SDL_SetVideoMode(conf.e.AnchoR, conf.e.AltoR,
                            profundidad_color,
173             SDL_SWSURFACE|SDL_RESIZABLE);
174             if(pantalla == NULL){
175                 printf("Error al redimensionar la pantalla: '%s'\n",
                        SDL_GetError());
176                 exit(1);
177             }
178 
179             //La pantalla nueva aparece en blanco
180             //o mejor dicho, en negro,
181             //por lo que hay que volver a dibujar
182             stringColor(pantalla, 0,conf.e.AltoR/2, "Haga clic derecho
                    para agregar una carga eléctrica", conf.colorFlecha);
183             actualizar(pantalla);
184 
185             //vuelca el buffer en la pantalla:
186             SDL_Flip (pantalla);
187         }
188         break;
189 
190         case SDL_MOUSEBUTTONDOWN:{
191             //Seleccionar / evaluar campo
192             if(evento.button.button == SDL_BUTTON_LEFT){
193                 seleccion=seleccionarCarga(conf.cargas,conf.
                        cantidadActualCargas,evento.button.x,evento.button.y)
                        ;
194                 if(seleccion==-1){
195                     Vector punto;
196                     Vector campo;
197                     punto.x=tV_Rx(evento.button.x,&conf.e);
198                     punto.y=tV_Ry(evento.button.y,&conf.e);
199                     campoElectricoCargas(conf.cargas, conf.
                            cantidadActualCargas, &punto, &campo);
200                     printf("Campo electrico: (%f,%f), magnitud: %f\n",
                            campo.x, campo.y, magnitudVector(&campo));
201                 }
```

```
202             }
203
204             //Agregar una carga
205             else if(evento.button.button == SDL_BUTTON_RIGHT){
206                 if(conf.cantidadActualCargas < MAX_CARGAS){
207                     conf.cargas[conf.cantidadActualCargas].valor=
                            CARGA_INICIAL;
208                     conf.cargas[conf.cantidadActualCargas].posicion.x=
                            tV_Rx(evento.button.x,&conf.e);
209                     conf.cargas[conf.cantidadActualCargas].posicion.y=
                            tV_Ry(evento.button.y,&conf.e);
210                     conf.cantidadActualCargas++;
211                     actualizar(pantalla);
212                     SDL_Flip(pantalla);
213                 }
214                 else
215                     printf("Ya se alcanzó el máximo número de cargas
                            permitidas\n");
216             }
217
218             else if(evento.button.button == SDL_BUTTON_MIDDLE){
219                 int s;
220                 if((s=seleccionarCarga(conf.cargas,conf.
                        cantidadActualCargas,evento.button.x,evento.button.y)
                        )>-1){
221                     conf.cargas[s].valor = -conf.cargas[s].valor;
222                     actualizar(pantalla);
223                     SDL_Flip(pantalla);
224                 }
225             }
226
227             else if(evento.button.button == SDL_BUTTON_WHEELUP){
228                 int s;
229                 if((s=seleccionarCarga(conf.cargas,conf.
                        cantidadActualCargas,evento.button.x,evento.button.y)
                        )>-1){
230                     conf.cargas[s].valor += INCREMENTO_CARGA;
231                     printf("Nuevo valor de la carga: %fC\n", conf.cargas[
                            s].valor);
232                 }
233                 else{//alejarse un 10%
234                     double incrementoAncho, incrementoAlto;
235                     incrementoAncho = abs(conf.e.XmaxV-conf.e.XminV)
                            *0.10;
236                     incrementoAlto = abs(conf.e.YmaxV-conf.e.YminV)*0.10;
237                     conf.e.XminV -= incrementoAncho/2;
238                     conf.e.XmaxV += incrementoAncho/2;
239                     conf.e.YminV -= incrementoAlto/2;
240                     conf.e.YmaxV += incrementoAlto/2;
241                 }
```

```
242                 actualizar(pantalla);
243                 SDL_Flip(pantalla);
244             }
245 
246             else if(evento.button.button == SDL_BUTTON_WHEELDOWN){
247                 int s;
248                 if((s=seleccionarCarga(conf.cargas,conf.
                        cantidadActualCargas,evento.button.x,evento.button.y)
                        )>-1){
249                     conf.cargas[s].valor -= INCREMENTO_CARGA;
250                     printf("Nuevo valor de la carga: %fC\n", conf.cargas[
                            s].valor);
251                 }
252                 else{//acercarse un 10%
253                     double decrementoAncho, decrementoAlto;
254                     decrementoAncho = abs(conf.e.XmaxV-conf.e.XminV)
                            *0.10;
255                     decrementoAlto = abs(conf.e.YmaxV-conf.e.YminV)*0.10;
256                     conf.e.XminV += decrementoAncho/2;
257                     conf.e.XmaxV -= decrementoAncho/2;
258                     conf.e.YminV += decrementoAlto/2;
259                     conf.e.YmaxV -= decrementoAlto/2;
260                 }
261                 actualizar(pantalla);
262                 SDL_Flip(pantalla);
263             }
264 
265         }
266         break;
267 
268         //asegurar que se deselecciona la carga
269         case SDL_MOUSEBUTTONUP:{
270             seleccion = -1;
271         }
272         break;
273 
274         //desplazar la carga seleccionada
275         case SDL_MOUSEMOTION:{
276             if(seleccion>-1){
277                 conf.cargas[seleccion].posicion.x=tV_Rx(evento.button.x,&
                        conf.e);
278                 conf.cargas[seleccion].posicion.y=tV_Ry(evento.button.y,&
                        conf.e);
279                 actualizar(pantalla);
280                 SDL_Flip(pantalla);
281             }
282         }
283         break;
284 
285         case SDL_KEYDOWN:{
```

```
286             if(evento.key.type == SDL_KEYDOWN && evento.key.keysym.mod &
                    KMOD_SHIFT)
287             switch(evento.key.keysym.sym){
288                 case SDLK_UP: {
289                     conf.numCuadrosH++;
290                     conf.numCuadrosV++;
291                     actualizar(pantalla);
292                     SDL_Flip(pantalla);
293                 }
294                 break;
295                 case SDLK_DOWN: {
296                     if(conf.numCuadrosH>MIN_CANT_CUADROS)
297                         conf.numCuadrosH--;
298                     if(conf.numCuadrosV>MIN_CANT_CUADROS)
299                         conf.numCuadrosV--;
300                     actualizar(pantalla);
301                     SDL_Flip(pantalla);
302                 }
303                 break;
304             }
305         }
306         break;
307 
308         //cuando el usuario quiera cerrar la ventana, la aplicación debe
                terminar
309         case SDL_QUIT:
310             corriendo = 0;
311         break;
312         }//fin del switch
313     }//fin de while PollEvent
314     }//fin de while corriendo
315 
316     SDL_Quit();
317     return 0;
318 }
```

4.5.5. El `Makefile`

Finalmente, este es el archivo `Makefile` necesario para automatizar todo el proceso de compilación[2]:

Listing 4.8: Archivo Makefile para el simulador

```
1 # c04/campos vectoriales/Makefile
2 
3 LDFLAGS = $(shell sdl-config --cflags)
```

[2]Sólo se requiere ejecutar: `$ make`

```
4  LDLIBS  = $(shell sdl-config --libs)
5  GFX = -lSDL_gfx
6
7  ##RM       = /bin/rm -f
8
9  #Esto indica que los siguientes identificadores,
10 #no son archivos, sino comandos de make:
11 .PHONY: limpiar
12 .PHONY: limpiartodo
13 .PHONY: all
14 #se puede por ejemplo ejecutar en la consola
15 #lo siguiente:
16 #'make limpiartodo', etc.
17
18 #Nombre del programa ejecutable:
19 PROG = simulador
20
21 #Un '*.o' por cada '*.c'
22 OBJ = main.o escala.o colores.o campo.o
23
24 #Cuando se ejecuta 'make', se ejecuta esto:
25 all: limpiartodo $(PROG) limpiar
26
27 #Esto compila todo el código y lo enlaza:
28 $(PROG): $(OBJ)
29         $(RM) $(PROG)
30         gcc -o $(PROG) $(OBJ) $(LDLIBS) $(LDFLAGS) $(GFX)
31
32 #Borra todos los archivos intermedios y de copia de seguridad
33 limpiar:
34         $(RM) *~ $(OBJ) $(PRG)
35
36 #Borra todos los archivos intermedios, de copia de seguridad
37 # y el programa ejecutable, si es que existe
38 limpiartodo:
39         make limpiar
40         $(RM) $(PROG)
```

4.6. Ejercicios

1. Considerando la figura 4.6, construya las funciones
   `int tR_Vx(double xV);`
   `int tR_Vy(double yV);`
   `double tV_Rx(int xR);`
   `double tV_Ry(int yR);`
   para convertir puntos del marco de referencia virtual al real y viceversa. Asuma

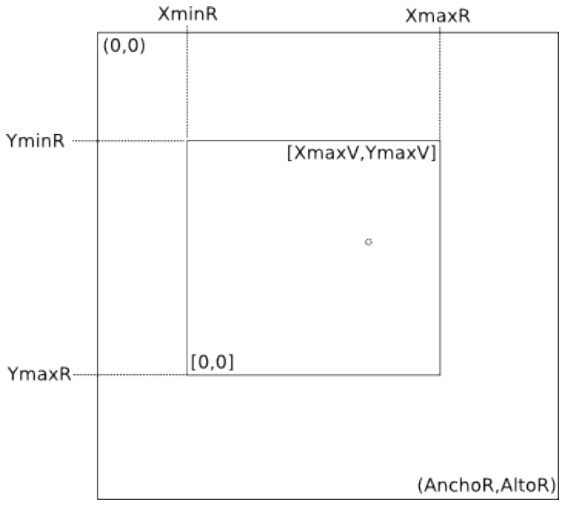


Figura 4.6: Diagrama para el ejercicio 1

que los símbolos mostrados en la figura son constantes.

2. Modifique el programa de la sección 4.5 en la página 115 para representar la diferencia de intensidad del campo en cada punto.
3. Agregar la funcionalidad de eliminar cargas al programa de la sección 4.5.
4. Modifique el programa de la sección 4.5 para que la cantidad de cargas puntuales aceptadas sea ilimitada.
5. Modifique el programa de la sección 4.5 para que el aumento/acercamiento tenga como centro la posición del cursor del ratón en ese momento en lugar del centro del espacio visible como sucede con la versión presentada.
6. Modifique el programa de la sección 4.5 para que sólo se pueda seleccionar una carga al hacer clic dentro de la superficie de la carga (ya que en la versión actual, la selección se logra con clic en el rectángulo circunscrito a la carga).
7. Realice todas las optimizaciones posibles al código de la sección 4.5, para lograr el mejor rendimiento[3].
8. Modifique el programa de la sección 4.5 para que el aumento de la cantidad de flechas de campo se logre con `Shift+Ruedas del ratón` sobre el campo eléctrico (en la versión actual es con `Shift+Flecha arriba/abajo`).
9. Modifique el programa de la sección 4.5 para permitir que las configuraciones iniciales (ver la estructura `configuracion` del programa principal) sean cargadas

[3] un buen lugar para empezar es en el archivo "`campo.c`", y el siguiente lugar, es el programa principal

desde un archivo y que las configuraciones al momento de cerrarse la aplicación se guarden en el mismo archivo.

5 Transformaciones Geométricas Bidimencionales

En este capítulo analizaremos los procedimientos estándares para realizar transformaciones geométricas bidimensionales sobre objetos gráficos inmersos en algún marco de referencia específico (real o virtual).

Todo el análisis girará en torno a la representación matricial de las coordenadas, por lo que se recomienda hacer un concienzudo repaso de operaciones con matrices, ya que el estudio de este capítulo depende transversalmente de dichas habilidades. Además, es preferible recordar algunos tópicos básicos de trigonometría analítica para poder tener una lectura más fluída.

5.1. Operaciones geométricas básicas

Existen básicamente tres operaciones o transformaciones geométricas básicas, a partir de las cuales se pueden realizar todas las demás. Estas transformaciones son la traslacicón, el escalamiento y la rotación, todas respecto del origen.

5.1.1. Traslación o Desplazamiento

La traslación o desplazamiento se refiere a mover un punto, un conjunto de puntos o un objeto compuesto, de su ubicación original hacia una nueva ubicación en su marco de referencia. La operación tiene un parámetro: el vector de desplazamiento.

Veamos un ejemplo gráfico:

Como podemos ver en la figura 5.1, el cuadro que representaremos de manera abstracta como P ha sido "trasladado" en un valor de $(3, -1)$ (este es el parámetro de la operación). En su nueva ubicación lo llamaremos P'. Tomemos su esquina inferior izquierda como base: Su posición original era $(2, 3)$ y la nueva es $(5, 2)$. El mismo desplazamiento ha afectado a todos los demás puntos del cuadro.

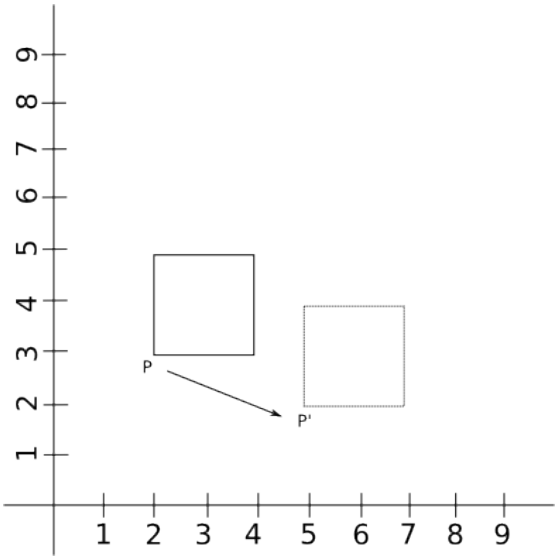


Figura 5.1: Ejemplo de traslación simple

5.1.2. Escalamiento

El escalamiento es la operación que nos permite agrandar o empequeñecer un conjunto de puntos o un objeto compuesto. La operación requiere de dos parámetros: el factor de escalamiento a aplicar en x y el factor de escalamiento a aplicar en y. La operación requiere además, de un punto de referencia, típicamente el origen del marco de referencia. En el caso de aplicar escalamiento básico a un punto, se produce el efecto de acercarlo o alejarlo del punto de referencia.

Veamos un ejemplo:

En la figura 5.2, se ilustra el escalamiento de P tomando como referencia al origen y con factores de escalamiento $\frac{1}{3}$ en x, y $\frac{1}{2}$ en y. El efecto, debido a que ambos factores son menores que 1, es que todos los puntos del cuadro P se han acercado al origen (pero más en x que en y).

Veamos otro ejemplo de escalamiento:

En la figura 5.3, se ilustra el escalamiento de P (en la misma posición que en el ejemplo anterior) tomando como referencia a su esquina inferior derecha y con factores de escalamiento $\frac{1}{2}$ en x, y $\frac{1}{2}$ en y.

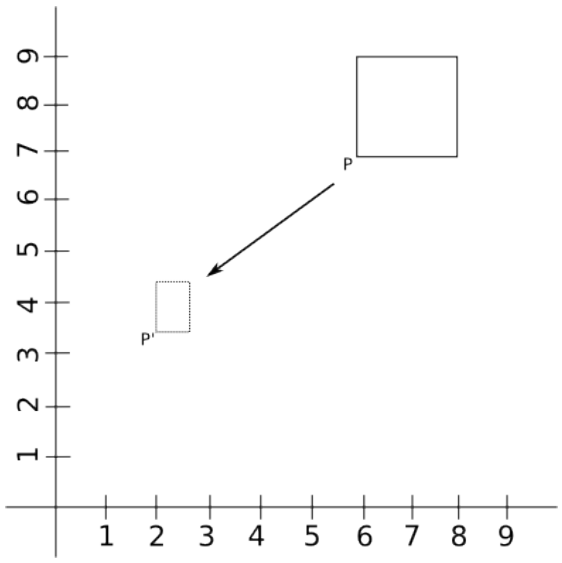


Figura 5.2: Ejemplo de escalamiento simple

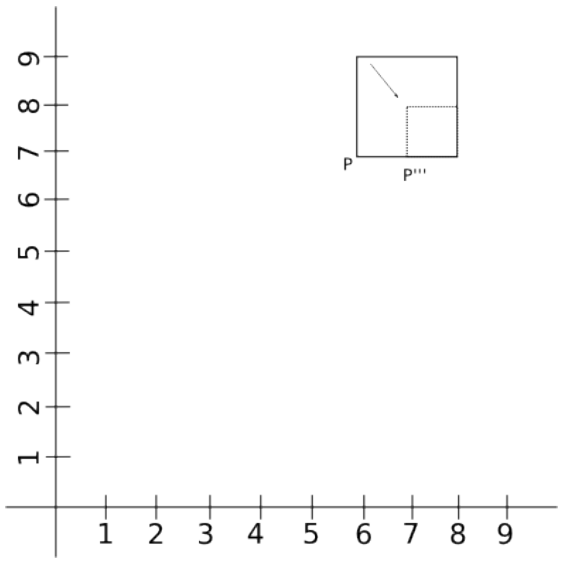


Figura 5.3: Ejemplo de escalamiento compuesto

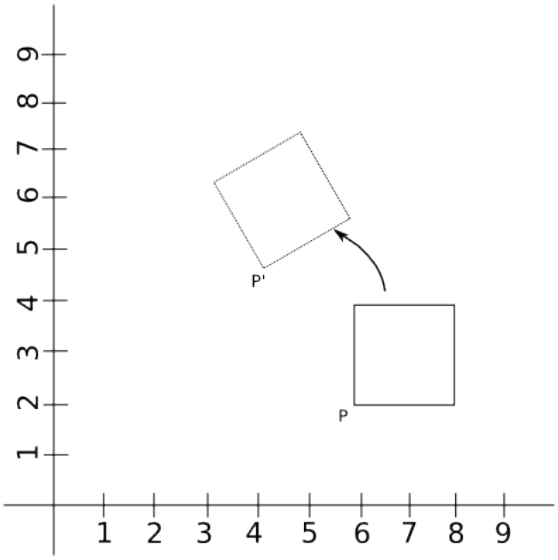


Figura 5.4: Ejemplo de rotación simple

5.1.3. Rotación

La rotación es la más compleja de las operaciones o transformaciones geométricas básicas. Consiste en girar un punto, un conjunto de puntos o un cuerpo compuesto, al rededor de un punto de referencia (el cetnro de rotación), típicamente el origen del marco de referencia.

Veamos un ejemplo:

En la figura 5.4, se ilustra la rotación de P, que está en las coordenadas $(6, 2)$. La rotación es de $\frac{\pi}{6}$ en sentido contrario al de las agujas del reloj a partir del eje x^+. El punto de referencia es el origen del marco de referencia. Las coordenadas del punto inferior izquierdo de P' son $(4{,}1962, 4{,}7321)$[1].

Un ejemplo más:

En la figura 5.5, se ilustra la rotación de P, que está en las mismas coordenadas del ejemplo anterior. La rotación es de $\frac{\pi}{4}$ en el sentido de las agujas del reloj a partir del eje x^-. El punto de referencia es esquina superior derecha de P. Las coordenadas del punto inferior izquierdo de P''' son $(5{,}1716, 4)$.

[1] Invitamos amablemente al lector a comprobarlo con compás y transportador

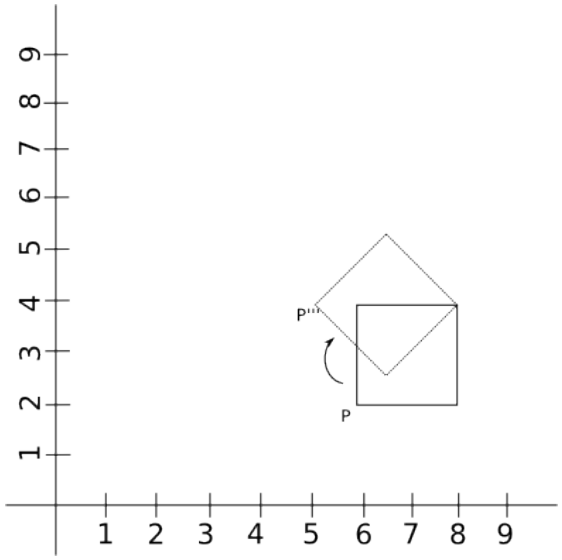


Figura 5.5: Ejemplo de rotación compuesta

5.2. Representación matricial

Aunque existen diversas maneras de representar las coordendas de los objetos gráficos y de representar las transformaciones geométricas que operan sobre ellos, vamos a elegir aquí la más estándar y flexible.

Los puntos se representan como vectores columna de tamaño 3×1:

$$P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.1}$$

La *operación de traslación bidimensional* se representa como una matriz de 3×3:

$$T(d_x, d_y) = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

La *operación de escalamiento bidimensional* (con el origen como punto de referencia) se representa similar:

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

La *operación de rotación bidimensional* (respecto del origen, en el sentido opuesto al de las manecillas del reloj) se representa así:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

De este modo, para efectuar transformaciones geométricas básicas, esas "funciones" en forma de matrices, deben premultiplicarse por los puntos que se pretende transformar, así:

$P' = T(d_x, d_y) \cdot P$ significa que P' es P con un desplazamiento de (d_x, d_y).

$P' = S(s_x, s_y) \cdot P$ significa que P' es P con factores de escalamiento s_x en x y s_y en y tomando el origen como referencia.

$P' = R(\theta) \cdot P$ significa que P' es P con una rotación de θ radianes en sentido opuesto al movimiento de las manecillas del reloj, a partir del eje x^+, tomando como eje de rotación al origen del marco de referencia.

-

5.3. Composición de tranformaciones geométricas

Con lo planteado hasta ahora, surge la necesidad de representar varias transformaciones geométricas encadenadas, para producir resultados "compuestos", que no se pueden lograr con las operaciones básicas representadas por las ecuaciones 5.2, 5.3 y 5.4. Ejemplos de ello son las transformaciones realizadas en las figuras 5.3 en la página 135 y 5.5 en la página anterior.

Analicemos el caso de la figura 5.3: Para lograr tal transformación, deberíamos primero trasladar P de tal manera que su esquina inferior derecha quede en el origen. Luego, hay que hacer el escalamiento con ambos factores a $\frac{1}{2}$. Luego, habría que re-desplazar el cuadro transformado, a su posición final, con la esquina inferior derecha a donde estaba antes.

Expresado de otra manera:

Sea $Q = (8, 7)$ la esquina inferior derecha del cuadro original P. Si aplicamos la transformación $T(-8, -7)$ a todos los puntos de P, lo habremos movido de tal manera que Q' de P' ha quedado en el origen. Posteriormente, aplicamos a P' el escalamiento $S\left(\frac{1}{2}, \frac{1}{2}\right)$ para obtener P'', que es un cuadro de ancho y alto igual a 1, con su esquina inferior derecha también en el origen. Después, aplicamos la traslación $T(8, 7)$ a P'', con lo que obtenemos P'''.

La matriz de transformación aplicada en conjunto es $T(8,7) \cdot S\left(\frac{1}{2}, \frac{1}{2}\right) \cdot T(-8,-7)$. Podemos aplicarla a los puntos de P para obtener los puntos de P''' directamente, sin hacer pasos intermedios.

Sigamos la secuencia de transformación para el punto Q como ejemplo:

$$\begin{aligned} Q' &= T(-8,-7) \cdot Q \\ Q'' &= S\left(\frac{1}{2}, \frac{1}{2}\right) \cdot Q' = S\left(\frac{1}{2}, \frac{1}{2}\right) \cdot T(-8,-7) \cdot Q \\ Q''' &= T(8,7) \cdot Q'' = T(8,7) \cdot S\left(\frac{1}{2}, \frac{1}{2}\right) \cdot T(-8,-7) \cdot Q \end{aligned}$$

5.4. Atención a la eficiencia

Sucede que todas las composiciones de matrices de transformación resultan en matrices de la siguiente forma:

$$M = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Por lo que su producto con un punto $P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ requiere 9 multiplicaciones y 6 sumas. Sin embargo, al realizar las operaciones, se puede apreciar que el resultado es previsible. El resultado siempre será que $x' = xr_{11} + yr_{12} + t_x$ y $y' = xr_{21} + yr_{22} + t_y$. Lo cual muestra que únicamente son necesarias 4 multiplicaciones y 4 sumas.

La importancia de esta consideración es porque las transformaciones de este tipo son extensivamente usadas en la graficación tridimensional. Y un gasto innecesario de tiempo de procesador para realizar operaciones de las que ya conocemos el resultado, haría disminuir el rendimiento sensiblemente.

5.5. Versión matricial del cambio de coordenadas

A continuación analizaremos otra perspectiva del cambio de coordenadas entre dos marcos de referencia. Esta vez usando transformaciones matriciales.

Reconsideremos la figura 4.2 en la página 111 y la ecuación 4.1 en la página 112. De ellas podemos concluir que lo primero que hacemos para hacer un cambio de coordenadas, es un escalamiento, y luego hacemos un desplazamiento.

Intuitivamente concluimos que la matriz de transformación es:

$$M_{R\leftarrow V} = T\left(\overrightarrow{V}_x^{(R)}, \overrightarrow{V}_y^{(R)}\right) \cdot S\left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}}, \frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\right) \tag{5.5}$$

Así que la transformación que habíamos realizado como $\overrightarrow{p}^{(R)} = \overrightarrow{T}_{R\leftarrow V}\left(\overrightarrow{p}^{(V)}\right)$, también se puede realizar como:

$$\begin{aligned}
\begin{bmatrix} \overrightarrow{p}_x^{(R)} \\ \overrightarrow{p}_y^{(R)} \\ 1 \end{bmatrix} &= M_{R\leftarrow V} \cdot \begin{bmatrix} \overrightarrow{p}_x^{(V)} \\ \overrightarrow{p}_y^{(V)} \\ 1 \end{bmatrix} \\
\begin{bmatrix} \overrightarrow{p}_x^{(R)} \\ \overrightarrow{p}_y^{(R)} \\ 1 \end{bmatrix} &= T\left(\overrightarrow{V}_x^{(R)}, \overrightarrow{V}_y^{(R)}\right) \cdot S\left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}}, \frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\right) \cdot \begin{bmatrix} \overrightarrow{p}_x^{(V)} \\ \overrightarrow{p}_y^{(V)} \\ 1 \end{bmatrix}
\end{aligned} \tag{5.6}$$

Hay un punto muy importante que recalcar en este punto, y es que así como hemos definido $M_{R\leftarrow V}$, también podríamos definir un encadenamiento de transformaciones sucesivas para pasar de un marco de referencia a otro, pasando por intermedio de otros más así: $M_{D\leftarrow A} = M_{D\leftarrow C} \cdot M_{C\leftarrow B} \cdot M_{B\leftarrow A}$, como en la figura 5.6. Esto proporciona la capacidad extra, de agregar rotaciones a dichos cambios de coordenadas o cualquier otra transformación aplicable.

En la figura 5.7 podemos ver un caso hipotético en el que el marco de referencia virtual, está inclinado θ radianes en el sentido opuesto a la del movimiento de las agujas del reloj. Significa que la versión matricial de la función de transformación del marco de referencia V al R es:

$$\begin{aligned}
\begin{bmatrix} \overrightarrow{p}_x^{(R)} \\ \overrightarrow{p}_y^{(R)} \\ 1 \end{bmatrix} &= M_{R\leftarrow V} \cdot \begin{bmatrix} \overrightarrow{p}_x^{(V)} \\ \overrightarrow{p}_y^{(V)} \\ 1 \end{bmatrix} \\
\begin{bmatrix} \overrightarrow{p}_x^{(R)} \\ \overrightarrow{p}_y^{(R)} \\ 1 \end{bmatrix} &= T\left(\overrightarrow{V}_x^{(R)}, \overrightarrow{V}_y^{(R)}\right) \cdot S\left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}}, \frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\right) \cdot R(-\theta) \cdot \begin{bmatrix} \overrightarrow{p}_x^{(V)} \\ \overrightarrow{p}_y^{(V)} \\ 1 \end{bmatrix}
\end{aligned}$$

Desde la perspectiva vectorial, habría sido difícil ofrecer una solución sencilla.

5.6. Reversión de transformaciones geométricas

En frecuentes ocaciones no sólo necesitaremos efectuar transformaciones geométricas a un conjunto de puntos, sino que también necesitaremos "*destransformarlos*". Para hacer

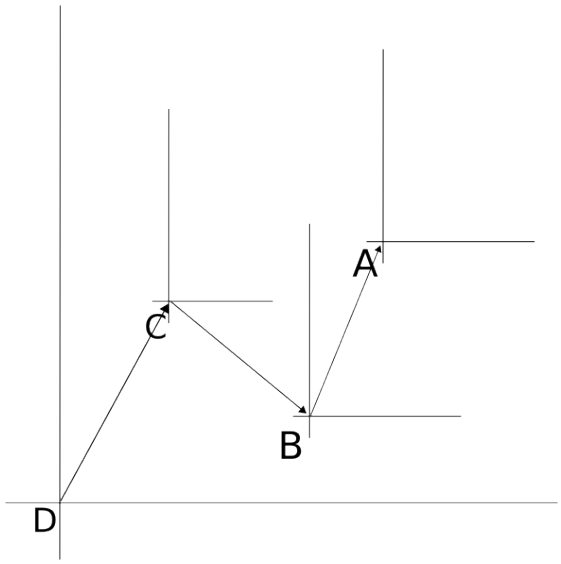


Figura 5.6: Cambio de coordenadas a través de múltiples marcos de referencia

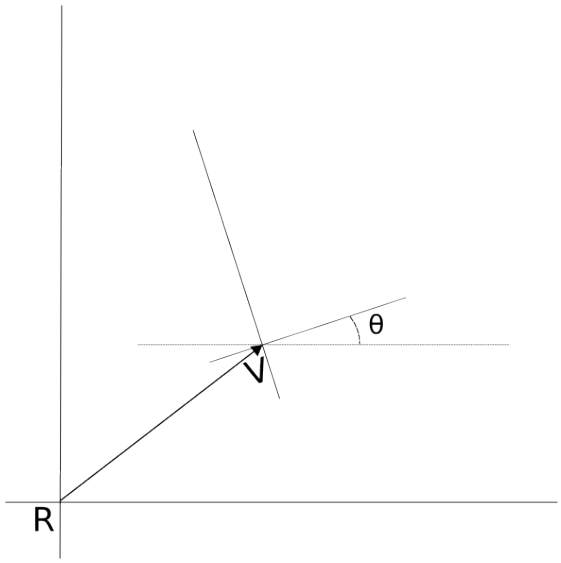


Figura 5.7: Cambio de coordenadas con rotación

esto, recordemos las siguientes propiedades matriciales:

$$\begin{aligned} P' &= M \cdot P \\ M^{-1} \cdot P' &= M^{-1} \cdot M \cdot P \\ &= \left(M^{-1} \cdot M\right) \cdot P \\ &= I \cdot P \\ M^{-1} \cdot P' &= P \end{aligned}$$

y

$$(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$$

Esto significa que si tenemos $P' = T\left(\overrightarrow{V}_x^{(R)}, \overrightarrow{V}_y^{(R)}\right) \cdot S\left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}}, \frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\right) \cdot P$, entonces:

$$\begin{aligned} \left(T\left(\overrightarrow{V}_x^{(R)}, \overrightarrow{V}_y^{(R)}\right) \cdot S\left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}}, \frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\right)\right)^{-1} \cdot P' &= I_3 \cdot P \\ \left(S\left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}}, \frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\right)\right)^{-1} \cdot \left(T\left(\overrightarrow{V}_x^{(R)}, \overrightarrow{V}_y^{(R)}\right)\right)^{-1} \cdot P' &= P \end{aligned}$$

y entonces basta con encontrar las respectivas matrices inversas y obtendremos la matriz de retro-transformación para P' (o sea, para regresar de P' a P).

Afortunadamente no es necesario calcular ninguna matriz inversa (ese algoritmo no es precisamente rápido[2]), ya que suceden los siguientes fenómenos (cuyas demostraciones se dejan como ejercicios):

$$T\left(d_x, d_y\right)^{-1} = T\left(-d_x, -d_y\right) \tag{5.7}$$

$$S\left(s_x, s_y\right)^{-1} = S\left(\frac{1}{s_x}, \frac{1}{s_y}\right) \tag{5.8}$$

$$R(\theta)^{-1} = R(-\theta) \tag{5.9}$$

Esto significa, según la explicación en la que estabamos, que:

[2] aunque no completamente inviable

$$\left(S\left(\frac{\Delta_x^{(R)}}{\Delta_x^{(V)}},\frac{\Delta_y^{(R)}}{\Delta_y^{(V)}}\right)\right)^{-1}\cdot\left(T\left(\overrightarrow{V}_x^{(R)},\overrightarrow{V}_y^{(R)}\right)\right)^{-1}\cdot P' = P$$

$$S\left(\frac{\Delta_x^{(V)}}{\Delta_x^{(R)}},\frac{\Delta_y^{(V)}}{\Delta_y^{(R)}}\right)\cdot T\left(-\overrightarrow{V}_x^{(R)},-\overrightarrow{V}_y^{(R)}\right)\cdot P' = P$$

Por lo que la idea general es que para invertir una serie de transformaciones geométricas, basta con aplicar las transformaciones inversas (siguiendo las ecuaciones 5.7, 5.8 y 5.9) en orden inveso[3].

5.7. Ejercicios

1. ¿Cuál es la matriz de transformación necesaria para lograr la transformación presentada en la figura 5.5?
2. Realizar la transformación de las otras aristas del cuadro analizado en la sección 5.3.
3. Considere la figura 5.8. Rote el triángulo -45º respecto de su esquina inferior derecha según la regla de la mano derecha, con la ayuda de una matriz de transformación bidimensional M_1.
   a) ¿Cuál es la definición de M_1?
   b) ¿Cuáles son las coordenadas de los puntos transformados?
4. Considere la figura 5.8 de nuevo. Rote el triángulo -30º respecto de su esquina inferior derecha según la regla de la mano derecha y después, duplique su tamaño manteniendo la coordenada de su punto inferior derecho. De nuevo haga esto con la ayuda de una matriz de transformación bidimensional M_2.
   a) ¿Cuál es la definición de M_2?
   b) ¿Cuáles son las coordenadas de los puntos transformados?
5. Considere la figura 5.8 otra vez. Triplique el tamaño del triángulo, manteniendo la coordenada del punto inferior izquierdo, con la ayuda de una matriz de transformación bidimensional M_3.
   a) ¿Cuál es la definición de M_3?
   b) ¿Cuáles son las coordenadas de los puntos transformados?

[3] es cierto, parece un juego de palabras, pero así es

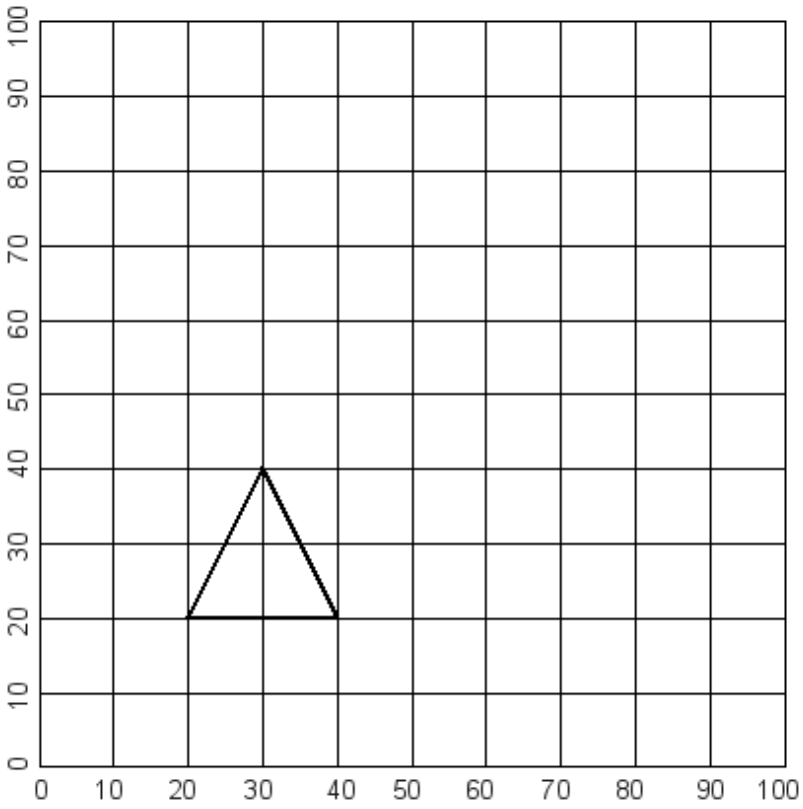


Figura 5.8: Ejercicio de transformación bidimensional

6. Demuestre la ecuación 5.7 en la página 142.
7. Demuestre la ecuación 5.8.
8. Demuestre la ecuación 5.9.

6 Transformaciones Geométricas Tridimencionales

En este capítulo se presentan los tópicos básicos y las herramientas matemáticas básicas para realizar transformaciones geométricas tridimensionales en marcos de referencia tridimensionales.

6.1. Sistemas de referencia

La mayoría de los curso de álgebra y cálculo vectorial utilizan el sistema de referencia de la mano derecha, por lo que la mayoría de los análisis posteriores a eso, lo utilizan también. No seremos la excepción, aunque conviene atender que algunas bibliotecas gráficas utilizan por defecto el sistema de referencia de la mano izquierda por diversas razones.

No ahondaremos más en el asunto ya que se asume que el lector ha aprobado algún curso de álgebra y/o cálculo vectorial. Baste mencionar que la matriz de transformación para pasar del sistema de mano derecha al sistema de mano izquierda (y viceversa) es $S(1, 1, -1)$.

6.2. Representación Matricial de Transformaciones Geométricas

Los puntos en tres dimensiones se representan como vectores columna de tamaño 4×1:

$$P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6.1}$$

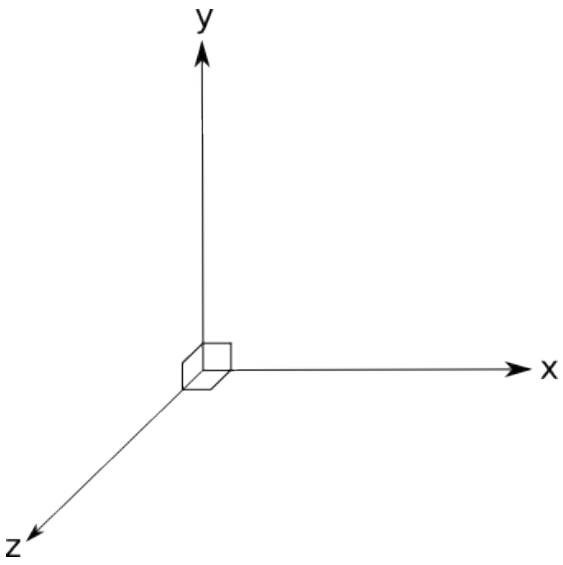


Figura 6.1: Sistema de referencia de mano derecha

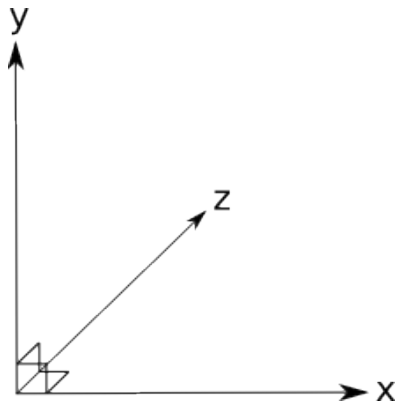


Figura 6.2: Sistema de referencia de mano izquierda

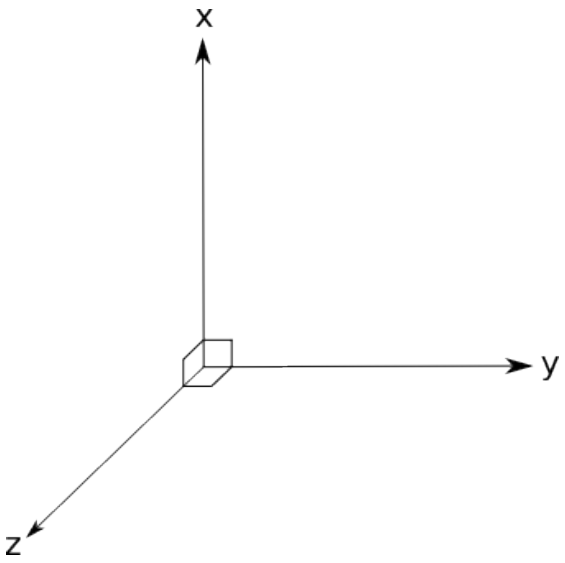


Figura 6.3: Otra manera de ver el sistema de referencia de mano izquierda

La *operación de traslación tridimensional* se representa como una matriz de 4×4:

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.2}$$

La *operación de escalamiento tridimensional* (con el origen como punto de referencia) se representa similar:

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

Las *operaciones de rotación* (respecto de cada eje principal, siguiendo la regla de la mano derecha) se representa así:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.5}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.6}$$

Nótese la similitud entre esta última ecuación $R_z(\theta)$ y la ecuación de $R(\theta)$ que es para el caso bidimensional (véase la ecuación 5.4 en la página 138).

6.3. Composición y reversión de Transformaciones Geométricas Tridimensionales

Las transformaciones geométricas tridimensionales, al igual que las bidimensionales, también pueden *"componerse"* e *"invertirse"*. La lógica matemática es idéntica, por lo que nos limitaremos en este capítulo a presentar las inversas de dichas transformaciones:

$$T(d_x, d_y, d_z)^{-1} = T(-d_x, -d_y, -d_z) \tag{6.7}$$

$$S(s_x, s_y, s_z)^{-1} = S\left(\frac{1}{s_x}, \frac{1}{s_y}, \frac{1}{s_z}\right) \tag{6.8}$$

$$R_x(\theta)^{-1} = R_x(-\theta) \tag{6.9}$$

$$R_y(\theta)^{-1} = R_y(-\theta) \tag{6.10}$$

$$R_z(\theta)^{-1} = R_y(-\theta) \tag{6.11}$$

A manera de práctica, proponemos el uso de la aplicación `transformaciones3D.jar`[1] que se encuentra en el material adjunto a esta obra. Esta aplicación permite, de una manera muy simple, practicar las transformaciones geométricas tridimensionales y ver sus efectos sobre dos objetos con volumen y un objeto plano que pueden traslaparse.

Para ejecutar la aplicación, se requiere de la máquina virtual de java (se recomienda la versión 1.6), procediendo así:

[1] Consúltese el capítulo A en la página 291 para más detalles sobre la compilación y ejecución de aplicaciones J2SE.

```
$ java -jar transformaciones3D.jar
```

Con esto se iniciará una ventana gráfica y se inicia un intérprete de comandos en la consola donde se ejecutó el comando anterior.

La ventana gráfica muestra al inicio un cubo y un rectángulo tridimensionales en el centro de un marco de referencia tridimensional de mano derecha que se puede rotar con arrastre del ratón. Los ejes x^+, y^+ y z^+ son lineas de colores *rojo*, *verde* y *azul* respectivamente (es un truco mnemotécnico que relaciona el modelo RGB[2] con los ejes x-y-z y la regla de la mano derecha). Con la rueda del ratón se puede acercar y alejar la imagen.

A través de la ventana gráfica no se pueden alterar los objetos mencionados, para eso es el intérprete de comandos en la consola (si escribe `ayuda` y presiona `<Enter>`, se mostrará una breve descripción de los comandos disponibles):

`:ayuda` muestra una breve reseña de los comandos permitidos.

`:reiniciar` reinicia la aplicación a su estado inicial por defecto.

`:%<comentario>` agrega un comentario sin efecto alguno sobre la aplicación.

`:<objeto> mostrar` muestra el objeto en cuestión (si ya está visible, no tiene efecto).

`:<objeto> ocultar` oculta el objeto en cuestión (pero todavía permite su transformación).

`:<objeto> t <dx> <dy> <dz>` provoca un desplazamiento $T(d_x, d_y, d_z)$ en el objeto especificado.

`:<objeto> e <sx> <sy> <sz>` provoca un escalamiento $S(s_x, s_y, s_z)$ en el objeto especificado.

`:<objeto> eu <s>` provoca un escalamiento uniforme $S(s, s, s)$, en el objeto especificado.

`:<objeto> {rx|ry|rz} <theta>` provoca una rotación (en grados sexagesimales) en el eje especificado para el objeto especificado.

`:salir` termina la aplicación.

Los objetos válidos para esta versión de la aplicación son `cubo`, `plano`, `ojo` y `todos`. El cubo tiene una arista de largo 4, el plano es de 4×3 y el ojo[3] es una pirámide de $4 \times 3 \times 1$. Los ejes tienen una longitud de 10. Si el comando introducido no es reconocido, el programa muestra lo que el usuario escribió y la ayuda. El ojo está por defecto oculto[4].

[2] en la subsección 2.5.1 en la página 76

[3] veremos su propósito en el siguiente capítulo

[4] su utilidad no es relevante en este momento, pero se le puede considerar, por el momento, como una pirámide rectangular.

Podríamos realizar cualquier combinación de transformaciones, pero vamos a seguir en este texto, las siguientes (obviamente recomendamos que después de cada instrucción introducida en la consola se examinen las transformaciones correspondientes en la ventana gráfica):

```
:plano ocultar
:cubo t 2 0 0
:cubo eu 2
:cubo t 0 -4 -4
:cubo t -8 0 0
:cubo t 4 4 4
:cubo e 1 1 .5
:cubo e 1 .5 1
:cubo rx 30
:cubo rx -30
:cubo ry 60
:cubo ry 30
:cubo rz 45
:cubo e 1 1 .5
:cubo rz 45
:cubo ocultar
:%%%%%%%%%%%
:ojo mostrar
:ojo ry 30
:ojo ry -30
:% préstese atención a estas últimas transformaciones:
:ojo t 5 0 0
:ojo rz 30
:ojo rx 60
:ojo t 0 2 0
:ojo t 0 -2 0
:ojo rx -60
:ojo rz -30
:ojo t -5 0 0
```

Note que las últimas cuatro transformaciones invirtieron exactamente las anteriores cuatro.

6.4. Ejercicios

1. Demuestre las ecuaciones 6.7 en la página 148, 6.8 y 6.9.

7 Vista Tridimensional (de 3D a 2D)[1]

El acto que comunmente llamaríamos «convertir algo de tres dimensiones a dos dimensiones» es, formalmente hablando, una *proyección* de un conjunto de objetos tridimensionales, sobre una superficie plana.

Existe una gran cantidad de tipos de proyecciones, cada una con sus aplicaciones y particularidades matemáticas; algunas aplicadas al arte digital, a la arquitectura, al diseño de piezas de maquinaria, etc. Abordarlas todas, excede el propósito de este libro, por lo que en el presente capítulo describiremos brevemente las dos principales: *las proyecciones ortogonales* y *las proyecciones de perspectiva*.

Recomendamos repasar en este momento a más tardar, los sistemas de coordenadas rectangular, cilíndrico y esférico y las transformaciones entre ellos.

7.1. Proyección Ortogonal

La proyección ortogonal consiste básicamente en proyectar las figuras sobre un "plano de proyección" en dirección ortogonal al plano.

Veamos algunos ejemplos en la figura 7.1 en la página siguiente.

En todas las imágenes podemos ver los ejes principales x, y y z representados por líneas de colores rojo, verde y azul respectivamente[2].

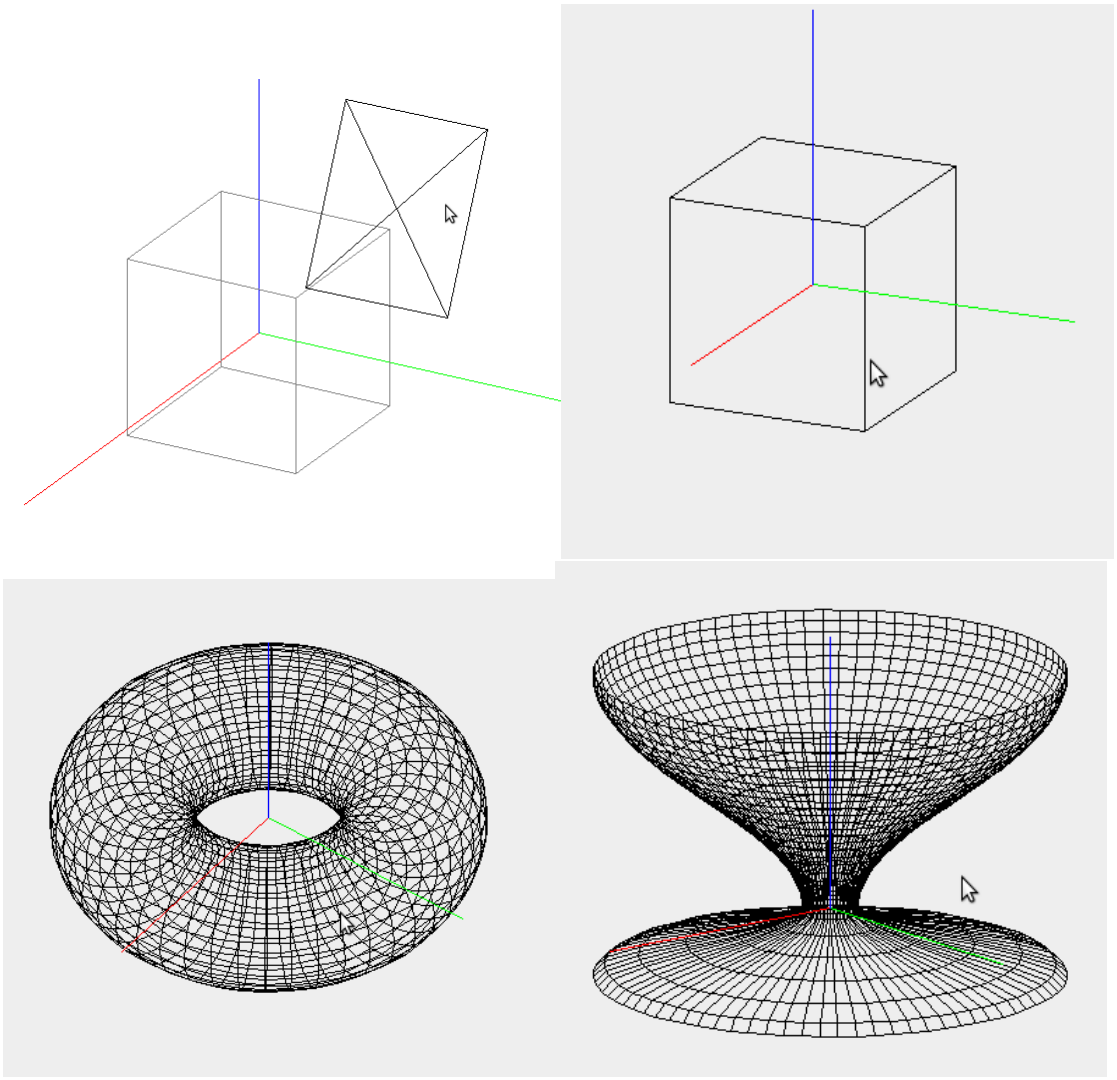
Los detalles geométricos formales no son de nuestro interés en este momento ya que el propósito de este texto no es adentrarnos en el estudio imagenológico de las proyecciones, sino entender su efecto visual desde la experiencia del usuario de las aplicaciones gráficas tridimensionales.

La principal característica a resaltar, es que **en este tipo de proyección *NO* existe la reproducción de la sensación de "profundidad" o "lejanía"**.

[1] Como comentario personal del autor, es curioso que modelemos realidades tridimensionales en modelos de almacenamiento lineales (la memoria de la computadora es unidimensional) que a su vez, representamos en pantallas planas bidimensionales.

[2] igual que se explicó en la sección 6.3 en la página 148

Figura 7.1: Ejemplos de proyección ortogonal



Esto podemos verlo en el primer recuadro de la figura recién referenciada. La figura es una instantanea de la aplicación utilizada en el capítulo anterior habiendo ejecutado los siguientes comandos:

Listing 7.1: Alejamiento del plano

```
:plano ry 90
:plano t -50 0 0
:plano rz 25
:plano ry -25
```

de lo que se puede deducir que el "plano" está bastante lejos.

7.2. Proyección en Perspectiva

En la proyección en perspectiva se proyectan las figuras sobre el mismo "plano de proyección" que en la proyección ortogonal, pero cada punto en una dirección oblicua al plano, alineada con un foco de proyección (ver la figura 7.3 en la página 158).

Veamos algunos ejemplos en la figura 7.2 en la página siguiente.

En todas las imágenes podemos ver los ejes principales x-y-z de la misma manera que en la figura 7.1 en la página anterior. De nuevo, los detalles geométricos formales no son de nuestro interés en este momento.

La principal característica a resaltar, es que **en este tipo de proyección *SÍ* existe la reproducción de la sensación de "profundidad" o "lejanía"**, tal como podemos ver en la figura.

Los recuadros se corresponden con los de la figura 7.1, con la diferencia que allá aparecen en proyección ortogonal y aquí en perspectiva, con la evidente sensación de que hay partes de los cuerpos que están más cerca de nosotros que otras.

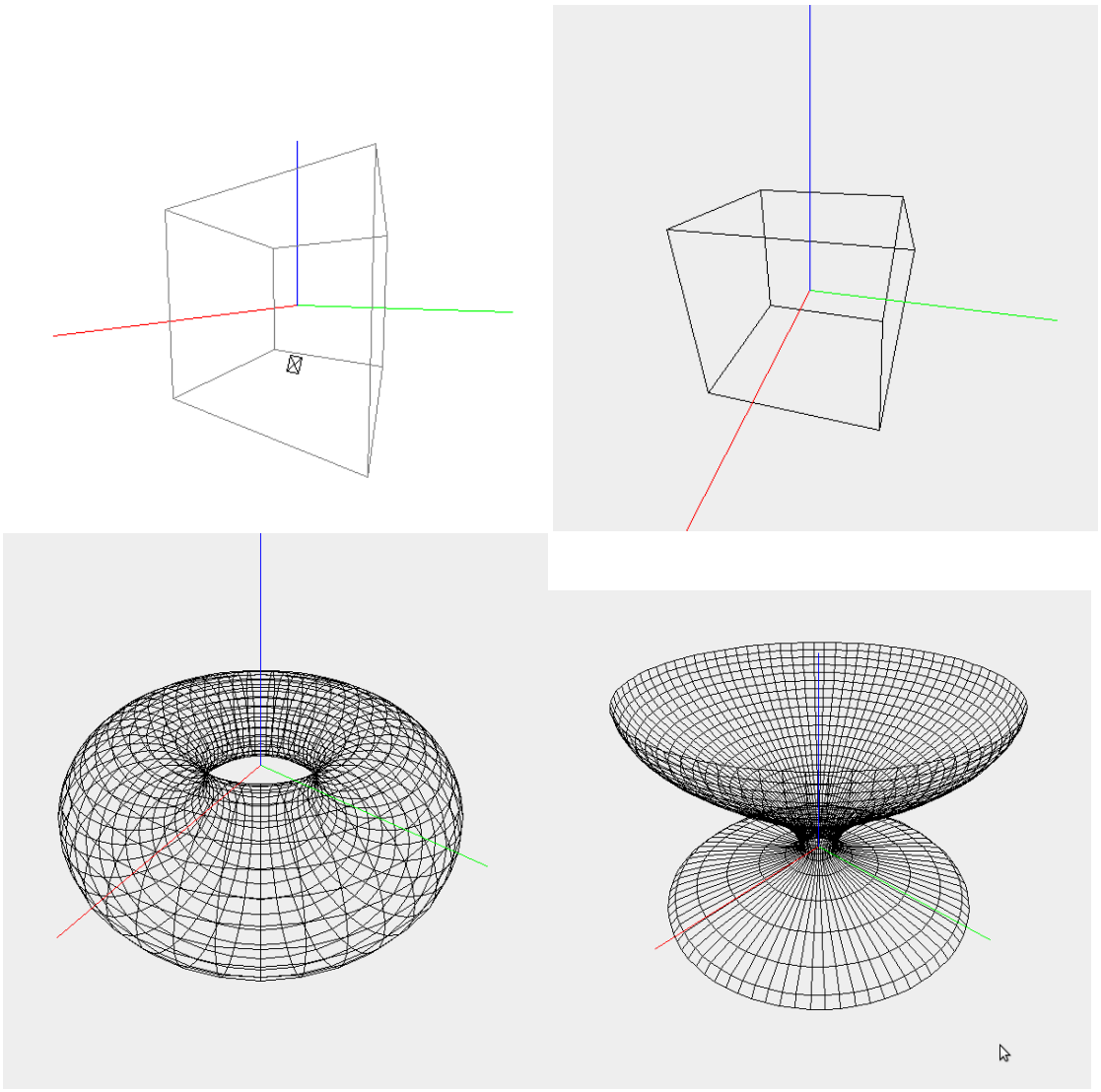
7.3. Portal de Visión

Ahora bien, la pregunta importante es: **¿Cómo producir proyecciones de los tipos presentados en las secciones anteriores?** Y a estas alturas del libro, una parte de la respuesta es obvia: *Segurísimo que se logran con transformaciones geométricas tridimensionales*[3].

La pregunta consecuente es: **¿Con qué secuencia de transformaciones?** En la presente sección trataremos de responder claramente a esta pregunta.

[3] por que si no, no habría tenido sentido estudiar todo eso... ¿verdad...?

Figura 7.2: Ejemplos de proyección en perspectiva



Antes de avanzar, es necesario definir el concepto de Portal de Visión: El **Portal de Visión** de una aplicación gráfica tridimensional es una ventana, un rectángulo flotante (desplazable o no) en el espacio tridimensional de la aplicación. Este representa la pantalla bidimensional a través de la cual, el usuario (o los usuarios) perciben esta realidad virtual tridimensional.

El *portal de visión*, es nuestro *plano de proyección* y su descripción resulta fundamental para construir aplicaciones gráficas tridimensionales.

Habiendo aclarado eso, empecemos con las proyecciones ortogonales, que son más fáciles de comprender.

A continuación haremos el siguiente ejercicio, apoyandonos en nuestra aplicación `transformaciones3D.jar`: Vamos a considerar el "ojo" como el portal de visión de una aplicación gráfica tridimensional imaginaria y vamos a realizar las transformaciones geométricas necesarias para transformar todo el escenario de tal manera que el portal de visión quede sobre el plano xy. Entonces, al ver exactamente desde atrás del portal de visión[4] es como si lo viéramos ya proyectado sobre la pantalla imaginaria (ya que habremos obviado la coordenada z de todos los puntos de las figuras).

Considere la siguiente secuencia de comandos para nuestra aplicación `transformaciones3D.jar` para realizar lo anteriormente descrito:

Listing 7.2: Proyección ortogonal

```
:% Ocultamos momentaneamente al cubo y al plano
:% para concentrarnos en el portal de visión.
:cubo ocultar
:plano ocultar
:% Coloquemos el "ojo" en posición:
:ojo mostrar
:ojo t 0 0 5
:ojo rz 90
:ojo ry 60
:ojo rz 30
:% Ahora el ojo está en posición de
:% representar el portal de visión.
:% Volvamos a visibilizar el cubo y el plano:
:cubo mostrar
:plano mostrar
:% Agreguemos un detalle:
:plano t 0 0 2
:% Veamos la imágen desde el portal:
:ojo ocultar
```

[4] asumimos que la parte de atrás es la que tiene la punta

```
:% Procedemos a hacer la alineación con 'xy'
:ojo mostrar
:todos rz -30
:todos ry -60
:todos rz -90
:todos t 0 0 -5
:% Ahora podríamos ocultar el ojo
:% para verlo mejor, :)
:ojo ocultar
```

Ahora, si efectivamente imaginamos que el "ojo" es una pantalla de computadora[5], y nos ubicamos exactamente detrás de él, veremos la imagen de la pantalla imaginaria.

Ahora revisemos qué fue exactamente lo que hicimos:

Primero colocamos el portal de visión en una posición conocida, con su centro en la coordenada $\left(r:5, \theta:\frac{\pi}{6}, \phi:\frac{\pi}{3}\right)$ (en las líneas 7 a 10 del código 7.2). Llamémosla simplemente (r, θ, ϕ). Luego aplicamos la siguiente transformación (en las líneas 22 a 25) al objeto a proyectar (en este caso el cubo y el plano):

$$T(0,0,-r) \cdot R_z\left(-\frac{\pi}{2}\right) \cdot R_y(-\phi) \cdot R_z(-\theta) \tag{7.1}$$

Luego podríamos alinear la esquina inferior izquierda del portal de visión con el origen, agregando $T(2,1{,}5,0)$. Además, podríamos agregar escalamiento, etc....

Lo importante de este ejercicio, es entender cómo lograr proyecciones ortogonales a través de transformaciones geométricas tridimensionales.

7.4. Implementación de proyecciones ortogonales

En este momento ya casi disponemos de los recursos teóricos para proceder a diseñar una estrategia de sistematización de proyecciones ortogonales. Un detalle importante que falta es considerar que la ecuación 7.1 no es la versión más apropiada de las alternativas disponibles. A continuación presentamos una secuencia de transformaciones que es más apropiada:

$$M_{2D \leftarrow 3D} = T(0,0,r) \cdot R_z\left(-\frac{\pi}{2}\right) \cdot R_y(\pi - \phi) \cdot R_z(-\theta) \tag{7.2}$$

Esta transformación es muy similar a la anteriormente presentada, pero presenta una importante mejora sin alterar significativamente el tiempo de ejecusión[6]: Coloca los

[5] por eso su tamaño es 4×3

[6] Tiene una inversión de signo menos y una suma más

objetos que están delante del portal de visión en el eje z^+ y con el eje y^+ hacia abajo (lo que es usual en graficación en computadoras).

Veamos el mismo caso de proyección del código 7.2, pero con la transformación 7.2:

Listing 7.3: Proyección ortogonal preferible

```
:% Ocultamos momentaneamente al cubo y al plano
:cubo ocultar
:plano ocultar
:% Coloquemos el "ojo" en posición:
:ojo mostrar
:ojo t 0 0 5
:ojo rz 90
:ojo ry 60
:ojo rz 30
:% Ahora el ojo está en posición.
:% Volvamos a visibilizar el cubo y el plano:
:cubo mostrar
:plano mostrar
:plano t 0 0 2
:% Procedemos a hacer la alineación con 'xy',
:% pero de otra manera:
:todos rz -30
:todos ry 120
:todos rz -90
:todos t 0 0 5
```

Luego de hacer esta transformación, convendrá hacer una transformación extra hacia el marco de referencia real, con la transformación 5.5 en la página 140 o alguna equivalente que se ajuste a las necesidades, así[7]:

$$\begin{aligned} P' &= M_{R\leftarrow V} \cdot M_{2D\leftarrow 3D} \cdot P \\ P' &= S(s_x, s_y, s_z) \cdot T\left(\frac{AnchoR}{2}, \frac{AltoR}{2}, 0\right) \cdot M_{2D\leftarrow 3D} \cdot P \end{aligned} \tag{7.3}$$

Es en este punto donde la aplicación `transformaciones3D.jar` implementa su funcionalidad de "acercamiento" y "alejamiento". Sugerimos abrir el código de dicha aplicación para ver todo este proceso en acción. En particular, el código que realiza la definición de la matriz de transformación está en la función `colocarPuntodeVision` del archivo `PortaldeVision.java`.

[7] recordando que el eje y^+ ya está hacia abajo

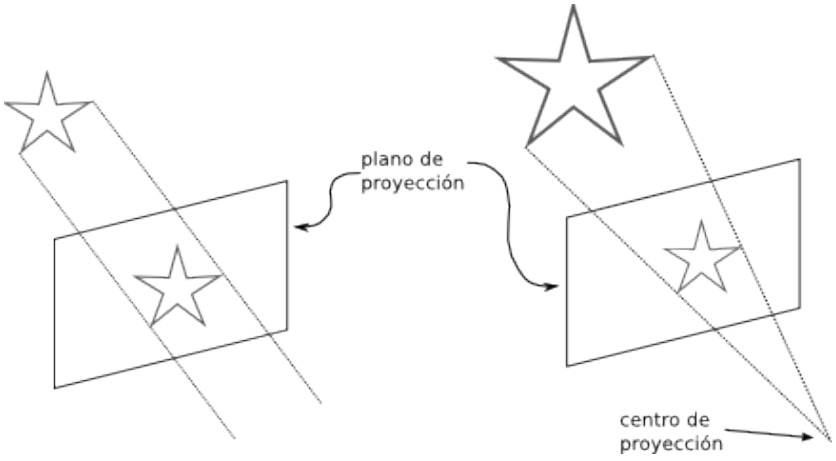


Figura 7.3: Proyección ortogonal y en perspectiva

7.5. Implementación de proyecciones en perspectiva

Antes que nada, veamos ahora algunos detalles geométricos anteriormente postergados de los tipos de proyecciones estudiados.

La figura 7.3 nos muestra la diferencia fundamental entre ambos tipos de proyecciones: En las proyecciones ortogonales las figuras se aplastan contra el plano en dirección siempre perpendicular a este (lo que se logra simplemente ignorando las componentes en z después de haber hecho la transformación 7.3), mientras que en las proyecciones en perspectiva, las figuras se proyectan en dirección de un "foco de proyección" o "centro de proyección".

Típicamente el centro de proyección está detrás del plano de proyección (como si fuera el ojo físico del observador, en el cual también hay un foco de proyección) y los objetos a proyectar están delante del plano (si los objetos están detrás del foco, el efecto visual es bastante desagradable y confuso).

Veamos a continuación cómo implementar este fenómeno óptico:

Utilicemos la figura 7.4 como referencia. En ella se presenta un punto (x, y, z) que es proyectado sobre el plano xy, que además es el plano de proyección[8]. El centro de proyección está a una distancia d del plano. El valor en x del punto "proyectado" debe ser x_{per} sobre el plano. Para encontrar ese valor, hacemos un análisis de triángulos semejantes:

[8] esto por simplicidad

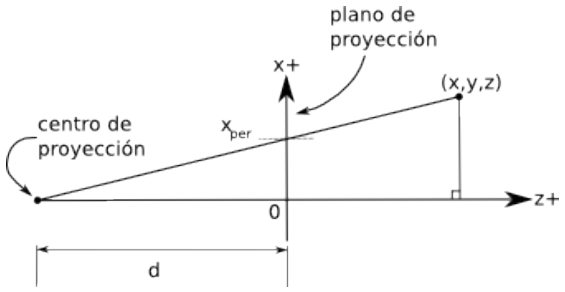


Figura 7.4: Deducción de proyección en perspectiva

$$\begin{aligned} \frac{x_{per}}{d} &= \frac{x}{d+z} \\ x_{per} &= \frac{d \cdot x}{d+z} = \frac{d}{d+z} \cdot x \end{aligned} \tag{7.4}$$

de la misma manera (nótese que $d \neq 0$):

$$\begin{aligned} \frac{y_{per}}{d} &= \frac{y}{d+z} \\ y_{per} &= \frac{d \cdot x}{d+z} = \frac{d}{d+z} \cdot y \end{aligned} \tag{7.5}$$

y

$$z_{per} = 0 \tag{7.6}$$

Lo que significa que podría aplicarse la transformación $S\left(\frac{d}{d+z}, \frac{d}{d+z}, 0\right)$ para realizar la proyección, posterior a la transformación 7.2 (recuerde las condiciones descritas al inicio de esta sección). O sea: $P'_{per} = S\left(\frac{d}{d+z}, \frac{d}{d+z}, 0\right) \cdot M_{2D \leftarrow 3D} \cdot P$

El problema de este último escalamiento es que depende del valor de z de cada punto a proyectar, por lo que no es viable para un procesamiento automatizado; es decir, no tiene sentido hacer una multiplicación matricial por cada punto a proyectar. Sería demasiado costoso, por lo que vamos a emplear otra estrategia:

1. Construir la matriz $M_{2D \leftarrow 3D}$ (la matriz de transformación ortogonal)
2. Hacer $P' = M_{2D \leftarrow 3D} \cdot P$ para todos los puntos P a proyectar (esto equivale a hacer proyección ortogonal)

3. Para cada punto P a proyectar, hacer

$$\begin{aligned} P''_x &= \frac{d}{d+P'_z} \cdot P'_x \\ P''_y &= \frac{d}{d+P'_z} \cdot P'_y \\ P''_z &= 0 \end{aligned}$$

4. Y después, aplicar el escalamiento y traslación final para cada punto: $P_{per} = M_{R \leftarrow V} \cdot P''$

Recomendamos, a manera de práctica, el uso de la aplicación `perspectiva3D.jar`[9] que se encuentra en el material adjunto a esta obra. Esta aplicación permite hacer las mismas cosas que su hermana `transformaciones3d.jar`, pero la proyección es en perspectiva y no ortogonal.

Para ejecutar la aplicación, también escrita en lenguaje java:

```
$ java -jar perspectiva3D.jar
```

Con esto se iniciará una ventana gráfica, idéntica a la de la otra aplicación, y se inicia un intérprete de comandos con las mismas alternativas.

Recomendamos que se haga una comparación entre el archivo `PortaldeVision.java` de la aplicación `perspectiva3D.jar` y el de `transformaciones3d.jar`, en ellos podemos ver una implementación de estas técnicas de proyección:

Listing 7.4: Fragmento del archivo PortaldeVision.java de transformaciones3d.jar

```
38          /*
39          * Recibe el centro del portal de visión en coordenadas
40          * esféricas (d, theta, phi)
41          */
42          public void colocarPuntodeVision(Objeto3DSimple objetos[], float
                theta, float phi, float distancia){
43              this.theta = theta;
44              this.phi = phi;
45              this.distancia=distancia;
46
47              matriz.identidad();
48
49              matriz.rotarZ(- theta);
50              matriz.rotarY(180 - phi);
51              matriz.rotarZ(-90);
52              matriz.trasladar(0, 0, distancia);
53
```

[9] Consúltese el capítulo A en la página 291 para más detalles sobre la compilación y compilación de aplicaciones J2SE.

```
54              //la media del ancho y alto real entre
55              //la media del ancho y alto virtual, por
56              //un factor de aumento
57              matriz.escalar((float)(
58                  (tamanhoR.width + tamanhoR.height)/(AnchoV+AltoV)
59                      /
60                  magnitud_aumento
61                  ));
62
63              //traslamiento final:
64              matriz.trasladar(tamanhoR.width/2, tamanhoR.height/2, 0);
65
66              for(int i=0; i<objetos.length; i++)
67                  objetos[i].transformarProyeccion(matriz, this);
68          }
```

Listing 7.5: Fragmento del archivo PortaldeVision.java de perspectiva3D.jar

```
45          /*
46          * Recibe el punto de visión en coordenadas esféricas (distancia,
                theta, phi)
47          *
48          * Asumimos que dicha distancia representa tanto la
49          * distancia del origen virtual al centro del portal de visión
50          * como a la distancia del centro de proyección
51          * al centro del portal de visión.
52          * Esto es por simplicidad en este código
53          * pero no necesariamente debe ser así siempre.
54          */
55          public void colocarPuntodeVision(Objeto3DSimple objetos[], float
                theta, float phi, float distancia){
56              this.theta = theta;
57              this.phi = phi;
58              this.distancia = this.distanciaProyeccion = distancia;
59
60              matriz.identidad();
61
62              matriz.rotarZ(- theta);
63              matriz.rotarY(180 - phi);
64              matriz.rotarZ(-90);
65              matriz.trasladar(0, 0, distancia);
66
67              for(int i=0; i<objetos.length; i++)
68                  objetos[i].transformarProyeccion(matriz, this, true);
69
70              matriz.identidad();
71              //la media del ancho y alto real entre
72              //la media del ancho y alto virtual, por
73              //un factor de aumento
74              matriz.escalar((float)(
```

```
75                  (tamanhoR.width + tamanhoR.height)/(AnchoV+AltoV)
76                      // / 3f
77                      /distancia
78                  ));
79
80              //traslamiento final:
81              matriz.trasladar(tamanhoR.width/2, tamanhoR.height/2, 0);
82
83              for(int i=0; i<objetos.length; i++)
84                  objetos[i].transformarProyeccion2(matriz, this);
85          }
```

7.6. Ejercicios

1. En la aplicación `transformaciones3d.jar` el portal de visión (es decir, su centro) tiene una distancia fija de una unidad (ver archivo `PortaldeVision.java`), lo que quiere decir que eventualmente los objetos a proyectar se encuentran "detrás" del Portal de Visión de la aplicación. ¿Por qué la imágen no se distorciona como sucede con `perspectiva3D.jar`?
2. Explique qué pasa cuando el centro de proyección en perspectiva está muy lejano de los objetos proyectados.
3. Explique geométricamente (o matemáticamente) qué es lo que sucede cuando se realiza la proyección en perspectiva de un objeto que se encuentra detrás del centro de proyección.
4. Todo el análisis realizado hasta ahora implica que el portal de visión ve siempre directamente al origen del universo virtual. Reflexione cómo hacer para dirigirlo en direciones arbitrarias. Es decir, ¿cómo hacer para que el usuario pueda ver en otras direcciones?

8 Interacción

Este capítulo no pretende ser un tratado completo sobre técnicas de interacción humano-máquina, sino una breve reflexión sobre la importancia de hacer aplicaciones gráficas interactivas amigables, fáciles de usar e intuitivas.

La razón por la que es importante que nuestras aplicaciones sean amigables (y el grado de amigabilidad dependerá de los usuarios a los que esté orientada) es porque gran parte del éxito en la difusión de una aplicación, ya sea un juego de video, una aplicación educativa, de entretenimiento, un simulador de algún fenómeno físico, social, económico, etc. dependerá del hecho que el usuario logre sus objetivos y que no se frustre al intentar aprender a usarlas. El respeto de estándares oficiales o de facto, es muy importante en estos días. Ya no estamos en los días en los que se podían definir arbitrariamente paradigmas de interacción.

8.1. Posicionamiento

En una aplicación con varios objetos (en dos o tres dimensiones), el usuario debería poder ser capaz de abrise paso, de algún modo, para lograr posicionarse al alcance de algún objeto específico que quisiera ver o modificar.

Como programadores, a veces olvidamos las reacciones lógicas de una persona normal[1], como por ejemplo, utilizar las teclas cursoras, la de `Tabulación` y las teclas `Página Anterior` y `Página Siguiente` para avanzar en una u otra dirección del espacio de trabajo de las aplicaciones que hacemos. A veces se nos olvida programar esas teclas y otras combinaciones que son obvias para todo el mundo en estos días.

Por ello es importante presentarle la aplicación a usuarios no familiarizados con el desarrollo de la aplicación (de preferencia personas normales) para evaluar objetivamente si la aplicación no tiene un interacción excesivamente compleja o confusa antes de darla por terminada.

En el caso de aplicaciones bidimensionales, el problema de posicionamiento suele resolverse con barras de desplazamiento al rededor del espacio de trabajo. Y en las ocaciones

[1] entiéndase por persona normal a una persona no informática

en que se operan objetos que pueden traslaparse, es conveniente disponer funciones del tipo “Enviar al fondo”, “Enviar atrás”, “Traer adelante” y “Traer al frente”.

En el caso de aplicaciones tridimensionales, el problema de posicionamiento representa comunmente un problema de navegación espacial. En donde usualmente habrá que diseñar una estrategia de limitar el volumen visible. Usualmente no todo lo que esté en el universo virtual debería ser visible por el usuario en cada posición específica. Este fenómeno puede verse en diversos juegos de video 3D de carreras, en las que el paisaje va “apareciendo” a medida que el jugador se va desplazando por la pista. Esto por supuesto se debe a dos factores: El primero es que requeriría demasiado procesamiento para mostrar todos los detalles lejanos y el segundo es que sería demasiada información para que un usuario típico la procese.

Partes importantes del problema de navegación, son el problema del cambio de dirección y la magnitud del desplazamiento lineal. El primero se refiere a cómo cambiar la dirección de movimiento o la orientación en general del portal de visión, y el segundo se refiere a la cantidad de desplazamiento a realizar en una dirección específica o en general en cualquier dirección.

En el caso de nuestras aplicaciones de ejemplo de los últimos capítulos, el problema de la navegación espacial está limitado ya que el portal de visión siempre mira fijamente en dirección del origen de coordenadas del universo virtual. Sin embargo, sí existe el problema del cambio de dirección, resuelto con arrastre del ratón; y también existe el problema de la magnitud de desplazamiento lineal, resuelto con la rueda del ratón.

La aplicación `perspectiva3D.jar` tiene un evidente problema con respecto a la limitación del volumen visible, y es que no limita el volumen visible, por lo que proyecta incluso objetos detrás del centro de proyección (lo cual genera una distorción visual[2]).

8.2. Selección

8.2.1. Selección por puntero

Una vez que el usuario ha logrado posicionarse al alcance del objeto de su interés, debería poder indicar que quiere operar con él y no sólo verlo. En estos días, lo más usual es hacer clic sobre el objeto. Cualquier aplicación que no siga dicho paradigma, resulta “extraña” para un usuario únicamente acostumbrado a operaciones visuales.

En el caso particular de aplicaciones tridimensionales, la selección directa por ratón implica resolver, entre otros, los problemas siguientes: **(a)** Averiguar qué objeto está más cerca del usuario, es decir, del portal de visión. Esto es porque no debería tener

[2] Recomendamos al lector verificar esto e intentar no marearse ni asustarse en el proceso ;-)

sentido seleccionar objetos que están detrás del portal de visión, en el caso de proyección ortogonal, o detrás del centro de proyección, en el caso de proyección en perspectiva. Y **(b)** Averguar a qué punto del universo virtual, corresponde el punto sobre el que el usuario hizo `clic`. Esto último requiere al menos, de lograr la inversión de todas las transformaciones geométricas aplicadas.

Nótese que si aplicamos las reglas de la sección 7.5 para la proyección en perspectiva, la transformación en conjunto se vuelve irreversible (debido a la ecuación 7.6 en la página 159).

Una estrategia simple es obviar ejecutar la ecuación 7.6. Dado que las transformaciones que le siguen sólo afectan las otras dos dimensiones, no habrá efectos colaterales negativos, y por el contrario, la inversión será posible (también como un proceso de tres faces).

8.2.2. Selección por nominación

Otra manera de seleccionar objetos, es haciendo referencia a ellos por su nombre, seleccionándolos de una lista o escribiéndolos directamente en una línea de comandos.

Ese es el caso de varias aplicaciones especializadas en simulación que permiten asignarle nombre a los objetos gráficos (y tal vez también a los no gráficos), como nuestras aplicaciones de estudio de los últimos capítulos. En ellas, la operación de selección de objetos es a través de línea de comandos, amarrada a la operación de transformación[3].

8.3. Transformación

Las transformaciones sobre objetos bidimensionales y tridimensionales deberían incluir sólo las transformaciones que tengan sentido para el propósito de la aplicación. Estas podrían incluir las transformaciones básicas estudiadas en los capítulos 5 y 6 y/u otras transformaciones compuestas, relevantes, de nuevo, para la lógica de la aplicación.

En el caso bidimensional, las transformaciones se logran resolver generalmente con cambios de marco de referencia (ver capítulo 4), pero en el caso tridimensional, hay que utilizar estrategias cognitivamente eficaces para que el usuario no se sienta excesivamente perdido. A continuación mencionamos algunos ejemplos:

- Para rotar una escena 3D, podría utilizarse la estrategia de asociar el arrastre del ratón en x con un cambio del valor θ de la coordenada del centro del portal de visión, y asociar el arrastre del ratón en y con un cambio del valor ϕ de la

[3] La aplicación es así porque su propósito es practicar las tranformaciones geométricas en términos de multiplicaciones matriciales y no en términos gráficos.

coordenada del centro del portal. Esta técnica es usada en las aplicaciones usadas en los últimos capítulos.

- Para realizar acercamiento o alejamiento en una escena 3D, puede usarse la ya tradicional rueda del ratón, tal vez en combinación con la tecla `ctrl` o `shift`.
- Para el desplazamiento de objetos o de toda una escena 3D, o del origen del marco de referencia podría usarse arrastre del ratón combinado con alguna tecla como `ctrl` o `shift`.

8.4. Conclusión

Evidentemente, existen diversas (y tal vez ilimitadas) formas de interacción entre los usuarios y las aplicaciones gráficas interactivas, por lo que, lo que finalmente podemos decir a manera de recomendación general, es seguir el principio que la magnitud de las acciones del usuario sea proporcional a la magnitud de las transformaciones que estas acciones realizan. Obviamente esta proporción también debe ser *"ergonómica"*, apropiada para la naturaleza humana.

A manera de ejemplo de esto último, recomendamos examinar el archivo `Config3D.java` de las aplicaciones usadas en los últimos capítulos. En ellos aparecen varias constantes que controlan el funcionamiento de las aplicaciones. Estos valores fueron definidos debido a que proporcionan un comportamiento más "cómodo" que otros valores. En particular las constantes que controlan la relación entre pixeles arrastrados con el ratón y el giro del universo virtual son:

Listing 8.1: Fragmento de código que especifica constantes ergonómicas

```
41      //para poder rotar con el mouse:
42      int numPixelesparaMovimiento = 1;
43      float factorRapidezGiro = 4f;
44
45      //aumento del zoom por movimiento de la rueda del ratón:
46      static final float MAGNITUD_AUMENTO = 0.1f;
47      static final float MAGNITUD_AUMENTO_INICIAL = 5f;
```

En otras combinaciones, no resulta fácil manipular la aplicación.

9 Curvas Paramétricas

9.1. Representación algebráica de Curvas Paramétricas

Las curvas paramétricas suelen representarse de la siguiente manera:

$$\vec{r}(t) = f(t)\hat{i} + g(t)\hat{j}$$

$$\vec{r}(t) = x(t)\hat{i} + y(t)\hat{j}$$

$$\vec{r}(t) = (x(t), y(t))$$

ó de esta otra:

$$\begin{cases} x = f(t) \\ y = g(t) \end{cases} \quad , \quad \text{«}restricciones\text{»}$$

Obviamente, en cada caso se deberán especificar las funciones paramétricas (f y g, o x y y según la notación).

Veamos algunos ejemplos de curvas paramétricas:

Una circunferencia con radio R se representa así: $\vec{r}(\theta) = R\cos\theta\hat{i} + R\sin\theta\hat{j},\ 0 \leq \theta \leq 2\pi$

Un segmento de línea recta, desde el punto (x_1, y_1) al (x_2, y_2) así:

$$\vec{r}(t) = ((1-t)x_1 + tx_2)\,\hat{i} + ((1-t)y_1 + ty_2)\,\hat{j},\ 0 \leq t \leq 1$$

Una elipse con radios a en x y b en y:

$$\vec{r}(\theta) = a\cos\theta\hat{i} + b\sin\theta\hat{j},\ 0 \leq \theta \leq 2\pi$$

En la figura 9.1 en la página siguiente se presentan una circunferencia con $R = 5$, un segmento de línea recta con $(x_1, y_1) = (2, 1)$ y $(x_2, y_2) = (4, 5)$ y una elipse con $a = 2$ y $b = 3$.

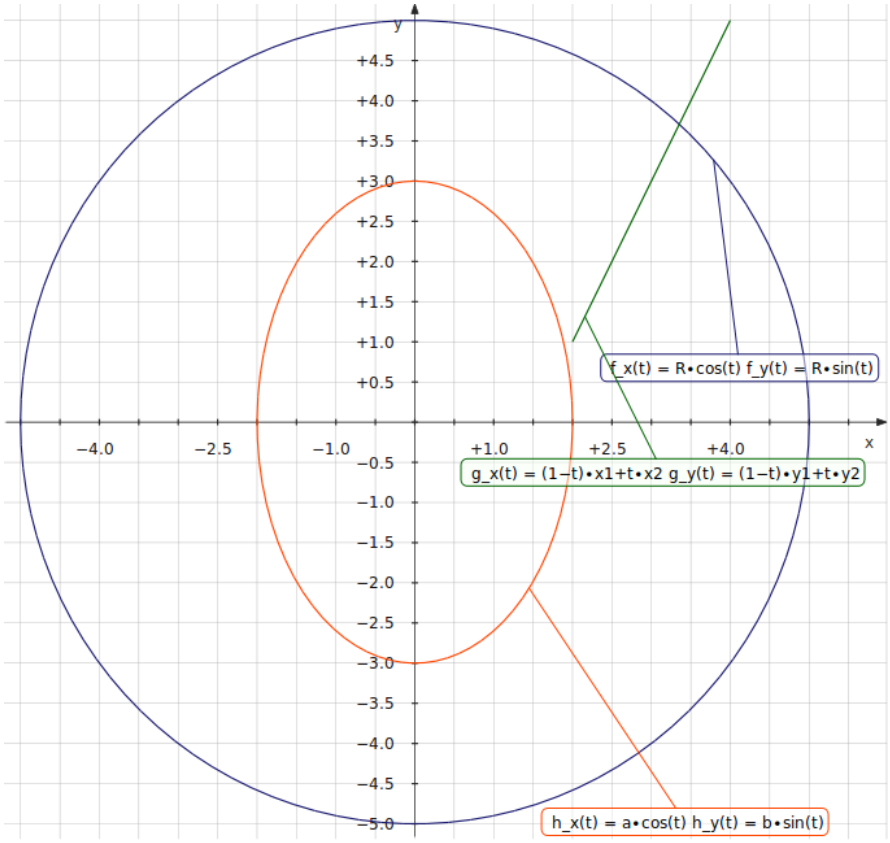


Figura 9.1: Ejemplo de curvas paramétricas planas

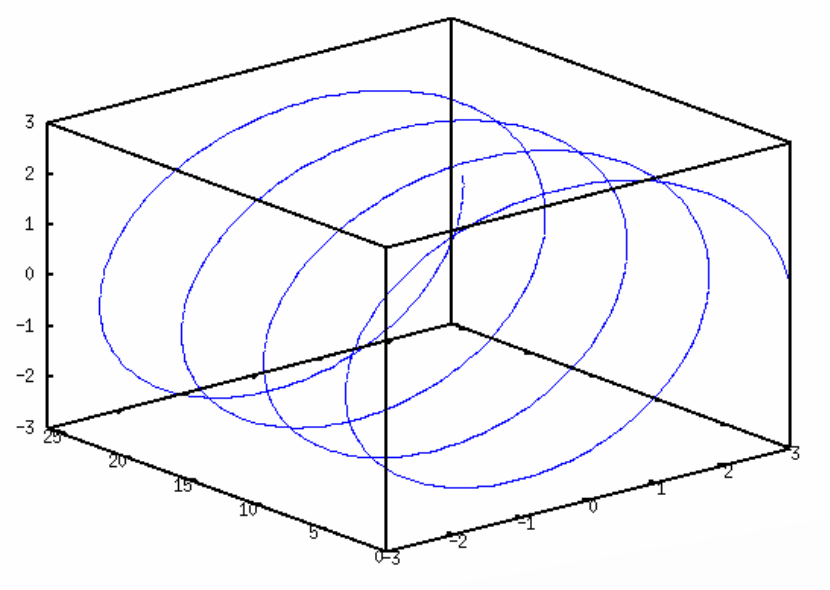


Figura 9.2: Resorte en alrededor del eje y, generado con Octave

Este es un resorte infinito con radio R alrededor del eje y:

$$\begin{cases} x = R\cos\theta \\ z = R\sin\theta \\ y = \theta \end{cases}$$

En la figura 9.2 se presenta un resorte con $R = 3$ y $0 \leq \theta \leq 8\pi$. Se realizó con el siguiente scritp de Octave:

```
theta = [0:pi/100:8*pi];
R = 3;
x = R*cos(theta);
z = R*sin(theta);
y = theta;
plot3(x,y,z)
```

9.2. Continuidad

9.2.1. Curvas suaves

Concepto de **curva continua** según [Henríquez 2001, p. 78]:

Se dice que una función es continua en un número c si

$$\lim_{x \to c} f(x) = f(c)$$

(Lo que a su vez se cumple cuando el límite por la izquierda y por la derecha son iguales).

Concepto de **curva suave**:

$f(x)$ es una curva suave en $[a, b]$ si $f(x)$ es continua en todos los puntos de $[a, b]$.

Curva paramétrica suave [Henríquez 2001, p. 229]:

$\vec{r}(t)$ es una curva suave en $[a, b]$ si $\vec{r'}(t)$ es continua en todos los puntos de $[a, b]$ y $\vec{r'} \neq \vec{0}$ en $]a, b[$.

9.2.2. Tipos de continuidad

A continuación presentamos un conjunto de conceptos importantes de referencia para conceptos posteriores, adaptados de [Foley et al., p. 374]:

Continuidad Geométrica

- Se dice que hay continuidad geométrica de grado 0, denotada por G^0, entre dos curvas suaves $\vec{S_1}$ y $\vec{S_2}$ en el punto p con $t = \tau$, si $\vec{S_1}(\tau) = \vec{S_2}(\tau)$. Es decir, si las curvas se unen en p.
- Se dice que hay continuidad geométrica de grado 1, denotada por G^1, entre dos curvas suaves $\vec{S_1}$ y $\vec{S_2}$ en el punto p con $t = \tau$, si hay continuidad G^0 (en ese mismo punto y con el mismo valor de parámetro) y $\vec{S_1'}(\tau) = k\vec{S_2'}(\tau)$, $k > 0$. Es decir, si las curvas se unen en p y los *vectores tangentes* tienen la misma dirección y sentido.

Nótese que G^1 no implica que la curva formada por la unión de $\vec{S_1}$ y $\vec{S_2}$ sea suave.

Continuidad Paramétrica

- Se dice que hay continuidad paramétrica de grado 1 (o de primer grado), denotada por C^1, entre dos curvas $\vec{S_1}$ y $\vec{S_2}$ en el punto p con $t = \tau$, si hay continuidad G^0 (en

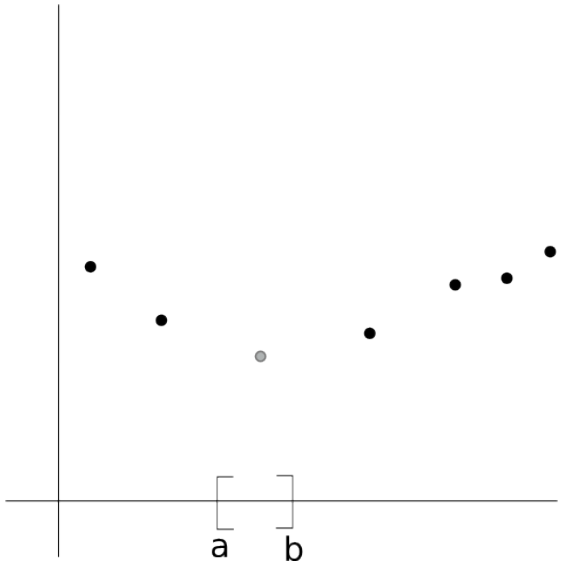


Figura 9.3: Interpolación

ese mismo punto y con el mismo valor de parámetro) y $\vec{S_1'}(\tau) = \vec{S_2'}(\tau)$. Es decir, si las curvas se unen en p y los vectores tangentes son iguales. También se puede definir que hay continuidad C^1 cuando hay continuidad G^1 con $k = 1$.

Nótese que $C^1 \Rightarrow G^1$, pero $G^1 \nRightarrow C^1$.

Nótese que C^1 implica que la curva formada por la unión de $\vec{S_1}$ y $\vec{S_2}$ es suave en todo su trayecto.

- Se dice que hay continuidad paramétrica de grado n, denotada por C^n, entre dos curvas $\vec{S_1}$ y $\vec{S_2}$ en el punto p con $t = \tau$, si hay continuidad C^{n-1}, en ese mismo punto y con el mismo valor de parámetro, y si $\overrightarrow{S_1^{(n)}}(\tau) = \overrightarrow{S_2^{(n)}}(\tau)$.

Nótese que si los vectores $\vec{S_1}$ y $\vec{S_2}$ representan (como función seccionada) la posición de un mismo objeto en función del tiempo, C^1 en su punto de unión, garantiza movimiento suave y velocidad continua, y C^2 allí mismo, garantiza velocidad suave y aceleración continua. Esto es muy deseable para representar desplazamientos sin irregularidades.

9.2.3. Interpolación

Interpolación según [RAE]: Calcular el valor *aproximado* de una magnitud en un intervalo, cuando se conocen algunos de los valores que toma a uno **y** otro lado de dicho intervalo.

Veamos la figura 9.3. En ella, el intervalo mencionado en la definición es $[a, b]$. Se conocen los puntos negros. Y el punto griz es un valor aproximado a partir de los puntos negros que sí son conocidos.

9.3. Trazadores interpolantes cúbicos - Curvas *Spline*

Se le conoce como *Trazadores Interpolantes Cúbicos* a un conjunto de segmentos de curva polinomial de tercer grado que unen una serie de puntos en el espacio —comunmente llamados *nodos*— entre sí; y que además, pueden ser utilizados para interpolar puntos desconocidos al interior de dicha serie de puntos.

Existen diversas maneras de construir trazadores interpolantes cúbicos, cada una con sus peculiaridades matemáticas y geométricas. En este libro nos concentraremos en un tipo particular: **Las curvas Spline con frontera natural**.

9.3.1. Curvas Spline y B-Spline

Con frecuencia, aparece en la literatura el término B-spline. Estas y las curvas Spline no son las mismas. Ambas consisten de segmentos de curvas polinomiales cúbicas, pero la diferencia radica en dos detalles: **(1)** Las curvas Spline **interpolan** todos los puntos de control o nodos y las B-spline no. **(2)** Todos los segmentos de las curvas Spline dependen simultaneamente de todos los puntos de control, mientras que los segmentos de las B-spline dependen sólo de los puntos de control más cercanos.

Una tercera diferencia — incidental si se quiere — es que **(3)** las B-spline, se calculan mucho más rápido que las Spline, debido a la alta complejidad del algoritmo para calcular las últimas.

9.3.2. Descripción geométrica

El término *Spline* proviene de las largas tiras flexibles de metal que usaban los constructores para establecer las superficies de aeroplanos, automóviles y barcos. A estas tiras se le colocaban pesos o prensas que servían para jalarlas en determinadas direcciones para forzar la forma que adoptarían las tiras durante su vida útil. Y a menos que se sometan a grandes tensiones, estas tiras mantienen continuidad C^2 una vez liberadas de las prensas.

En la actualidad, los dibujantes y arquitectos usan tiras semirígidas de plástico para dibujar curvas suaves, necesitando a veces, que interpolen ciertos puntos preestablecidos. Es el mismo principio que con los fuselajes.

9.3.3. Descripción matemática

Definición (adaptado de [Burden y Faires 2002]): Dada una función f definida en $[a, b]$ y un conjunto de nodos $a = x_0 < x_1 < \ldots < x_n = b$, un trazador interpolante cúbico S para f es una función que cumple con las condiciones siguientes:

1. $S(x)$ es una función seccionada, y en todos los casos, es un polinomio cúbico, denotado por $S_j(x)$, en el subintervalo $[x_j, x_{j+1}]$ para cada $j = 0, 1, \ldots, n-1$;
2. $S(x_j) = f(x_j)$ para cada $j = 0, 1, \ldots, n$;
3. $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$ para cada $j = 0, 1, \ldots, n-2$;
4. $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$ para cada $j = 0, 1, \ldots, n-2$;
5. $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$ para cada $j = 0, 1, \ldots, n-2$;
6. Una de las siguientes condiciones de frontera se satisface:
   a) $S''(x_0) = S''(x_n) = 0$ (frontera libre o natural);
   b) $S'(x_0) = f'(x_0)$ y $S'(x_n) = f'(x_n)$ (frontera sujeta).

El trazador S tiene la siguiente forma:

$$S(x) = \begin{cases} S_0(x) & x_0 \leq x < x_1 \\ S_1(x) & x_1 \leq x < x_2 \\ \vdots & \vdots \\ S_{n-1}(x) & x_{n-1} \leq x \leq x_n \end{cases}$$

donde $S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$ para cada $j = 0, 1, \ldots, n-1$. Está claro que $S_j(x_j) = a_j = f(x_j)$.

Hay varias cosas que hay que notar: Para un valor de x concreto, se opera con el nodo más cercano por la izquierda. Por otro lado, es de recalcar que las condiciones 3, 4 y 5 garantizan continuidad C^2. Además, la condición 2 implica que la curva interpola todos los nodos.

9.3.4. Algoritmo de cálculo de coeficientes

Para construir el trazador interpolante cúbico natural S de la función f, que se define en los números $x_0 < x_1 < \ldots < x_n$ y que satisface $S''(x_0) = S''(x_n) = 0$ se procede de la siguiente manera (algoritmo adaptado del *algoritmo 3.4* de [Burden y Faires 2002, p. 146]):

Valores de entrada:

$$n;\ x_0, x_1, \ldots, x_n;\ a_0 = f(x_0), a_1 = f(x_1), \ldots, a_n = f(x_n)$$

Algoritmo:

1. INICIO
2. PARA $i = 0, 1, ..., n-1$
   a) $h_i = x_{i+1} - x_i$
3. PARA $i = 1, 2, ..., n-1$
   a) $\alpha_i = \frac{3}{h_i}(a_{i+1} - a_i) - \frac{3}{h_{i-1}}(a_i - a_{i-1})$
4. HACER[1]:
   $l_0 = 1$
   $\mu_0 = 0$
   $z_0 = 0$
5. PARA $i = 1, 2, ..., n-1$
   a) HACER:
   $l_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$
   $\mu_i = \frac{h_i}{l_i}$
   $z_i = \frac{\alpha_i - h_{i-1}z_{i-1}}{l_i}$
6. HACER:
   $c_n = 0$
7. PARA $j = n-1, n-2, ..., 0$
   a) HACER:
   $c_j = z_j - \mu_j c_{j+1}$
   $b_j = \frac{a_{j+1} - a_j}{h_j} - \frac{h_j(c_{j+1} + 2c_j)}{3}$
   $d_j = \frac{c_{j+1} - c_j}{3h_j}$
8. FIN

Valores de salida:

a_j, b_j, c_j, d_j para $j = 0, 1, ..., n-1$

[1] Los pasos 4, 5, 6 y parte del 7 resuelven un sistema lineal tridiagonal descrito en el *algoritmo 6.7* de [Burden y Faires 2002, p. 408]

9.3.5. Algoritmos de cálculo de puntos interpolados

Cálculo de un sólo punto[2]

Este algoritmo recibe un valor de la variable independiente y calcula su imagen aproximada utilizando los trazadores.

Valores de entrada: x; n; $x_0, x_1, \ldots, x_n$ y a_j, b_j, c_j, d_j para $j = 0, 1, \ldots, n-1$

1. INICIO
2. SI $x \geq x_0$
   - *a*) $j = 0$
   - *b*) MIENTRAS $(j < n) \wedge (x > x_{j+1})$
     1) $j = j + 1$
   - *c*) SI $j < n$
     1) HACER:
        $y = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$
        o bien con la regla de Horner:
        $y = a_j + (x - x_j)\left(b_j + (x - x_j)\left(c_j + (x - x_j)d_j\right)\right)$
     2) FIN
3. ERROR: "El valor de x, está fuera del dominio de S".
4. FIN

Valores de salida: y para obtener $(x, y) \in S$ ó ¡ERROR!.

Cálculo de «todos» los puntos

Este algoritmo recibe los trazadores y un valor p que indica el número de bloques interpolados que se generarán (es decir que se calcularán $p + 1$ puntos uniformemente espaciados, en $[x_0, x_n]$).

Valores de entrada: p; n; $x_0, x_1, \ldots, x_n$ y a_j, b_j, c_j, d_j para $j = 0, 1, \ldots, n-1$

1. INICIO
2. HACER: $i = 1$
3. PARA $k = 0, 1, \ldots, p$
   - *a*) HACER: $\widetilde{x}_k = x_0 + \frac{k}{p}(x_n - x_0)$
   - *b*) MIENTRAS $(\widetilde{x}_k > x_i)$

[2] Este algoritmo implementa búsqueda lineal, pero podría ser más eficiente con búsqueda binaria.

1) HACER: $i = i + 1$

c) HACER $i = i - 1$

d) HACER:
$\widetilde{y}_k = a_i + b_i(\widetilde{x}_k - x_i) + c_i(\widetilde{x}_k - x_i)^2 + d_i(\widetilde{x}_k - x_i)^3$
o bien con la regla de Horner:
$\widetilde{y}_k = a_i + (\widetilde{x}_k - x_i)\left(b_i + (\widetilde{x}_k - x_i)\left(c_i + (\widetilde{x}_k - x_i)d_i\right)\right)$

4. FIN

Valores de salida: $\widetilde{x}_0, \widetilde{x}_1, \ldots, \widetilde{x}_p$ y $\widetilde{y}_0, \widetilde{y}_1, \ldots, \widetilde{y}_p$.

9.3.6. Extensión para curvas paramétricas multidimensionales

El caso bidimensional

Para conectar los puntos $(x_0,y_0), (x_1,y_1), \ldots, (x_n, y_n)$, en el orden dado, aún cuando no formen una función, se usa una nueva variable t en un intervalo $[t_0,t_n]$ (típicamente $[t_0 = 0, t_n = 1]$) con $t_0 < t_1 < \ldots < t_n$ (la distribución puede ser uniforme o no).

Se recomponen los pares ordenados dados como si fueran dos problemas diferentes, así:

$$\Rightarrow (t_0,x_0), (t_1,x_1), ..., (t_n, x_n)$$
$$\Rightarrow (t_0,y_0), (t_1,y_1), ..., (t_n, y_n)$$

Se construyen entonces dos trazadores interpolantes cúbicos, $x = S^{(x)}(t)$ y $y = S^{(y)}(t)$. Así que los valores que interpolan los puntos originales son:

$$(x,y) = \begin{cases} \left(S_0^{(x)}(t), S_0^{(y)}(t)\right) & t_0 \le t < t_1 \\ \left(S_1^{(x)}(t), S_1^{(y)}(t)\right) & t_1 \le t < t_2 \\ \vdots & \vdots \\ \left(S_{n-1}^{(x)}(t), S_{n-1}^{(y)}(t)\right) & t_{n-1} \le t \le t_n \end{cases}$$

donde $S_j^{(x)}(t) = a_j^{(x)} + b_j^{(x)}(t - t_j) + c_j^{(x)}(t - t_j)^2 + d_j^{(x)}(t - t_j)^3$ y
$S_j^{(y)}(t) = a_j^{(y)} + b_j^{(y)}(t - t_j) + c_j^{(y)}(t - t_j)^2 + d_j^{(y)}(t - t_j)^3$ para cada $j = 0, 1, ..., n - 1$.

El caso tridimensional

Para conectar los puntos $(x_0,y_0,z_0), (x_1,y_1,z_1), \ldots, (x_n,y_n,z_n)$, en el orden dado, se usa una nueva variable t en un intervalo $[t_0,t_n]$ (de nuevo, típicamente $[t_0 = 0, t_n = 1]$) con $t_0 < t_1 < ... < t_n$.

Se recomponen los pares ordenados dados como si fueran tres problemas diferentes, así:

$$\Rightarrow (t_0,x_0),(t_1,x_1),...,(t_n,x_n)$$
$$\Rightarrow (t_0,y_0),(t_1,y_1),...,(t_n,y_n)$$
$$\Rightarrow (t_0,z_0),(t_1,z_1),...,(t_n,z_n)$$

Se construyen entonces tres trazadores interpolantes cúbicos, $x = S^{(x)}(t)$, $y = S^{(y)}(t)$ y $z = S^{(z)}(t)$. Así que los valores que interpolan los puntos originales son:

$$(x,y,z) = \begin{cases} \left(S_0^{(x)}(t), S_0^{(y)}(t), S_0^{(z)}(t)\right) & t_0 \le t < t_1 \\ \left(S_1^{(x)}(t), S_1^{(y)}(t), S_1^{(z)}(t)\right) & t_1 \le t < t_2 \\ \vdots & \vdots \\ \left(S_{n-1}^{(x)}(t), S_{n-1}^{(y)}(t), S_{n-1}^{(z)}(t)\right) & t_{n-1} \le t \le t_n \end{cases}$$

donde $S_j^{(x)}(t) = a_j^{(x)} + b_j^{(x)}(t-t_j) + c_j^{(x)}(t-t_j)^2 + d_j^{(x)}(t-t_j)^3$,

$S_j^{(y)}(t) = a_j^{(y)} + b_j^{(y)}(t-t_j) + c_j^{(y)}(t-t_j)^2 + d_j^{(y)}(t-t_j)^3$ y

$S_j^{(z)}(t) = a_j^{(z)} + b_j^{(z)}(t-t_j) + c_j^{(z)}(t-t_j)^2 + d_j^{(z)}(t-t_j)^3$ para cada $j = 0, 1, ..., n-1$.

Eficiencia en el cálculo de un sólo punto

Los algoritmos usados para construir y calcular estos trazadores interpolantes cúbicos paramétricos no deben, por razones de eficiencia, realizarse de forma completamente independiente. En lugar de eso, se debe procurar aprovechar los cálculos para maximizar la eficiencia de tales procesos.

Como ejemplo de ello, considere la siguiente versión del algorito de cálculo de un punto interpolado, para el caso paramétrico con dos dimensiones:

Valores de entrada: t; n; $t_0, t_1, \ldots, t_n$; $a_j^{(x)}, b_j^{(x)}, c_j^{(x)}, d_j^{(x)}$; y $a_j^{(y)}, b_j^{(y)}, c_j^{(y)}, d_j^{(y)}$ para $j = 0, 1, ..., n-1$

1. INICIO
2. SI $t \ge t_0$
   - *a*) $j = 0$
   - *b*) MIENTRAS $(j < n) \wedge (t > t_{j+1})$
     - 1) $j = j + 1$
   - *c*) SI $j < n$

1) HACER:
$x = a_j^{(x)} + b_j^{(x)}(t - t_j) + c_j^{(x)}(t - t_j)^2 + d_j^{(x)}(t - t_j)^3$
$y = a_j^{(y)} + b_j^{(y)}(t - t_j) + c_j^{(y)}(t - t_j)^2 + d_j^{(y)}(t - t_j)^3$
2) FIN

3. ERROR: "El valor de t, está fuera del dominio de S".
4. FIN

Valores de salida: x; y, para obtener $(x, y) \in S$ ó ¡ERROR!.

Eficiencia en el cálculo todos los puntos

Así mismo, se muestra el algoritmo que calcula $p+1$ puntos uniformemente espaciados, en $[t_0, t_n]$):

Valores de entrada: p; n; $t_0, t_1, \ldots, t_n$; $a_j^{(x)}, b_j^{(x)}, c_j^{(x)}, d_j^{(x)}$ y $a_j^{(y)}, b_j^{(y)}, c_j^{(y)}, d_j^{(y)}$ para $j = 0, 1, \ldots, n-1$

1. INICIO
2. HACER: $i = 1$
3. PARA $k = 0, 1, \ldots, p$
   a) HACER: $\widetilde{t}_k = t_0 + \frac{k}{p}(t_n - t_0)$
   b) MIENTRAS $\left(\widetilde{t}_k > t_i\right)$
      1) HACER: $i = i + 1$
   c) HACER $i = i - 1$
   d) HACER:
   $\widetilde{x}_k = a_i^{(x)} + b_i^{(x)}(\widetilde{t}_k - t_i) + c_i^{(x)}(\widetilde{t}_k - t_i)^2 + d_i^{(x)}(\widetilde{t}_k - t_i)^3$
   $\widetilde{y}_k = a_i^{(y)} + b_i^{(y)}(\widetilde{t}_k - t_i) + c_i^{(y)}(\widetilde{t}_k - t_i)^2 + d_i^{(y)}(\widetilde{t}_k - t_i)^3$
4. FIN

Valores de salida: $\widetilde{x}_0, \widetilde{x}_1, \ldots, \widetilde{x}_p$ y $\widetilde{y}_0, \widetilde{y}_1, \ldots, \widetilde{y}_p$.

9.3.7. Ejemplo de implementación

A continucación presentamos un sencillo programa en lenguaje c estándar que implementa la manipulación de trazadores interpolantes cúbicos paramétricos de dos dimensiones. La aplicación simplemente muestra una pantalla con una serie de nodos conectados por una curva spline y permite mover los puntos (los nodos) al tiempo que la curva se adapta a la forma que los puntos exigen.

En los siguientes archivos se especifican la estructura de datos usada y las funciones que la operarán (nótese que es de propósito general).

Listing 9.1: Archivo de cabecera de funciones de Trazadores

```
1  // /c09/trazadores/tic.h
2  #include <stdio.h>
3
4  #define TIC_COD_EXITO 0
5  #define TIC_MSG_EXITO "Éxito\n"
6  #define TIC_COD_ERROR_PARAMETROS -1
7  #define TIC_MSG_ERROR_PARAMETROS "Error␣de␣parámetro(s)\n"
8  #define TIC_COD_ERROR_DOMINIO -2
9  #define TIC_MSG_ERROR_DOMINIO "Error␣de␣dominio\n"
10 #define TIC_COD_ERROR_MEMORIA -3
11 #define TIC_MSG_ERROR_MEMORIA "Error␣de␣solicitud␣de␣memoria␣dinámica\n"
12
13 #define H    0
14 #define ALFA 1
15 #define L    2
16 #define MIU  3
17 #define Z    4
18
19 /*
20     Contiene toda la información para calcular
21     los trazadores.
22
23     El cálculo de los trazadores es el siguiente:
24         S(x) = S[j](x) =
25             a[j] + b[j]*(x-x[j]) + c[j]*(x-x[j])^2 + d[j]*(x-x[j])^3 =
26             a[j] + (x-x[j]) * ( b[j] + (x-x[j]) * ( c[j] + (x-x[j]) * d[j
                   ] ) )
27         para x[j] <= x <= x[j+1]
28         (nótese que los cálculos se hacen con
29         el nodo más cercano a 'x' por la izquierda)
30
31     Se asume que:
32         x[0] < x[1] < ... < x[num_trazadores]
33 */
34 typedef struct trazadores{
35
36     //coordenadas de los nodos
37     double *x;
38     //las imágenes 'y' están en el arreglo 'a'
39     double *y;
40
41     //coeficientes de los polinomios
42     double *a;
43     double *b;
44     double *c;
45     double *d;
```

```
46
47      //número de polinomios
48      int num_trazadores;
49
50      //siempre debe cumplirse la siguiente relación:
51      //num_nodos = num_trazadores + 1
52
53  } trazadores;
54
55
56  /*
57  Entradas:
58      ('x','y') define los nodos
59      'num_nodos' indica el número de nodos
60  Salida:
61      Se devuelve un puntero a la estructura creada o NULL si hubo error.
62  */
63  trazadores *TIC_crear_trazadores(double x[], double y[], int num_nodos);
64
65  /*
66  Toma el valor 'x' y calcula su imagen
67  en función del trazador que le corresponda
68  y la deposita en 'y'
69  */
70  int TIC_calcular(trazadores *tr, double x, double *y);
71
72  /*
73  Entradas:
74      ('x','y') define los nodos
75  Salida:
76      Se devuelve un codigo de resultado.
77
78  Se asume que los arreglos son de la misma longitud entre sí
79  y con el valor 'num_nodos' usado para crear a 'tr'
80  */
81  int TIC_modificar_trazador(trazadores* tr, double x[], double y[]);
82
83
84  /*
85      Libera la memoria dinámicamente solicitada de la estructura
86  */
87  void TIC_liberar_memoria(trazadores *tr);
88
89  /*
90  Imprime en la salida especificada los valores
91  almacenados en la estructura.
92  */
93  void TIC_imprimir(trazadores *tr, FILE *f);
94
95  /*************** Paramétricos 2D *****************************/
```

```
96
97  typedef struct trazadoresP{
98
99      //coordenadas de los nodos
100     double *t;
101
102     //las imágenes 'x' están en el arreglo 'ax'
103     double *x;
104     //coeficientes de los polinomios en x
105     double *ax;
106     double *bx;
107     double *cx;
108     double *dx;
109
110     //las imágenes 'y' están en el arreglo 'ay'
111     double *y;
112     //coeficientes de los polinomios en y
113     double *ay;
114     double *by;
115     double *cy;
116     double *dy;
117
118     //número de polinomios
119     int num_trazadores;
120
121     //la siguiente relación siempre debe cumplirse:
122     //num_nodos = num_trazadores + 1
123
124 } trazadoresP;
125
126 trazadoresP *TIC_crear_trazadoresP(double x[], double y[], int num_nodos)
        ;
127
128 /*
129 Toma el valor 't' y calcula su imagen
130 en función del trazador que le corresponda
131 y la deposita en ('x','y')
132 */
133 int TIC_calcularP(trazadoresP *tr, double t, double *x, double *y);
134
135 /*
136 Se asume que los arreglos son de la misma longitud entre sí
137 y con el valor 'num_nodos' usado para crear 'tr'
138 */
139 int TIC_modificar_trazadorP(trazadoresP* tr, double x[], double y[]);
140
141 /*
142 Cuando se han actualizado los valores de los nodos
143 pero no se han agregado o quitado.
144 Se recalculan los coeficientes.
```

```
145 */
146 int TIC_actualizar_trazadorP(trazadoresP* tr);
147
148 void TIC_liberar_memoriaP(trazadoresP *tr);
149
150 void TIC_imprimirP(trazadoresP *tr, FILE *f);
```

Listing 9.2: Código fuente de funciones de trazadores

```
1  // /c09/trazadores/tic.c
2  #include "tic.h"
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  trazadores *TIC_crear_trazadores(double x[], double y[], int num_nodos){
7      trazadores *tr = NULL;
8      int i;
9
10     if((tr = (trazadores*)malloc(sizeof(trazadores)))==NULL){
11         fprintf(stderr, TIC_MSG_ERROR_MEMORIA);
12         return NULL;
13     }
14     tr->x = tr->y = tr->a = tr->b = tr->c = tr->d = NULL;
15
16     /*
17     'x', 'y' y 'a' requieren 'num_nodos' elementos,
18     'b', 'c' y 'd' sólo 'num_nodos-1=num_trazadores',
19     pero si 'c' tiene 'num_nodos', se simplifica la parte final del
           algoritmo
20     */
21     if((tr->x = (double*)malloc(5 * num_nodos * sizeof(double)))==NULL){
22         fprintf(stderr, TIC_MSG_ERROR_MEMORIA);
23         TIC_liberar_memoria(tr);
24         free(tr);
25         return NULL;
26     }
27
28     for(i=0; i < 5 * num_nodos; i++)
29         tr->x[i]=0.0;
30
31     tr->a = tr->y =
32             tr->x + num_nodos;
33     tr->b = tr->a + num_nodos;
34     tr->c = tr->b + num_nodos;
35     tr->d = tr->c + num_nodos;
36
37     tr->num_trazadores = num_nodos - 1;
38
39     if(!TIC_modificar_trazador(tr, x, y))
40         return tr;
```

```
41      else{
42          TIC_liberar_memoria(tr);
43          free(tr);
44          return NULL;
45      }
46  }
47
48  int TIC_modificar_trazador(trazadores* tr, double x[], double y[]){
49      int i;
50      int n;
51      double *tmp[5];
52
53      //Verificar integridad de los parámetros
54      if(tr==NULL ||
55          tr->a == NULL || tr->b == NULL || tr->c == NULL || tr->d == NULL
                ||
56          x==NULL || y == NULL ||
57          tr->num_trazadores < 1)
58      {
59          fprintf(stderr, TIC_MSG_ERROR_PARAMETROS);
60          return TIC_COD_ERROR_PARAMETROS;
61      }
62
63      //Gestionar memoria temporal para el algoritmo
64      if(
65          (tmp[0] = (double*)malloc(5 * tr->num_trazadores * sizeof(double)
                ))
66          ==NULL)
67      {
68          fprintf(stderr, TIC_MSG_ERROR_MEMORIA);
69          return TIC_COD_ERROR_MEMORIA;
70      }
71      for(i=1; i<5; i++)
72          tmp[i] = tmp[0] + i * tr->num_trazadores;
73
74
75      //Cálculo de los trazadores
76      n = tr->num_trazadores;
77      for(i=0; i<=n; i++){
78          tr->x[i] = x[i];
79          tr->a[i] = y[i]; //Copiar los valores
80      }
81      //paso 2...
82      for(i=0; i<n; i++)
83          tmp[H][i] = tr->x[i+1] - tr->x[i];
84      //paso 3...
85      for(i=1; i<n; i++)
86          tmp[ALFA][i] = 3 * (
87              (tr->a[i+1] - tr->a[i]) / tmp[H][i] -
88              (tr->a[i] - tr->a[i-1]) / tmp[H][i-1]
```

```
89          );
90      //paso 4...
91      tmp[L][0] = 1;
92      tmp[MIU][0] = 0;
93      tmp[Z][0] = 0;
94
95      //paso 5...
96      for(i=1; i<n; i++){
97          tmp[L][i] = 2 * (tr->x[i+1] - tr->x[i-1]) - tmp[H][i-1]*tmp[MIU][
                i-1];
98          tmp[MIU][i] = tmp[H][i] / tmp[L][i];
99          tmp[Z][i] = (tmp[ALFA][i] - tmp[H][i-1]*tmp[Z][i-1]) / tmp[L][i];
100     }
101     //paso 6...
102     tr->c[n] = 0;
103
104     //paso 7...
105     for(i=n-1; i>=0; i--){
106         tr->c[i] = tmp[Z][i] - tmp[MIU][i] * tr->c[i+1];
107         tr->b[i] = (tr->a[i+1] - tr->a[i]) / tmp[H][i]
108             - tmp[H][i] * (tr->c[i+1] + 2*tr->c[i]) / 3;
109         tr->d[i] = (tr->c[i+1] - tr->c[i]) / (3*tmp[H][i]);
110     }
111
112     //Liberar memoria temporal
113     free(tmp[0]);
114
115     return TIC_COD_EXITO;
116 }
117
118 int TIC_calcular(trazadores *tr, double x, double *y){
119
120     if(tr==NULL || y == NULL){
121         fprintf(stderr, TIC_MSG_ERROR_PARAMETROS);
122         return TIC_COD_ERROR_PARAMETROS;
123     }
124
125     if(x >= tr->x[0]){
126         /*
127         int j=0;
128
129         while(j < tr->num_trazadores && x > tr->x[j+1])
130             j++;
131         if(j < tr->num_trazadores){
132             double tmpX = x - tr->x[j];
133             *y = tr->a[j] + tmpX * ( tr->b[j] + tmpX * ( tr->c[j] + tmpX
                    * tr->d[j] ) );
134             return TIC_COD_EXITO;
135         }
136         */
```

```
137
138         //Esta alternativa es más eficiente que la de arriba:
139         int j=1;
140
141         while(j <= tr->num_trazadores && x > tr->x[j])
142             j++;
143         if(--j < tr->num_trazadores){
144             double tmpX = x - tr->x[j];
145             *y = tr->a[j] + tmpX * ( tr->b[j] + tmpX * ( tr->c[j] + tmpX
                    * tr->d[j] ) );
146             return TIC_COD_EXITO;
147         }
148     }
149
150     //fprintf(stderr, TIC_MSG_ERROR_DOMINIO);
151     return TIC_COD_ERROR_DOMINIO;
152
153 }
154
155 void TIC_liberar_memoria(trazadores *tr){
156     free(tr->x);
157 }
158
159 void TIC_imprimir(trazadores *tr, FILE *f){
160     int i;
161     fprintf(f,"\nTrazadores:\n");
162     fprintf(f,"i\tx_i\ty_i\ta_i\tb_i\tc_i\td_i\n");
163     for(i=0; i<=tr->num_trazadores; i++)
164         fprintf(f,"%d\t%3.2g\t%3.2g\t%3.2g\t%3.2g\t%3.2g\t%3.2g\n",
165         i, tr->x[i], tr->y[i],
166         tr->a[i], tr->b[i], tr->c[i], tr->d[i]);
167 }
168
169 /**** Paramétricos *********************/
170 trazadoresP *TIC_crear_trazadoresP(double x[], double y[], int num_nodos)
        {
171     trazadoresP *tr = NULL;
172     int i;
173
174     if((tr = (trazadoresP*)malloc(sizeof(trazadoresP)))==NULL){
175         fprintf(stderr, TIC_MSG_ERROR_MEMORIA);
176         return NULL;
177     }
178     tr->x = tr->y = tr->t
179         = tr->ax = tr->bx = tr->cx = tr->dx
180         = tr->ay = tr->by = tr->cy = tr->dy = NULL;
181
182     /*
183     'x', 'y' y 'a' requieren 'num_nodos' elementos,
184     'b', 'c' y 'd' sólo 'num_nodos-1=num_trazadores',
```

```
185     pero si 'c' tiene 'num_nodos', se simplifica la parte final del
            algoritmo,
186     por lo que todos tendrán el mismo tamaño de 'num_nodos'.
187     */
188     if((tr->t = (double*)malloc(9 * num_nodos * sizeof(double)))==NULL){
189         fprintf(stderr, TIC_MSG_ERROR_MEMORIA);
190         free(tr);
191         return NULL;
192     }
193
194     for(i=0; i < 9 * num_nodos; i++)
195         tr->t[i]=0.0;
196
197     tr->ax = tr->x =
198               tr->t + num_nodos;
199     tr->bx = tr->ax + num_nodos;
200     tr->cx = tr->bx + num_nodos;
201     tr->dx = tr->cx + num_nodos;
202
203     tr->ay = tr->y =
204               tr->dx + num_nodos;
205     tr->by = tr->ay + num_nodos;
206     tr->cy = tr->by + num_nodos;
207     tr->dy = tr->cy + num_nodos;
208
209     tr->num_trazadores = num_nodos - 1;
210
211     if(!TIC_modificar_trazadorP(tr, x, y))
212         return tr;
213     else{
214         TIC_liberar_memoriaP(tr);
215         free(tr);
216         return NULL;
217     }
218 }
219
220 int TIC_calcularP(trazadoresP *tr, double t, double *x, double *y){
221
222     if(tr==NULL || x == NULL || y == NULL){
223         fprintf(stderr, TIC_MSG_ERROR_PARAMETROS);
224         return TIC_COD_ERROR_PARAMETROS;
225     }
226
227     if(t >= tr->t[0]){
228         int j=1;
229
230         while(j <= tr->num_trazadores && t > tr->t[j])
231             j++;
232         if(--j < tr->num_trazadores){
233             double tmpT = t - tr->t[j];
```

```
234             *x = tr->ax[j] + tmpT * ( tr->bx[j] + tmpT * ( tr->cx[j] +
                    tmpT * tr->dx[j] ) );
235             *y = tr->ay[j] + tmpT * ( tr->by[j] + tmpT * ( tr->cy[j] +
                    tmpT * tr->dy[j] ) );
236             return TIC_COD_EXITO;
237         }
238     }
239
240     //fprintf(stderr, TIC_MSG_ERROR_DOMINIO);
241     return TIC_COD_ERROR_DOMINIO;
242
243 }
244
245
246 int TIC_actualizar_trazadorP(trazadoresP* tr){
247     return TIC_modificar_trazadorP(tr, tr->ax, tr->ay);
248 }
249
250 int TIC_modificar_trazadorP(trazadoresP* tr, double x[], double y[]){
251     int i;
252     int n;
253     double *tmp[5];
254
255     //Verificar integridad de los parámetros
256     if(tr==NULL ||
257         tr->ax == NULL || tr->bx == NULL || tr->cx == NULL || tr->dx ==
                NULL ||
258         tr->ay == NULL || tr->by == NULL || tr->cy == NULL || tr->dy ==
                NULL ||
259         x == NULL || y == NULL ||
260         tr->num_trazadores < 1)
261     {
262         fprintf(stderr, TIC_MSG_ERROR_PARAMETROS);
263         return TIC_COD_ERROR_PARAMETROS;
264     }
265
266     //Gestionar memoria temporal para el algoritmo
267     if(
268         (tmp[0] = (double*)malloc(5 * tr->num_trazadores * sizeof(double)
                ))
269         ==NULL)
270     {
271         fprintf(stderr, TIC_MSG_ERROR_MEMORIA);
272         return TIC_COD_ERROR_MEMORIA;
273     }
274     for(i=1; i<5; i++)
275         tmp[i] = tmp[0] + i * tr->num_trazadores;
276
277     //Cálculo de los trazadores
278     n = tr->num_trazadores;
```

```
279
280     //Distribuir uniformemente el parametro 't'
281     //y asignar los coeficientes 'a'
282     for(i=0; i<=n; i++){
283         tr->t[i] = i / (float)n;
284         tr->ax[i] = x[i];
285         tr->ay[i] = y[i];
286     }
287     //paso 2...
288     for(i=0; i<n; i++)
289         tmp[H][i] = tr->t[i+1] - tr->t[i];
290 /**** en x *****************/
291     //paso 3...
292     for(i=1; i<n; i++)
293         tmp[ALFA][i] = 3 * (
294             (tr->ax[i+1] - tr->ax[i]) / tmp[H][i] -
295             (tr->ax[i] - tr->ax[i-1]) / tmp[H][i-1]
296         );
297     //paso 4...
298     tmp[L][0] = 1;
299     tmp[MIU][0] = 0;
300     tmp[Z][0] = 0;
301
302     //paso 5...
303     for(i=1; i<n; i++){
304         tmp[L][i] = 2 * (tr->t[i+1] - tr->t[i-1]) - tmp[H][i-1]*tmp[MIU][
                i-1];
305         tmp[MIU][i] = tmp[H][i] / tmp[L][i];
306         tmp[Z][i] = (tmp[ALFA][i] - tmp[H][i-1]*tmp[Z][i-1]) / tmp[L][i];
307     }
308     //paso 6...
309     //tmp[L][n] = 1;
310     //tmp[Z][n] = 0;
311     tr->cx[n] = 0;
312
313     //paso 7...
314     for(i=n-1; i>=0; i--){
315         tr->cx[i] = tmp[Z][i] - tmp[MIU][i] * tr->cx[i+1];
316         tr->bx[i] = (tr->ax[i+1] - tr->ax[i]) / tmp[H][i]
317             - tmp[H][i] * (tr->cx[i+1] + 2*tr->cx[i]) / 3;
318         tr->dx[i] = (tr->cx[i+1] - tr->cx[i]) / (3*tmp[H][i]);
319     }
320 /**** en y *****************/
321     //paso 3...
322     for(i=1; i<n; i++)
323         tmp[ALFA][i] = 3 * (
324             (tr->ay[i+1] - tr->ay[i]) / tmp[H][i] -
325             (tr->ay[i] - tr->ay[i-1]) / tmp[H][i-1]
326         );
327     //paso 4...
```

```
328     tmp[L][0] = 1;
329     tmp[MIU][0] = 0;
330     tmp[Z][0] = 0;
331 
332     //paso 5...
333     for(i=1; i<n; i++){
334         tmp[L][i] = 2 * (tr->t[i+1] - tr->t[i-1]) - tmp[H][i-1]*tmp[MIU][
                i-1];
335         tmp[MIU][i] = tmp[H][i] / tmp[L][i];
336         tmp[Z][i] = (tmp[ALFA][i] - tmp[H][i-1]*tmp[Z][i-1]) / tmp[L][i];
337     }
338     //paso 6...
339     //tmp[L][n] = 1;
340     //tmp[Z][n] = 0;
341     tr->cy[n] = 0;
342 
343     //paso 7...
344     for(i=n-1; i>=0; i--){
345         tr->cy[i] = tmp[Z][i] - tmp[MIU][i] * tr->cy[i+1];
346         tr->by[i] = (tr->ay[i+1] - tr->ay[i]) / tmp[H][i]
347             - tmp[H][i] * (tr->cy[i+1] + 2*tr->cy[i]) / 3;
348         tr->dy[i] = (tr->cy[i+1] - tr->cy[i]) / (3*tmp[H][i]);
349     }
350 /**********************/
351 
352     //Liberar memoria temporal
353     free(tmp[0]);
354 
355     return TIC_COD_EXITO;
356 }
357 
358 
359 void TIC_liberar_memoriaP(trazadoresP *tr){
360     free(tr->t);
361 }
362 
363 void TIC_imprimirP(trazadoresP *tr, FILE *f){
364     int i;
365     fprintf(f,"\nTrazadores␣Paramétricos:\n");
366     fprintf(f,"i␣t_i\tx_i\ty_i\tax_i\tbx_i\tcx_i\tdx_i\tay_i\tby_i\tcy_i\
            tdy_i\n");
367     for(i=0; i<=tr->num_trazadores; i++)
368         fprintf(f,"%d->\t%3.2g\t%3.2g\t%3.2g\n\t%3.2g\t%3.2g\t%3.2g\t%3.2
                g\n\t%3.2g\t%3.2g\t%3.2g\t%3.2g\n",
369         i, tr->t[i], tr->x[i], tr->y[i],
370         tr->ax[i], tr->bx[i], tr->cx[i], tr->dx[i],
371         tr->ay[i], tr->by[i], tr->cy[i], tr->dy[i]);
372 }
```

Este es el programa principal, que utiliza el código de los dos archivos anteriores. No

es nada espectacular, ni aplicado. Es simplemente para comprender la naturaleza geométrica de las curvas spline.

Listing 9.3: Programa principal

```
// /c09/trazadores/main.c
#include <SDL/SDL.h>
#include "tic.h"
#include <SDL/SDL_gfxPrimitives.h>
#include <stdio.h>

#define ANCHO 640
#define ALTO  480

#define ANCHO_RECTANGULO 10

#define XMIN  0.0
#define XMAX 15.0
#define YMIN  0.0
#define YMAX  3.0
#define TPASO 0.005

#define TAM 21

static inline int x_real(double x_virtual){
    return (int) (ANCHO * x_virtual / XMAX);
}
static inline int y_real(double y_virtual){
    return (int) (ALTO * (1.0 - y_virtual / YMAX));
}
static inline double x_virtual(int x_real){
    return (XMAX * x_real) / ANCHO;
}
static inline double y_virtual(int y_real){
    return YMAX * (1.0 - y_real / (double)ALTO);
}

//Variables globales
double x[TAM] = {0.9, 1.3, 1.9, 2.1, 2.6, 3.0, 3.9, 4.4, 4.7, 5.0, 6.0,
    7.0, 8.0, 9.2, 10.5, 11.3, 11.6, 12.0, 12.6, 13.0, 13.3};
double y[TAM] = {1.3, 1.5, 1.85, 2.1, 2.6, 2.7, 2.4, 2.15, 2.05, 2.1,
    2.25, 2.3, 2.25, 1.95, 1.4, 0.9, 0.7, 0.6, 0.5, 0.4, 0.25};
trazadoresP *tr = NULL;
int i, j, k, l;
double tvar, xvar, yvar;
Uint32 color_fondo, color1, color2;

Uint32 gfxColor(Uint8 r, Uint8 g, Uint8 b){
    return
        r << 24 |
        g << 16 |
```

```
45          b << 8  |
46          255;    //este valor es la opacidad del color
47                  //y debe ser máxima para que sea sólido
48  }
49
50  void dibujo(SDL_Surface *pantalla){
51      i=j=k=l=0;
52      SDL_Rect rect;
53
54      //Borra la pantalla
55      SDL_FillRect(pantalla, NULL, color_fondo);
56
57      //dibujo de los rectángulos
58      rect.w = rect.h = ANCHO_RECTANGULO;
59      for(i=0; i<=tr->num_trazadores; i++){
60          rect.x = x_real(tr->x[i])-ANCHO_RECTANGULO/2;
61          rect.y = y_real(tr->y[i])-ANCHO_RECTANGULO/2;
62          rect.h = rect.w = ANCHO_RECTANGULO;
63          rectangleColor(pantalla, rect.x, rect.y, rect.x + rect.w, rect.y
                + rect.h, color2);
64      }
65
66      //dibujo de la curva
67      tvar = 0.0;
68      TIC_calcularP(tr, tvar, &xvar, &yvar);
69      i=x_real(xvar);
70      j=y_real(yvar);
71      for(tvar = TPASO; tvar <= 1; tvar+=TPASO){
72          if(!TIC_calcularP(tr, tvar, &xvar, &yvar)){
73              aalineColor(pantalla, i, j, k=x_real(xvar), l=y_real(yvar),
                    color1);
74              i=k; j=l;
75          }
76      }
77      TIC_calcularP(tr, 1.0, &xvar, &yvar);
78      aalineColor(pantalla, i, j, k=x_real(xvar), l=y_real(yvar), color1);
79
80      //vuelca el buffer en la pantalla:
81      SDL_Flip (pantalla);
82  }
83
84  int main(int argc, char *argv[]){
85      SDL_Surface *pantalla = NULL;
86      SDL_Event evento;
87      int profundidad_color;
88      const SDL_VideoInfo *info;
89      int i;
90
91      int corriendo = 1;
92      int seleccionado = 0;
```

```
93
94      if(SDL_Init(SDL_INIT_VIDEO) < 0 ) {
95          fprintf(stderr, "No se puede iniciar SDL: %s\n", SDL_GetError());
96          exit(1);
97      }
98      atexit(SDL_Quit);
99
100     //Este if es importante para poder usar SDL_gfx
101     info = SDL_GetVideoInfo();
102     if ( info->vfmt->BitsPerPixel > 8 ) {
103         profundidad_color = info->vfmt->BitsPerPixel;
104         //printf("%d\n", profundidad_color);
105     } else {
106         profundidad_color = 16;
107     }
108
109     pantalla = SDL_SetVideoMode(ANCHO, ALTO, profundidad_color,
            SDL_SWSURFACE);
110     if(pantalla == NULL)
111     {
112         fprintf(stderr, "No se puede establecer el modo de video %dx%d: %
                s\n",
113              ANCHO, ALTO, SDL_GetError());
114         exit(1);
115     }
116     SDL_WM_SetCaption("Trazadores Interpolantes Cúbicos Paramétricos!",
            NULL);
117
118     color_fondo = SDL_MapRGB (pantalla->format,0,0,0);
119     color1 = gfxColor(255,255,255);
120     color2 = gfxColor(128,128,128);
121     if((tr = TIC_crear_trazadoresP(x, y, TAM))==NULL){
122         fprintf(stderr, "Error al crear los trazadores\n");
123         exit(1);
124     }
125     TIC_imprimirP(tr, stderr);
126
127     dibujo(pantalla);
128
129     while(corriendo) {
130         while(SDL_PollEvent(&evento)) {
131             switch(evento.type){
132                 case SDL_MOUSEMOTION:{
133                     if(seleccionado){
134                         //actualizar el punto
135                         tr->x[i]=x_virtual(evento.button.x);
136                         tr->y[i]=y_virtual(evento.button.y);
137                         TIC_actualizar_trazadorP(tr);
138                         dibujo(pantalla);
139                     }
```

```
140                 }
141                     break;
142                 case SDL_MOUSEBUTTONDOWN:{
143                     for(i=0; i<=tr->num_trazadores; i++){
144                         if(evento.button.button == SDL_BUTTON_LEFT && //
                               si hace clic sobre un nodo...
145                             ((evento.button.x > x_real(tr->x[i])-
                                   ANCHO_RECTANGULO/2) &&
146                              (evento.button.y > y_real(tr->y[i])-
                                   ANCHO_RECTANGULO/2) &&
147                              (evento.button.x < x_real(tr->x[i])+
                                   ANCHO_RECTANGULO/2) &&
148                              (evento.button.y < y_real(tr->y[i])+
                                   ANCHO_RECTANGULO/2))
149                         ){
150                             //se selecciona y el índice queda en 'i'
151                             seleccionado = 1;
152                             //printf("seleccionado\n");
153                             break;
154                         }
155                     }
156                 }
157                     break;
158                 case SDL_MOUSEBUTTONUP:
159                     seleccionado = 0;
160                     break;
161 
162                 case SDL_QUIT:
163                     corriendo = 0;
164                     break;
165             }
166         }
167     }
168 
169     SDL_Quit();
170     return 0;
171 }
```

Listing 9.4: Archivo Makefile para la aplicación

```
1  # /c09/trazadores/Makefile
2  #Esto es un comentario
3 
4  LDFLAGS = $(shell sdl-config --cflags)
5  LDLIBS  = $(shell sdl-config --libs) -lSDL_gfx
6  ##RM       = /bin/rm -f
7 
8  #Esto indica que los siguientes identificadores,
9  #no son archivos, sino comandos de make:
10 .PHONY: limpiar
```

```
11 .PHONY: limpiartodo
12 .PHONY: all
13 #se puede por ejemplo ejecutar en la consola
14 #lo siguiente:
15 #'make limpiartodo', etc.
16
17 #Nombre del programa ejecutable:
18 PROG = trazadores
19
20 #Un '*.o' por cada '*.c'
21 OBJ = tic.o main.o
22
23 #Cuando se ejecuta 'make', se ejecuta esto:
24 all: $(PROG) limpiar
25
26 $(PROG): $(OBJ)
27         gcc -o $(PROG) $(OBJ) $(LDLIBS) $(LDFLAGS)
28
29 #Borra todos los archivos intermedios y de copia de seguridad
30 limpiar:
31         $(RM) *~ $(OBJ)
32
33 #Borra todos los archivos intermedios, de copia de seguridad
34 # y el programa ejecutable, si es que existe
35 limpiartodo:
36         make limpiar
37         $(RM) $(PROG)
```

9.3.8. Aplicación para animaciones

Como se sabe, para realizar una secuencia animada, se necesita una serie de «cuadros» con «pequeñas» variaciones entre sí y tienen que mostrarse a una velocidad suficientemente alta como para que el ojo humano no note el cambio y el cerebro (también humano) reconstruya las partes «faltantes».

Las computadoras pueden usarse para que el dibujante/artista no deba hacer tantos cuadros sino sólo unos cuantos, y sean estas quienes ayuden a completar los cuadros necesarios para que el ojo humano no note el cambio.

La idea general es que el dibujante/artista sólo realice algunos cuadros (mientras más, mejor) y estos cuadros se unan por curvas Splines a través del tiempo.

El mecanismo es el siguiente: Todos los cuadros a fundir en una sóla animación deben tener la misma cantidad de *puntos de control* (por «puntos de control» nos referimos a puntos a partir de los cuales se puede construir toda la escena). Estos puntos deben ser coherentes entre cada cuadro. A cada cuadro se le asigna un valor de tiempo en

el cual debe ocurrir, a partir del inicio de la animación. Luego se calcula una curva Spline paramétrica para cada punto de control, como se explica en la sección 9.3.6 en la página 176, donde el parámetro adicional es el tiempo.

Luego, pueden crearse tantos cuadros como se desee, aplicando un algoritmo basado en el algoritmo en la página 178.

Un ejemplo aplicado, sencillo, puede estudiarse en la aplicación `editorSplines.py` que se describe a continuación:

arrastre con clic izquierdo Sobre los cuadros que representan los puntos de control, permite moverlos.
Sobre el fondo blanco, mueve el marco virtual.

clic derecho Sobre un punto de control, agrega un puntos más.

ruedas del ratón+CTRL Aumento/disminución del detalle/finura de la animación.

ruedas del ratón+SHIFT Aumento/disminución del tamaño de los cuadros de control.

ruedas del ratón Implementa acercamiento/alejamiento.

ruedas del ratón + tecla x/y Expande/Contrae el marco virtual horizontalmente/verticalmente.

clic central Sobre un punto de control, lo borra.

CTRL+tecla Izquierda Se desplaza hacia el cuadro anterior.

CTRL+tecla Derecha Se desplaza hacia el cuadro siguiente. Crea un nuevo cuadro a partir del anterior cuando está en el último.

CTRL+tecla L Alterna dibujo de las líneas rectas que unen los puntos de control.

CTRL+tecla C Alterna dibujo de los rectángulos que marcan los puntos de control.

CTRL+tecla G Guarda el archivo (interfaz de línea de comandos).

CTRL+tecla Arriba/Abajo Aumento/disminución del grueso de los cuadros de control.

tecla Espacio Inicia la animación desde el primer cuadro hasta el último y regresa al cuadro en el que se encontraba antes de la animación.

Listing 9.5: Programa para generar animaciones sencillas en Python

```
#!/usr/bin/env python
# coding: utf-8
""" c09/bezier_py/editorBezier.py - Programa para hacer animaciones
con curvas Splines.

Los datos se guardan en archivos xml.
"""

import pygame
```

```
10 import sys, os, random
11 import splines
12
13 PAUSA = 100
14
15 ANCHO_POR_DEFECTO = 800
16 ALTO_POR_DEFECTO = 500
17
18 SUPERIOR_DER = ( 10.0, 10.0)
19 INFERIOR_IZQ = (-10.0,-10.0) #valores por defecto
20
21 __dibujarLineas = True
22 __dibujarCuadros = True
23 __dibujarCurvas = False
24
25 BOTON_IZQ = 1
26 BOTON_CEN = 2
27 BOTON_DER = 3
28 RUEDA_ARRIBA = 4
29 RUEDA_ABAJO = 5
30
31 __colorLineas = (128,128,128)
32 __colorCurva = (0,0,255)
33 __colorCuadro = (0,0,0)
34 __colorCuadroPrimero = (255,0,0)
35 __colorFondo = (255,255,255)
36
37 __gruesoLineas = 1
38
39 def __inicializarMatriz(matriz):
40     matriz.ponerEscala( \
41         (ANCHO_POR_DEFECTO, ALTO_POR_DEFECTO), \
42         (-10.0,-10.0),(10.0,10.0))
43
44
45 def dibujar(matriz, ixS, pantalla, actualizarCurvas=True):
46     '''Se␣recibe␣una␣matriz␣de␣puntos,
47 ␣␣␣␣y␣se␣dibuja␣la␣fila␣ixS-ésima␣en␣la␣pantalla␣especificada.
48 ␣␣␣␣'''
49
50     #Borrar la pantalla
51     if actualizarCurvas:
52         pantalla.fill(__colorFondo)
53
54     #Dibujar la curva:
55     if __dibujarCurvas and actualizarCurvas:
56         pass
57
58     #Dibujar los cuadrados:
59     if __dibujarCuadros:
```

```
60          pygame.draw.rect(pantalla, __colorCuadroPrimero,
61              pygame.Rect(
62                  matriz.e.vrx(matriz.puntos[0][ixS][0])-matriz.
                        anchoRectangulo/2,
63                  matriz.e.vry(matriz.puntos[0][ixS][1])-matriz.
                        anchoRectangulo/2,
64                  matriz.anchoRectangulo, matriz.anchoRectangulo), matriz.
                        gruesoRectangulo)
65          for i in range(1, len(matriz.puntos)):
66              pygame.draw.rect(pantalla, __colorCuadro,
67                  pygame.Rect(
68                      matriz.e.vrx(matriz.puntos[i][ixS][0])-matriz.
                            anchoRectangulo/2,
69                      matriz.e.vry(matriz.puntos[i][ixS][1])-matriz.
                            anchoRectangulo/2,
70                      matriz.anchoRectangulo, matriz.anchoRectangulo),
                            matriz.gruesoRectangulo)
71
72      #Dibujar las líneas:
73      if __dibujarLineas and actualizarCurvas:
74          for i in range(len(matriz.puntos)-1):
75              pygame.draw.line(pantalla, __colorLineas,
76                  matriz.e.vr(matriz.puntos[i][ixS]),
77                  matriz.e.vr(matriz.puntos[i+1][ixS]), __gruesoLineas)
78
79      #Redibujar el buffer
80      pygame.display.flip()
81
82
83  def guardarMatriz(matriz):
84      if len(sys.argv)>1:
85          matriz.guardarArchivo(sys.argv[1])
86          print("Archivo␣guardado")
87      else:
88          archivo = raw_input("Escriba␣el␣nombre␣del␣archivo␣a␣guardar␣(si␣
                lo␣omite,␣su␣trabajo␣no␣se␣guardará):␣")
89          if archivo:
90              matriz.guardarArchivo(archivo)
91              print("Archivo␣guardado")
92          else:
93              print("Archivo␣no␣guardado")
94
95  def iniciarAnimacion(matriz, pantalla):
96      pygame.display.set_caption("Animación␣en␣curso␣-␣Por␣favor␣espere")
97      listaPuntosAnimacion = []
98      for i in range(matriz.numNodos()):
99          listaPuntosAnimacion.append(matriz.trazador(i))
100
101     for j in range(len(listaPuntosAnimacion[0])):
102         #Borrar la pantalla
```

```
103         pantalla.fill(__colorFondo)
104 
105         #Dibujar los cuadrados:
106         for i in range(matriz.numNodos()):
107             pygame.draw.rect(pantalla, __colorCuadro,
108                 pygame.Rect(
109                     matriz.e.vrx(listaPuntosAnimacion[i][j][0])-matriz.
                        anchoRectangulo/2,
110                     matriz.e.vry(listaPuntosAnimacion[i][j][1])-matriz.
                        anchoRectangulo/2,
111                     matriz.anchoRectangulo, matriz.anchoRectangulo),
                        matriz.gruesoRectangulo)
112 
113         #Dibujar las líneas:
114         for i in range(matriz.numNodos()-1):
115             pygame.draw.line(pantalla, __colorLineas,
116                 matriz.e.vr(listaPuntosAnimacion[i][j]),
117                 matriz.e.vr(listaPuntosAnimacion[i+1][j]), __gruesoLineas
                    )
118 
119         #Redibujar el buffer
120         pygame.display.flip()
121 
122         #Hacer pausa
123         pygame.time.delay(PAUSA)
124 
125 
126 if __name__ == "__main__":
127     pygame.init()
128 
129     matriz = splines.MatrizSpline()
130     if len(sys.argv)>1:
131         if os.path.exists(sys.argv[1]):
132             matriz.cargarArchivo(sys.argv[1])
133         else:
134             __inicializarMatriz(matriz)
135     else:
136         __inicializarMatriz(matriz)
137     pantalla = pygame.display.set_mode(matriz.dimensiones(), pygame.
        RESIZABLE)
138     pygame.display.set_caption( \
139         "Editor␣de␣Secuencia␣de␣puntos␣-␣" + \
140         (len(sys.argv)>1 and sys.argv[1] or "SinNombre.bezier"))
141 
142     ratonPresionado = 0 #código del botón del ratón que ha sido
        presionado
143     punto = None
144     ixPunto = -1
145     ixSecuencia = 0
146 
```

```
147     dibujar(matriz, ixSecuencia, pantalla)
148     while True:
149         for evento in pygame.event.get():
150 
151             modificadores = pygame.key.get_mods()
152 
153             #Movimiento del ratón
154             if evento.type == pygame.MOUSEMOTION:
155                 #Arrastrando punto
156                 if ratonPresionado == BOTON_IZQ and punto:
157                     punto[0] = matriz.e.rvx(evento.pos[0])
158                     punto[1] = matriz.e.rvy(evento.pos[1])
159                     dibujar(matriz, ixSecuencia, pantalla)
160 
161                 #Desplazar el espacio virtual
162                 elif ratonPresionado == BOTON_IZQ and not punto:
163                     dxv = matriz.e.magnitudrvx(evento.rel[0])
164                     dyv = matriz.e.magnitudrvy(evento.rel[1])
165                     matriz.e.minxv -= dxv
166                     matriz.e.maxxv -= dxv
167                     matriz.e.minyv -= dyv
168                     matriz.e.maxyv -= dyv
169                     dibujar(matriz, ixSecuencia, pantalla)
170 
171             #Clic izquierdo - Identificar el punto para moverlo
172             elif evento.type == pygame.MOUSEBUTTONDOWN \
173                  and evento.button == BOTON_IZQ:
174                 ratonPresionado = evento.button
175                 xclic, yclic = evento.pos
176                 for i in range(len(matriz.puntos)):
177                     px,py = matriz.e.vr(matriz.puntos[i][ixSecuencia])
178                     if xclic > px-matriz.anchoRectangulo/2 and \
179                        yclic > py-matriz.anchoRectangulo/2 and \
180                        xclic < px+matriz.anchoRectangulo/2 and \
181                        yclic < py+matriz.anchoRectangulo/2 :
182                         punto = matriz.puntos[i][ixSecuencia]
183                         ixPunto = i
184                         dibujar(matriz, ixSecuencia, pantalla)
185                         break
186 
187             #Clic derecho - Agregar nuevo segmento
188             elif evento.type == pygame.MOUSEBUTTONDOWN \
189                  and not ratonPresionado \
190                  and evento.button == BOTON_DER:
191                 xclic, yclic = evento.pos
192                 for i in range(len(matriz.puntos)):
193                     px,py = matriz.e.vr(matriz.puntos[i][ixSecuencia])
194                     if xclic > px-matriz.anchoRectangulo/2 and \
195                        yclic > py-matriz.anchoRectangulo/2 and \
196                        xclic < px+matriz.anchoRectangulo/2 and \
```

```
197                         yclic < py+matriz.anchoRectangulo/2 :
198                          l = []
199                          if i == len(matriz.puntos)-1: #está al final
200                              for k in range(len(matriz.puntos[0])):
201                                  l.append([
202                                      matriz.puntos[i][k][0], #ponerlo
                                             sobre el anterior
203                                      matriz.puntos[i][k][1]
204                                  ])
205                          else:
206                              for k in range(len(matriz.puntos[0])):
207                                  l.append([
208                                      (matriz.puntos[i][k][0] + matriz.
                                             puntos[i+1][k][0])/2,
209                                      (matriz.puntos[i][k][1] + matriz.
                                             puntos[i+1][k][1])/2
210                                  ])
211                          matriz.puntos.insert(i+1, l)
212                          dibujar(matriz, ixSecuencia, pantalla)
213                          break
214
215                 #Ruedas del ratón
216                 elif evento.type == pygame.MOUSEBUTTONDOWN \
217                      and not ratonPresionado \
218                      and (evento.button == RUEDA_ABAJO or evento.button ==
                             RUEDA_ARRIBA):
219
220                     #Aumentar o disminuir el número de pasos
221                     if modificadores & pygame.KMOD_CTRL:
222                         if evento.button == RUEDA_ABAJO:
223                             np = matriz.numPasos()-1
224                             if np>1:
225                                 matriz.numPasos(-1)
226                         else:
227                             matriz.numPasos(1)
228                         print("Número de pasos para la animación: " + str(
                                matriz.numPasos()))
229
230                     #Aumentar o disminuir el tamaño de los cuadros
231                     elif modificadores & pygame.KMOD_SHIFT:
232                         if evento.button == RUEDA_ABAJO:
233                             matriz.anchoRectangulo = matriz.anchoRectangulo-5
234                             if matriz.anchoRectangulo<5: matriz.
                                    anchoRectangulo = 5
235                         else:
236                             matriz.anchoRectangulo = matriz.anchoRectangulo+5
237                         print("Ancho de los cuadrados: " + str(matriz.
                                anchoRectangulo))
238
239                     #Hacer acercamiento o alejamiento
```

```
240                 else:
241                     porcentajeDistanciaX = 0.05*(matriz.e.maxxv-matriz.e.
                            minxv)
242                     porcentajeDistanciaY = 0.05*(matriz.e.maxyv-matriz.e.
                            minyv)
243                     teclas = pygame.key.get_pressed()
244                     if evento.button == RUEDA_ABAJO: #acercamiento
245                         if not teclas[pygame.K_x] and not teclas[pygame.
                                K_y]:
246                             matriz.e.minxv += porcentajeDistanciaX
247                             matriz.e.maxxv -= porcentajeDistanciaX
248                             matriz.e.minyv += porcentajeDistanciaY
249                             matriz.e.maxyv -= porcentajeDistanciaY
250                         elif teclas[pygame.K_x]:
251                             matriz.e.minxv += porcentajeDistanciaX
252                             matriz.e.maxxv -= porcentajeDistanciaX
253                         elif teclas[pygame.K_y]:
254                             matriz.e.minyv += porcentajeDistanciaY
255                             matriz.e.maxyv -= porcentajeDistanciaY
256                     else: #alejamiento
257                         if not teclas[pygame.K_x] and not teclas[pygame.
                                K_y]:
258                             matriz.e.minxv -= porcentajeDistanciaX
259                             matriz.e.maxxv += porcentajeDistanciaX
260                             matriz.e.minyv -= porcentajeDistanciaY
261                             matriz.e.maxyv += porcentajeDistanciaY
262                         elif teclas[pygame.K_x]:
263                             matriz.e.minxv -= porcentajeDistanciaX
264                             matriz.e.maxxv += porcentajeDistanciaX
265                         elif teclas[pygame.K_y]:
266                             matriz.e.minyv -= porcentajeDistanciaY
267                             matriz.e.maxyv += porcentajeDistanciaY
268                     dibujar(matriz, ixSecuencia, pantalla)
269 
270                 #Eliminar un punto
271                 elif evento.type == pygame.MOUSEBUTTONDOWN \
272                      and not ratonPresionado \
273                      and evento.button == BOTON_CEN:
274                     xclic, yclic = evento.pos
275                     for i in range(len(matriz.puntos)):
276                         px,py = matriz.e.vr(matriz.puntos[i][ixSecuencia])
277                         if xclic > px-matriz.anchoRectangulo/2 and \
278                            yclic > py-matriz.anchoRectangulo/2 and \
279                            xclic < px+matriz.anchoRectangulo/2 and \
280                            yclic < py+matriz.anchoRectangulo/2 :
281                             matriz.puntos.pop(i)
282                             dibujar(matriz, ixSecuencia, pantalla)
283                             break
284 
285                 #Detener el movimiento
```

```
286             elif evento.type == pygame.MOUSEBUTTONUP:
287                 if evento.button == ratonPresionado == BOTON_IZQ:
288                     ratonPresionado = 0
289                     punto = None
290                     ixPunto = -1
291 
292             #Cuando se redimensiona la ventana
293             elif evento.type == pygame.VIDEORESIZE:
294                 matriz.cambiarDimensiones(evento.size)
295                 pantalla = pygame.display.set_mode(matriz.dimensiones(),
                        pygame.RESIZABLE)
296                 dibujar(matriz, ixSecuencia, pantalla)
297 
298             #Otras acciones
299             elif evento.type == pygame.KEYDOWN and \
300                  (modificadores & pygame.KMOD_CTRL):
301                 if evento.key == pygame.K_LEFT:
302                     if ixSecuencia > 0:
303                         ixSecuencia -= 1
304                         print("Cuadro␣{0}/{1}".format(ixSecuencia+1, len(
                                matriz.puntos[0])))
305                         dibujar(matriz, ixSecuencia, pantalla)
306                 elif evento.key == pygame.K_RIGHT:
307                     if ixSecuencia == len(matriz.puntos[0])-1:
308                         for l in matriz.puntos:
309                             l.append([ l[-1][0], l[-1][1] ])
310                     ixSecuencia += 1
311                     print("Cuadro␣{0}/{1}".format(ixSecuencia+1, len(
                            matriz.puntos[0])))
312                     dibujar(matriz, ixSecuencia, pantalla)
313                 elif evento.key == pygame.K_l:
314                     __dibujarLineas = not __dibujarLineas
315                     if __dibujarLineas:
316                         print("Dibujo␣de␣líneas␣encendido")
317                     else:
318                         print("Dibujo␣de␣líneas␣apagado")
319                     dibujar(matriz, ixSecuencia, pantalla)
320 
321                 elif evento.key == pygame.K_c:
322                     __dibujarCuadros = not __dibujarCuadros
323                     if __dibujarCuadros:
324                         print("Dibujo␣de␣cuadros␣encendido")
325                     else:
326                         print("Dibujo␣de␣cuadros␣apagado")
327                     dibujar(matriz, ixSecuencia, pantalla)
328 
329                 elif evento.key == pygame.K_g:
330                     guardarMatriz(matriz)
331 
332                 #Aumentar o disminuír el grueso de los cuadros
```

```
333                 elif evento.key == pygame.K_DOWN:
334                     matriz.gruesoRectangulo = matriz.gruesoRectangulo-1
335                     if matriz.gruesoRectangulo<1: matriz.gruesoRectangulo
                            = 1
336                     print("Grueso de los cuadrados: " + str(matriz.
                            gruesoRectangulo))
337                     dibujar(matriz, ixSecuencia, pantalla)
338                 elif evento.key == pygame.K_UP:
339                     matriz.gruesoRectangulo = matriz.gruesoRectangulo+1
340                     print("Grueso de los cuadrados: " + str(matriz.
                            gruesoRectangulo))
341                     dibujar(matriz, ixSecuencia, pantalla)
342 
343             elif evento.type == pygame.KEYDOWN and \
344                  evento.key == pygame.K_SPACE:
345                 titulo = pygame.display.get_caption()[0]
346                 iniciarAnimacion(matriz, pantalla)
347                 pygame.display.set_caption(titulo)
348                 dibujar(matriz, ixSecuencia, pantalla)
349 
350             elif evento.type == pygame.KEYDOWN and \
351                  evento.key == pygame.K_F1:
352                     print("Ayuda...") #Pendiente...
353 
354             elif evento.type == pygame.QUIT:
355                 pygame.display.quit() #Cierra la ventana y apaga el
                        subsistema de video
356                 guardarMatriz(matriz)
357                 sys.exit()
```

La biblioteca principal, que contiene la implementación de algunos algoritmos es esta:

Listing 9.6: Biblioteca para Curvas Splines Bidimensionales en Python

```
1  # coding: utf-8
2  """ c09/trazadores_py/splines.py - Modulo para procesar un arreglo Spline
       .
3  
4  Los datos se guardan en archivos xml.
5  """
6  
7  import xml.dom.minidom
8  
9  __H     = 0
10 __ALFA  = 1
11 __L     = 2
12 __MIU   = 3
13 __Z     = 4
14 
15 A = 0
```

```
B = 1
C = 2
D = 3

def coeficientesTrazador(independiente, dependiente):
    '''Devuelve la matriz de coeficientes
    de los 'n' trazadores interpolantes cúbicos paramétricos
    para la secuencia de nodos descritos en
    la matriz {independiente x dependiente}.

    __La salida tiene la forma:
    [[a1, a2, ..., an],
     [b1, b2, ..., bn],
     [c1, c2, ..., cn],
     [d1, d2, ..., dn]]
    donde 'n+1' es el número de nodos que representan
    los dos vectores, dependiente e independiente.
    '''
    if len(independiente) != len(dependiente):
        raise Exception("__Los arreglos deben ser de igual longitud")
    numNodos = len(independiente)
    numTrazadores = len(independiente) - 1
    def listaCeros():
        l = []
        for i in range(numNodos):
            l.append(0.0)
        return l

    coef = [ 0, listaCeros(), listaCeros(), listaCeros()]
    tmp = [ listaCeros(), listaCeros(), listaCeros(), listaCeros(),
        listaCeros()]

    #Cálculo de los trazadores
    n = numTrazadores
    coef[A] = dependiente[:] #Copiar todo el arreglo

    #paso 2...
    for i in range(n):
        tmp[__H][i] = independiente[i+1] - independiente[i]

    #paso 3...
    for i in range(1, n):
        tmp[__ALFA][i] = 3 * (
            (coef[A][i+1] - coef[A][i]) / tmp[__H][i] - \
            (coef[A][i] - coef[A][i-1]) / tmp[__H][i-1]
        )

    #paso 4...
    tmp[__L][0] = 1
    tmp[__MIU][0] = 0
```

```
65      tmp[__Z][0] = 0
66
67      #paso 5...
68      for i in range(1, n):
69          tmp[__L][i] = 2 * (independiente[i+1] - independiente[i-1]) - tmp
                [__H][i-1]*tmp[__MIU][i-1]
70          tmp[__MIU][i] = tmp[__H][i] / tmp[__L][i]
71          tmp[__Z][i] = (tmp[__ALFA][i] - tmp[__H][i-1]*tmp[__Z][i-1]) /
                tmp[__L][i]
72
73      #paso 6...
74      coef[C][n] = 0.0
75
76      #paso 7...
77      for i in range(n-1, -1, -1):
78          coef[C][i] = tmp[__Z][i] - tmp[__MIU][i] * coef[C][i+1]
79          coef[B][i] = (coef[A][i+1] - coef[A][i]) / tmp[__H][i] \
80              - tmp[__H][i] * (coef[C][i+1] + 2*coef[C][i]) / 3
81          coef[D][i] = (coef[C][i+1] - coef[C][i]) / (3*tmp[__H][i])
82
83      return coef
84
85  class MatrizSpline():
86      '''Esta␣clase␣mantiene␣una␣matriz␣de␣puntos␣para␣modelar␣una␣matriz␣
            Spline.
87
88  ␣␣␣␣Cada␣lista␣principal␣de␣la␣matriz,␣es␣una␣secuencia␣independiente.
89  ␣␣␣␣'''
90
91      def __init__(self, escala=None):
92          if escala:
93              self.puntos = [
94                  [ [escala.minxv, escala.minyv] ],
95                  [ [(escala.minxv+escala.maxxv)/2.0, (escala.minyv+escala.
                        maxyv)/2.0] ],
96                  [ [escala.maxxv, escala.maxyv] ]
97              ]
98          else:
99              self.puntos = [
100                 [ [0,0] ],[ [0,0] ],[ [0,0] ]
101             ]
102         self.anchoRectangulo = 20
103         self.gruesoRectangulo = 1
104
105         self.e = escala
106         self.__numPasos = 25
107
108         self.__t = None
109
110     def numNodos(self):
```

```
111             return len(self.puntos)
112         def numCuadros(self):
113             return len(self.puntos[0])
114 
115         def numPasos(self, valorIncremento=0):
116             '''Permite␣aumentar␣o␣disminuír␣el␣número␣de␣líneas
117 ␣␣␣␣␣␣␣␣que␣conformarán␣una␣secuencia␣de␣segmentos␣de␣Bézier
118 ␣␣␣␣␣␣␣␣'''
119             if valorIncremento:
120                 self.__numPasos += valorIncremento
121                 if self.__numPasos == 0:
122                     self.__numPasos -= valorIncremento
123             return self.__numPasos
124 
125         def dimensiones(self):
126             """Devuelve␣las␣dimensiones␣de␣la␣pantalla␣donde␣debe␣mostrarse
127 ␣␣␣␣␣␣␣␣la␣serie␣de␣puntos"""
128             return self.e.maxxr, self.e.maxyr
129 
130         def cambiarDimensiones(self, tam):
131             "cambiar␣las␣dimensiones␣de␣la␣escala␣real"
132             self.e.maxxr, self.e.maxyr = tam
133 
134         def ponerEscala(self, (ANCHO, ALTO), (minx, miny), (maxx, maxy)):
135             """Configura␣la␣escala␣de␣visualización␣de␣esta␣matriz
136 ␣␣␣␣␣␣␣␣"""
137             self.e = Escala( \
138                 (ANCHO, ALTO), \
139                 (minx, miny), (maxx, maxy))
140 
141         def trazador(self, i):
142             '''Devuelve␣una␣lista␣de␣pares␣ordenados␣del␣tipo␣[x,y].
143 ␣␣␣␣␣␣␣␣La␣lista␣tiene␣self.__numPasos+1␣pares
144 ␣␣␣␣␣␣␣␣que␣interpolan␣a␣la␣i-ésima␣secuencia␣de␣esta␣matriz
145 ␣␣␣␣␣␣␣␣de␣Splines.
146 ␣␣␣␣␣␣␣␣'''
147             numTrazadores = self.numCuadros()-1
148             l = []
149             self.__t = [float(x)/(self.numCuadros()-1) for x in range(self.
                    numCuadros())]
150             coefX = coeficientesTrazador(self.__t, \
151                 [p[0] for p in self.puntos[i]]
152             )
153             coefY = coeficientesTrazador(self.__t, \
154                 [p[1] for p in self.puntos[i]]
155             )
156 
157             ix_t = 1
158             for t in [float(x)/self.__numPasos for x in range(self.__numPasos
                    +1)]:
```

```
159             while t > self.__t[ix_t]:
160                 ix_t += 1
161             ix_t -= 1
162 
163             tmpT = t - self.__t[ix_t]
164             l.append( [ \
165             coefX[A][ix_t] + tmpT * ( coefX[B][ix_t] + tmpT * ( coefX[C][
                    ix_t] + tmpT * coefX[D][ix_t] ) ), \
166             coefY[A][ix_t] + tmpT * ( coefY[B][ix_t] + tmpT * ( coefY[C][
                    ix_t] + tmpT * coefY[D][ix_t] ) )
167             ] )
168         #for h in l:
169         #    print(h)
170         return l
171 
172     def cargarArchivo(self, nombreArchivo):
173         documentoxml = xml.dom.minidom.parse(nombreArchivo)
174 
175         curvasxml = documentoxml.getElementsByTagName("curvas")[0]
176         rectangulosxml = documentoxml.getElementsByTagName("rectangulos")
                [0]
177         escalaxml = documentoxml.getElementsByTagName("escala")[0]
178         secuenciasxml = documentoxml.getElementsByTagName("secuencias")
                [0]
179 
180         self.__numPasos = int(curvasxml.getAttribute("pasos"))
181         self.__t = None
182 
183         self.anchoRectangulo = int(rectangulosxml.getAttribute("ancho"))
184         self.gruesoRectangulo = int(rectangulosxml.getAttribute("grueso")
                )
185 
186         self.e = Escala( \
187             (int(escalaxml.getAttribute("maxxr")), int(escalaxml.
                    getAttribute("maxyr"))), \
188             (float(escalaxml.getAttribute("minxv")), float(escalaxml.
                    getAttribute("minyv"))), \
189             (float(escalaxml.getAttribute("maxxv")), float(escalaxml.
                    getAttribute("maxyv"))))
190 
191         self.puntos = []
192         for nodos_xml in secuenciasxml.getElementsByTagName("nodos"):
193             l = []
194             for nodoxml in nodos_xml.getElementsByTagName("nodo"):
195                 l.append([float(nodoxml.getAttribute("x")), \
196                           float(nodoxml.getAttribute("y"))])
197             self.puntos.append(l)
198 
199     def guardarArchivo(self, nombreArchivo):
200         implementacionxml = xml.dom.minidom.getDOMImplementation()
```

```
201         documentoxml = implementacionxml.createDocument(None, "splines",
                None)
202         raiz = documentoxml.documentElement
203 
204         curvas = documentoxml.createElement("curvas")
205         curvas.setAttribute("pasos", str(self.__numPasos))
206         raiz.appendChild(curvas)
207 
208         rectangulos = documentoxml.createElement("rectangulos")
209         rectangulos.setAttribute("ancho", str(self.anchoRectangulo))
210         rectangulos.setAttribute("grueso", str(self.gruesoRectangulo))
211         raiz.appendChild(rectangulos)
212 
213         escala = documentoxml.createElement("escala")
214         escala.setAttribute("maxxr", str(self.e.maxxr))
215         escala.setAttribute("maxyr", str(self.e.maxyr))
216 
217         escala.setAttribute("minxv", str(self.e.minxv))
218         escala.setAttribute("minyv", str(self.e.minyv))
219         escala.setAttribute("maxxv", str(self.e.maxxv))
220         escala.setAttribute("maxyv", str(self.e.maxyv))
221         raiz.appendChild(escala)
222 
223         secuenciasxml = documentoxml.createElement("secuencias")
224         for lista in self.puntos:
225             nodos_xml = documentoxml.createElement("nodos")
226             for p in lista:
227                 nodoxml = documentoxml.createElement("nodo")
228                 nodoxml.setAttribute("x", str(p[0]))
229                 nodoxml.setAttribute("y", str(p[1]))
230                 nodos_xml.appendChild(nodoxml)
231             secuenciasxml.appendChild(nodos_xml)
232         raiz.appendChild(secuenciasxml)
233 
234         f = open(nombreArchivo, 'w')
235         f.write(documentoxml.toprettyxml())
236         f.close()
237 
238 
239 
240 class Escala():
241     """Clase␣de␣Escalas␣bidimensionales␣de␣ventana␣completa.
242 
243 ␣␣␣␣Esta␣clase␣permite␣modelar␣un␣espacio␣bidimensional␣cuadrado
244 ␣␣␣␣con␣dos␣escalas␣diferentes:␣una␣Real␣y␣una␣Virtual.
245 
246 ␣␣␣␣La␣escala␣Real␣está␣medida␣en␣pixeles␣en␣el␣dominio␣de␣los␣enteros.
247 ␣␣␣␣Su␣origen␣se␣encuentra␣en␣la␣esquina␣superior␣izquierda␣y
248 ␣␣␣␣en␣la␣esquina␣inferior␣derecha␣se␣encuentra␣la␣coordenada␣(ANCHO-1,
        ALTO-1).
```

```
249 ␣␣␣␣Se␣asume␣que␣ANCHO␣y␣ALTO␣son␣positivos.
250
251 ␣␣␣␣La␣escala␣Virtual␣tiene␣unidades␣arbitrarias␣en␣el␣dominio␣de␣los␣
        reales.
252 ␣␣␣␣En␣la␣esquina␣inferior␣izquierda␣se␣encuentra␣la␣coordenada␣[minx,
        miny]␣y
253 ␣␣␣␣en␣la␣esquina␣superior␣derecha␣[maxx,␣maxy].
254 ␣␣␣␣Se␣asume␣que␣minx␣<␣maxx␣y␣que␣miny␣<␣maxy.
255 ␣␣␣␣"""
256     def __init__(self, (ANCHO, ALTO), (minx, miny), (maxx, maxy)):
257         #self.minxr = 0 #Siempre es cero
258         #self.minyr = 0 #Siempre es cero
259
260         self.maxxr = ANCHO
261         self.maxyr = ALTO
262
263         self.minxv = minx
264         self.minyv = miny
265
266         self.maxxv = maxx
267         self.maxyv = maxy
268
269     def rvx(self, xr):
270         '''Convierte␣un␣valor␣x␣Real␣en␣su␣correspondiente␣Virtual'''
271         return (self.maxxv-self.minxv)*float(xr)/self.maxxr + self.minxv
272     def rvy(self, yr):
273         '''Convierte␣un␣valor␣y␣Real␣en␣su␣correspondiente␣Virtual'''
274         return (self.minyv-self.maxyv)*float(yr)/self.maxyr + self.maxyv
275
276     def vrx(self, xv):
277         '''Convierte␣un␣valor␣x␣Virtual␣en␣su␣correspondiente␣Real'''
278         return int( (xv - self.minxv)*self.maxxr/(self.maxxv-self.minxv)
                )
279     def vry(self, yv):
280         '''Convierte␣un␣valor␣y␣Virtual␣en␣su␣correspondiente␣Real'''
281         return int( (yv - self.maxyv)*self.maxyr/(self.minyv-self.maxyv)
                )
282
283     def rv(self, (xr, yr)):
284         '''Convierte␣un␣par␣(x,y)␣Real␣en␣su␣correspondiente␣Virtual'''
285         return self.rvx(xr), self.rvy(yr)
286     def vr(self, (xv, yv)):
287         '''Convierte␣un␣par␣(x,y)␣Virtual␣en␣su␣correspondiente␣Real'''
288         return self.vrx(xv), self.vry(yv)
289
290     def magnitudvrx(self, deltaxv):
291         '''Convierte␣una␣'distancia'␣deltaxv␣en␣la␣escala␣Virtual
292 ␣␣␣␣␣␣␣␣a␣su␣correspondiente␣'distancia'␣en␣la␣escala␣Real'''
293         return deltaxv * self.maxxr / (self.maxxv - self.minxv)
294     def magnitudvry(self, deltayv):
```

```
295         '''Convierte una 'distancia' deltayv en la escala Virtual
296 ␣␣␣␣␣␣␣␣a su correspondiente 'distancia' en la escala Real'''
297         return deltayv * self.maxyr / (self.minyv - self.maxyv)
298 
299     def magnitudrvx(self, deltaxr):
300         '''Convierte una 'distancia' deltaxr en la escala Real
301 ␣␣␣␣␣␣␣␣a su correspondiente 'distancia' en la escala Virtual'''
302         return deltaxr * (self.maxxv - self.minxv) / self.maxxr
303     def magnitudrvy(self, deltayr):
304         '''Convierte una 'distancia' deltayr en la escala Real
305 ␣␣␣␣␣␣␣␣a su correspondiente 'distancia' en la escala Virtual'''
306         return deltayr * (self.minyv - self.maxyv) / self.maxyr
```

9.4. Curvas de *Bézier*

Las curvas de *Bézier* fueron nombradas así en honor de *Pierre Bézier*[3], quien las utilizó para el diseño de carrocerías de automóviles en 1962 en la empresa Rénault.

9.4.1. Descripción geométrica

Estas curvas están determinadas típicamente por cuatro puntos ordenados, de los cuales el primero y el último determinan el inicio y fin de la curva, y los otros dos describen los vectores tangentes inicial y final que controlan la trayectoria de la curva entre los puntos inicial y final. Es importante recalcar que estas curvas **no sirven para interpolar**, ya que no pasan por todos los puntos de control.

9.4.2. Descripción matemática

Dados cuatro puntos P_0, P_1, P_2, P_3 (que pueden ser unidimensionales, bidimensionales, tridimensionales, etc.), llamados puntos de control, se define la curva de Bézier de la siguiente manera:

$$\vec{B}(t) = (1-t)^3\vec{P}_0 + 3t(1-t)^2\vec{P}_1 + 3t^2(1-t)\vec{P}_2 + t^3\vec{P}_3 \ , \ 0 \leqslant t \leqslant 1 \qquad (9.1)$$

es decir:

$x(t) = (1-t)^3x_0 + 3t(1-t)^2x_1 + 3t^2(1-t)x_2 + t^3x_3 \ , \ 0 \leqslant t \leqslant 1,$

$y(t) = (1-t)^3y_0 + 3t(1-t)^2y_1 + 3t^2(1-t)y_2 + t^3y_3 \ , \ 0 \leqslant t \leqslant 1$ y además,

$z(t) = (1-t)^3z_0 + 3t(1-t)^2z_1 + 3t^2(1-t)z_2 + t^3z_3 \ , \ 0 \leqslant t \leqslant 1$ (si es que aplica), etc...

[3] aunque fueron inventadas por *Paul de Casteljau* en 1959

9.4.3. Polinomios de Bernstein

La teoría básica para calcular las curvas de Bézier se basa en la idea que *cada punto de la curva es un promedio ponderado de los puntos de control*. Esto se consigue con coeficientes especialmente diseñados para eso. Estos coeficientes determinan el polinomio que forma la ecuación de Bézier, y son conocidos como **Polinomios de Bernstein**, y se definen así:

$$b(i,n,t) = \binom{n}{i}(1-t)^{n-i}t^i \tag{9.2}$$

donde $\binom{n}{i} = \frac{n!}{i!(n-i)!}$, $0 \leqslant i \leqslant n \in \mathbb{N},\ 0 \leqslant t \leqslant 1$.

n es el grado del polinomio (para las curvas de Bézier normales, es 3. i es el índice del polinomio. Así, los polinomios de Bernstein de grado 3 son:

$$\begin{aligned} b(0,3,t) &= (1-t)^3 \\ b(1,3,t) &= 3t(1-t)^2 \\ b(2,3,t) &= 3t^2(1-t) \\ b(3,3,t) &= t^3 \end{aligned}$$

Sus gráficas pueden apreciarse en la figura 9.4 en la página siguiente. Pero su característica más importante, servir para generar un promedio ponderado puede apreciarse en la figura 9.5 en la página 213: El hecho de que su suma siempre resulte en 1 es lo que permite usarlos para cálculos de promedios ponderados uniformemente espaciados.

9.4.4. Generalización de curvas de *Bézier* de grado n

En función de los Polinomios de Bernstein, se definen las curvas de Bézier de n-grado como:

$$\vec{B}_n(t) = \sum_{i=0}^{n} b(i,n,t)\vec{P}_i \tag{9.3}$$

De modo que podemos enunciar las siguientes curvas de Bézier de primero, segundo, tercer, cuarto y quinto grado (los coeficientes numéricos responden al Triángulo de Pascal):

$B_1(t) = (1-t)P_0 + tP_1,\ 0 \leqslant t \leqslant 1$

$B_2(t) = (1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2,\ 0 \leqslant t \leqslant 1$

$B_3(t) = (1-t)^3P_0 + 3t(1-t)^2P_1 + 3t^2(1-t)P_2 + t^3P_3,\ 0 \leqslant t \leqslant 1$

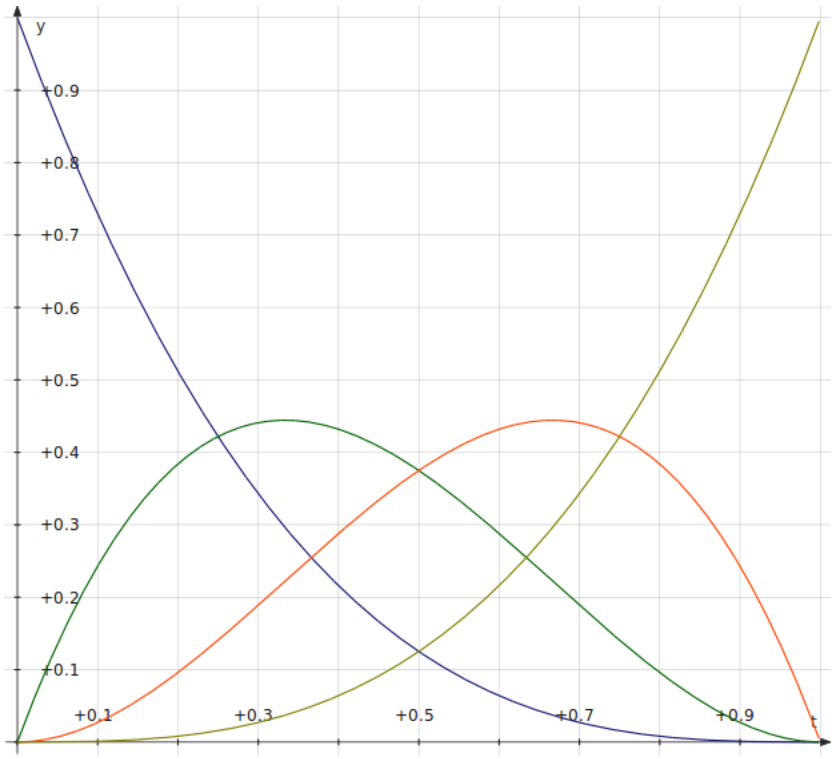


Figura 9.4: Gráfica de los Polinomios de Bernstein de grado 3

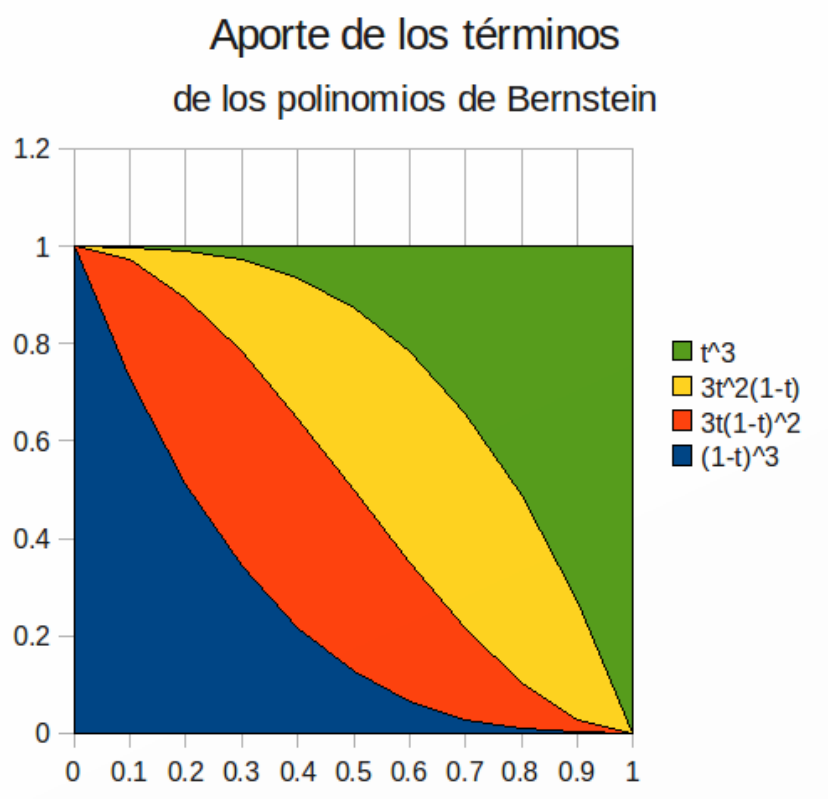


Figura 9.5: Aporte de cada uno de los polinomios de Bernstein de grado 3

$B_4(t) = (1-t)^4 P_0 + 4t(1-t)^3 P_1 + 6t^2(1-t)^2 P_2 + 4t^3(1-t)P_3 + t^4 P_4,\ 0 \leqslant t \leqslant 1$

$B_5(t) = (1-t)^5 P_0 + 5t(1-t)^4 P_1 + 10t^2(1-t)^3 P_2 + 10t^3(1-t)^2 P_3 + 5t^4(1-t)P_4 + t^5 P_5,\ 0 \leqslant t \leqslant 1$

Usar curvas de Bézier de grado muy alto no reportan significativas ventajas respecto de los de tercer grado desde el punto de vista de la eficiencia de los algoritmos necesarios para manipularlas, pero presentan un comportamiento muy interesante, como puede verse en la sección Constructing Bézier Curves de [Wikipedia-Bézier Curve].

Debido a que en general sólo se usan curvas de Bézier de tercer grado, el nombre de estas se ha generalizado, de tal forma que cuando uno se refiere sin más, a *Curvas de Bézier*, se refiere implícitamente a Curvas de Bézier de Tercer Grado (a menos que se especifique lo contrario, por supuesto).

9.4.5. Unión de segmentos de Bézier

La gran utilidad de las curvas de Bézier es que son rápidas de calcular, pero tienen la limitante que tienen un número fijo de puntos de control. A este grupo de puntos se les llama *Segmentos de Curva de Bézier* o simplemente *Segmentos de Bézier*. El problema del tamaño fijo de los segmentos de Bézier, se resuelve ensamblando una serie de segmentos para construir una secuencia de segmentos de Bézier. Obviamente es de nuestro interés, construir una serie de segmentos de Bézier que no sólo sea continua, sino también suave. Para lograr esto, analicemos lo siguiente:

- Para que haya continuidad G^0 en la unión de dos curvas de Bézier, P y Q, es necesario que el último punto de P coincida con el primero de Q:

$$P_3 = Q_0$$

- Para que haya continuidad G^1, es necesario que haya continuidad G^0, y es necesario que el último vector tangente de P sea linealmente dependiente del primero de Q y además tener la misma dirección:

$$\overrightarrow{P_2P_3} = k\overrightarrow{Q_0Q_1},\ k > 0$$

- Para que haya continuidad C^1 (muy deseable para la mayoría de las aplicaciones), es necesario además de continuidad G^1, que el último vector tangente de P sea igual al primero de Q:

$$\overrightarrow{P_2P_3} = \overrightarrow{Q_0Q_1}$$

Para poder, entonces, tener una secuencia de segmentos de Bézier (de tercer grado) que generen una curva suave, es necesario garantizar que se cumplan estas tres características descritas.

9.4.6. Ejemplos de implementación

A continuación se presentan dos aplicaciones sencillas, ilustrativas de manipulación de curvas de Bézier que se incluyen en el material adjunto a este libro: `bezier1` y `bezier2`. El primero permite manipular un solo segmento de Bézier, para familiarizarse con su naturaleza. El segundo permite manipular tres segmentos de Bézier encadenados.

bezier1

Los siguientes archivos presentan una implementación de segmentos independientes de Bézier:

Listing 9.7: Archivo de cabecera de operaciones de segmentos de Bézier

```
1  // /c09/bezier/bezier1.h
2  #include <stdio.h>
3
4  #define BEZIER_COD_EXITO 0
5  #define BEZIER_MSG_EXITO "Éxito\n"
6  #define BEZIER_COD_ERROR_PARAMETROS -1
7  #define BEZIER_MSG_ERROR_PARAMETROS "Error␣de␣parámetro(s)\n"
8  #define BEZIER_COD_ERROR_DOMINIO -2
9  #define BEZIER_MSG_ERROR_DOMINIO "Error␣de␣dominio\n"
10 #define BEZIER_COD_ERROR_MEMORIA -3
11 #define BEZIER_MSG_ERROR_MEMORIA "Error␣de␣solicitud␣de␣memoria␣dinámica\
       n"
12
13 /*
14     Contiene los cuatro puntos de control
15     para los segmentos de Bezier.
16
17     El cálculo de lo puntos internos del segmento es el siguiente:
18
19     B(t) = Bx(t)·i + By(t)·j Bz(t)·k,
20         Bx(t) = (1-t)^3 * x[0] + 3t(1-t)^2 * x[1] + 3t^2(1-t) * x[2] + t
               ^3 * x[3]
21         By(t) = (1-t)^3 * y[0] + 3t(1-t)^2 * y[1] + 3t^2(1-t) * y[2] + t
               ^3 * y[3]
22         Bz(t) = (1-t)^3 * z[0] + 3t(1-t)^2 * z[1] + 3t^2(1-t) * z[2] + t
               ^3 * z[3]
23     para 0 <= t <= 1
24 */
25 typedef struct{
26     //coordenadas de los cuatro nodos
27     double x[4];
28     double y[4];
29     double z[4];
30 } segmentoBezier;
```

```
31
32 /*
33      Dado un valor del parámetro 't',
34      devuelve los valores del punto calculado
35      del segmento de Bézier.
36      Asume que el segmento es tridimensional.
37 */
38 int BEZIER_calcular(segmentoBezier *sb, double t, double *x, double *y,
       double *z);
39
40 /*
41      Dado un valor del parámetro 't',
42      devuelve los valores del punto calculado
43      del segmento de Bézier.
44      Esta versión asume que los puntos son coplanares en xy.
45 */
46
47 int BEZIER_calcular2d(segmentoBezier *sb, double t, double *x, double *y)
       ;
48
49 /*
50      Imprime en la salida especificada los valores
51      almacenados en la estructura.
52 */
53 void BEZIER_imprimir(segmentoBezier *sb, FILE *f);
```

Listing 9.8: Código fuente de funciones de segmentos de Bézier

```
1  // /c09/bezier/bezier1.c
2  #include "bezier1.h"
3
4  int BEZIER_calcular(segmentoBezier *sb, double t, double *x, double *y,
       double *z){
5      if(sb == NULL || x == NULL || y==NULL || z == NULL)
6          return BEZIER_COD_ERROR_PARAMETROS;
7      if(t<0.0 || t>1.0)
8          return BEZIER_COD_ERROR_DOMINIO;
9
10     double  _1_t1 = 1-t;
11     double  _1_t2 = _1_t1 * _1_t1;
12     double  _1_t3 = _1_t2 * _1_t1;
13     double  t2 = t*t;
14     double  t3 = t2*t;
15
16     *x = _1_t3 * sb->x[0] + 3*t*_1_t2 * sb->x[1] + 3*t2*_1_t1 * sb->x[2]
           + t3 * sb->x[3];
17     *y = _1_t3 * sb->y[0] + 3*t*_1_t2 * sb->y[1] + 3*t2*_1_t1 * sb->y[2]
           + t3 * sb->y[3];
18     *z = _1_t3 * sb->z[0] + 3*t*_1_t2 * sb->z[1] + 3*t2*_1_t1 * sb->z[2]
           + t3 * sb->z[3];
```

```
19      return BEZIER_COD_EXITO;
20  }
21
22  int BEZIER_calcular2d(segmentoBezier *sb, double t, double *x, double *y)
        {
23      if(sb == NULL || x == NULL || y == NULL)
24          return BEZIER_COD_ERROR_PARAMETROS;
25      if(t<0.0 || t>1.0)
26          return BEZIER_COD_ERROR_DOMINIO;
27
28      double  _1_t1 = 1-t;
29      double  _1_t2 = _1_t1 * _1_t1;
30      double  _1_t3 = _1_t2 * _1_t1;
31      double  t2 = t*t;
32      double  t3 = t2*t;
33
34      *x = _1_t3 * sb->x[0] + 3*t*_1_t2 * sb->x[1] + 3*t2*_1_t1 * sb->x[2]
            + t3 * sb->x[3];
35      *y = _1_t3 * sb->y[0] + 3*t*_1_t2 * sb->y[1] + 3*t2*_1_t1 * sb->y[2]
            + t3 * sb->y[3];
36      return BEZIER_COD_EXITO;
37  }
38
39  void BEZIER_imprimir(segmentoBezier *sb, FILE *f){
40      int i;
41      fprintf(f,"\nPuntos␣de␣control␣de␣Bézier:\n");
42      fprintf(f,"i\tx_i\ty_i\tz_i\n");
43      for(i=0; i<4; i++)
44          fprintf(f,"%d\t%3.2g\t%3.2g\t%3.2g\n",
45          i+1, sb->x[i], sb->y[i], sb->z[i]);
46  }
```

El siguiente código es una sencilla aplicación que usa el código de los archivos anteriores para permitirle al usuario, como se mencionó antes, manipular un solo segmento de Bézier:

Listing 9.9: Programa principal de manipulación de un segmento de Bézier

```
1   // /c09/bezier/main1.c
2   #include <SDL/SDL.h>
3   #include "bezier1.h"
4   #include <SDL/SDL_gfxPrimitives.h>
5
6   #define ANCHO 640
7   #define ALTO  480
8
9   #define ANCHO_RECTANGULO 15
10
11  #define XMIN  0.0
12  #define XMAX 15.0
```

```
13 #define YMIN  0.0
14 #define YMAX 15.0
15 #define TPASO 0.01
16
17 #define TAM 4
18
19 static inline int x_real(double x_virtual){
20     return (int) (ANCHO * x_virtual / XMAX);
21 }
22 static inline int y_real(double y_virtual){
23     return (int) (ALTO * (1.0 - y_virtual / YMAX));
24 }
25 static inline double x_virtual(int x_real){
26     return (XMAX * x_real) / ANCHO;
27 }
28 static inline double y_virtual(int y_real){
29     return YMAX * (1.0 - y_real / (double)ALTO);
30 }
31
32 Uint32 gfxColor(Uint8 r, Uint8 g, Uint8 b){
33     return
34         r << 24 |
35         g << 16 |
36         b << 8  |
37         255;     //este valor es la opacidad del color
38                  //y debe ser máxima para que sea sólido
39 }
40
41 //Variables globales
42 double x[TAM] = {1.3, 3.0, 6.0, 13.0};
43 double y[TAM] = {1.3, 5.0, 7.4, 14.2};
44 segmentoBezier sb;
45 int i, j, k, l;
46 double tvar, xvar, yvar;
47 Uint32 color_fondo, color1, color2, color3;
48
49 void dibujo(SDL_Surface *pantalla){
50     i=j=k=l=0;
51     SDL_Rect rect;
52
53     //Borra la pantalla
54     SDL_FillRect(pantalla, NULL, color_fondo);
55
56     for(i=1; i<4; i++){
57         lineColor(pantalla,
58             x_real(sb.x[i-1]), y_real(sb.y[i-1]),
59             x_real(sb.x[i])  , y_real(sb.y[i])  , color3);
60     }
61
62     tvar = 0.0;
```

```
63      BEZIER_calcular2d(&sb, tvar, &xvar, &yvar);
64      i=x_real(xvar);
65      j=y_real(yvar);
66      for(tvar = TPASO; tvar <= 1.0; tvar+=TPASO){
67          if(!BEZIER_calcular2d(&sb, tvar, &xvar, &yvar)){
68              lineColor(pantalla, i, j, k=x_real(xvar), l=y_real(yvar),
                    color1);
69              i=k; j=l;
70          }
71      }
72      BEZIER_calcular2d(&sb, 1.0, &xvar, &yvar);
73      lineColor(pantalla, i, j, k=x_real(xvar), l=y_real(yvar), color1);
74
75      //dibujar los rectángulos de control
76      rect.w = rect.h = ANCHO_RECTANGULO;
77      for(i=0; i<4; i++){
78          rect.x = x_real(sb.x[i])-ANCHO_RECTANGULO/2;
79          rect.y = y_real(sb.y[i])-ANCHO_RECTANGULO/2;
80          rectangleColor(pantalla, rect.x, rect.y, rect.x + rect.w, rect.y
                + rect.h, color2);
81      }
82
83      //vuelca el buffer en la pantalla:
84      SDL_Flip (pantalla);
85  }
86
87  int main(int argc, char *argv[]){
88      SDL_Surface *pantalla = NULL;
89      SDL_Event evento;
90      int profundidad_color;
91      const SDL_VideoInfo *info;
92      Uint32 color;
93      int i;
94
95      int corriendo = 1;
96      int seleccionado = 0;
97
98      if(SDL_Init(SDL_INIT_VIDEO) < 0 ) {
99          fprintf(stderr, "No␣se␣puede␣iniciar␣SDL:␣%s\n", SDL_GetError());
100         exit(1);
101     }
102     atexit(SDL_Quit);
103
104     //Este if es importante para poder usar SDL_gfx
105     info = SDL_GetVideoInfo();
106     if ( info->vfmt->BitsPerPixel > 8 ) {
107         profundidad_color = info->vfmt->BitsPerPixel;
108         //printf("%d\n", profundidad_color);
109     } else {
110         profundidad_color = 16;
```

```
111     }
112
113     pantalla = SDL_SetVideoMode(ANCHO, ALTO, profundidad_color,
            SDL_SWSURFACE);
114     if(pantalla == NULL)
115     {
116         fprintf(stderr, "No se puede establecer el modo de video %dx%d: %
                s\n",
117                 ANCHO, ALTO, SDL_GetError());
118         exit(1);
119     }
120     SDL_WM_SetCaption("Segmento de Bezier", NULL);
121
122     color_fondo = SDL_MapRGB (pantalla->format,0,0,0);
123     color1 = gfxColor(255,255,255);
124     color2 = gfxColor(255,0,0);
125     color3 = gfxColor(0,0,255);
126
127     for(i=0; i<4; i++){
128         sb.x[i] = x[i];
129         sb.y[i] = y[i];
130     }
131     //BEZIER_imprimir(&sb, stdout);
132
133     dibujo(pantalla);
134
135     while(corriendo) {
136         while(SDL_PollEvent(&evento)) {
137             switch(evento.type){
138                 case SDL_MOUSEMOTION:{
139                     if(seleccionado){
140                         //actualizar el punto
141                         sb.x[i]=x_virtual(evento.button.x);
142                         sb.y[i]=y_virtual(evento.button.y);
143                         dibujo(pantalla);
144                     }
145                 }
146                     break;
147                 case SDL_MOUSEBUTTONDOWN:{
148                     for(i=0; i<4; i++){
149                         if(evento.button.button == SDL_BUTTON_LEFT && //
                            si hace clic sobre un nodo...
150                             ((evento.button.x > x_real(sb.x[i])-
                                ANCHO_RECTANGULO/2) &&
151                              (evento.button.y > y_real(sb.y[i])-
                                 ANCHO_RECTANGULO/2) &&
152                              (evento.button.x < x_real(sb.x[i])+
                                 ANCHO_RECTANGULO/2) &&
153                              (evento.button.y < y_real(sb.y[i])+
                                 ANCHO_RECTANGULO/2))
```

```
154                         ){
155                             //se selecciona y el índice queda en 'i'
156                             seleccionado = 1;
157                             //printf("seleccionado\n");
158                             break;
159                         }
160                     }
161                 }
162                     break;
163                 case SDL_MOUSEBUTTONUP:
164                     seleccionado = 0;
165                     break;
166 
167                 case SDL_KEYDOWN:
168                     if(evento.key.keysym.sym==SDLK_SPACE)
169                         dibujo(pantalla);
170                     break;
171                 case SDL_QUIT:
172                     corriendo = 0;
173                     break;
174             }
175         }
176     }
177 
178     SDL_Quit();
179     return 0;
180 }
```

bezier2

Este programa, presenta una implementación de una secuencia de tres segmentos de Bézier, con la funcionalidad de forzar continuidad C^1 entre los segmentos o mantener sólo continuidad G^0.

Veamos su uso:

arrastre con clic izquierdo Sobre los cuadros que representan los nodos de control, permite transformar la secuencia, manteniendo por defecto continuidad C^1 en las uniones de los segmentos.

letra l Alterna el dibujo de las líneas guías entre los nodos de control.

letra c Alterna el dibujo de los cuadros que marca la posición de los nodos de control.

letra i Imprime en consola las coordenadas de los nodos de control.

letra x Alterna entre forzar continuidad C^1 entre los segmentos (por defecto) o mantener sólo continuidad G^0. Cuando se activa la continuidad C^1 la posición de algunos nodos es forzada para garantizarla.

Los siguientes archivos presentan una implementación de una secuencia de segmentos de Bézier de longitud arbitraria:

Listing 9.10: Archivo de cabecera para operar secuencias de segmentos de Bézier

```
1  // /c09/bezier/bezier2.h
2  #include "bezier1.h"
3
4  /************ Unión de segmentos de Bézier *************************/
5
6  /*
7      Nodo para una lista lineal doble con nodo de control
8      para modelar una secuencia de segmentos de Bézier
9      con al menos un segmento.
10     Pero el nodo de control no es del mismo tipo
11     de los nodos.
12 */
13 typedef struct nodoBezier{
14     //coordenadas de los tres nodos,
15     //el último coincide con el primero
16     //del siguiente segmento.
17     double x[3];
18     double y[3];
19     double z[3];
20
21     struct nodoBezier *ant;
22     struct nodoBezier *sig;
23 } nodoBezier;
24
25 /*
26     Nodo de control para una secuencia de
27     segmentos de Bézier.
28 */
29 typedef struct{
30     //la coordenada del primer punto
31     //del primer segmento:
32     double x, y, z;
33
34     nodoBezier primero;
35     int numNodos;   //representa el número de estructuras, no de puntos
36     int continuidad;//indica si se garantizará la continuidad de 3er
           orden
37 } listaBezier;
38
39 /*
40     Imprime en la salida especificada los valores
41     almacenados en la estructura.
42 */
43 void BEZIER_imprimir2(listaBezier *lista, FILE *f);
44
45 /*
```

```
46      'numNodos' representa el número de segmentos de Bézier deseados
47  */
48  int BEZIER_crearLista(listaBezier *lista, int numNodos, double
        valorInicial);
49
50  /*
51      La longitud de los arreglos de entrada 'x' y 'y'
52      se asume a '3*lista->numNodos+1'.
53      También se asume que los puntos son coplanares en xy.
54      Se asume que la 'lista' ya ha sido creada.
55  */
56  int BEZIER_modificarLista2d(listaBezier *lista, double *x, double *y);
57
58  /*
59      La longitud de los arreglos de entrada 'x', 'y' y 'z'
60      se asume a '3*lista->numNodos+1'.
61      Se asume que la 'lista' ya ha sido creada.
62  */
63  int BEZIER_modificarLista(listaBezier *lista, double *x, double *y,
        double *z);
64
65  /*
66      Libera la memoria utilizada para
67      almacenar los 'nodoBezier'
68  */
69  void BEZIER_liberarLista(listaBezier *lista);
70
71  /*
72      Dado un valor del parámetro 't',
73      devuelve los valores del punto calculado
74      del segmento de Bézier.
75      Asume que el segmento es tridimensional.
76  */
77  int BEZIER_calculardeLista(listaBezier *lista, double t, double *x,
        double *y, double *z);
78
79  /*
80      Dado un valor del parámetro 't',
81      devuelve los valores del punto calculado
82      del segmento de Bézier.
83      Esta versión asume que los puntos son coplanares en xy.
84  */
85  int BEZIER_calculardeLista2d(listaBezier *lista, double t, double *x,
        double *y);
86
87  /*
88      Dado un índice de punto, actualizar el nodo de Bézier correspondiente
89  */
90  int BEZIER_actualizarNodo(listaBezier *lista, int ix, double x, double y,
        double z);
```

```
91
92  /*
93       Dado un índice, recuperar el punto
94       en las direcciones de 'x' y 'y'.
95  */
96  int BEZIER_recuperarNodo2d(listaBezier *lista, int ix, double *x, double
        *y);
97
98  /*
99       Dado un índice, recuperar el punto
100      en las direcciones de 'x', 'y' y 'z'.
101 */
102 int BEZIER_recuperarNodo(listaBezier *lista, int ix, double *x, double *y
        , double *z);
103
104 /*
105      Copia los valores de todos los puntos a los arreglos 'x', 'y' y 'z'.
106      Estos arreglos deben contener suficiente espacio.
107 */
108 int BEZIER_recuperarNodos(listaBezier *lista, double *x, double *y,
        double *z);
109
110 /*
111      Copia los valores de todos los puntos a los arreglos 'x' y 'y'.
112      Estos arreglos deben contener suficiente espacio.
113      Asume que los puntos son coplanares en xy.
114 */
115 int BEZIER_recuperarNodos2d(listaBezier *lista, double *x, double *y);
116
117 /*
118      Garantiza que la curva sea continua, forzando
119      algunos puntos.
120      Tendrán prioridad los puntos previos.
121 */
122 int BEZIER_acomodarContinua(listaBezier *lista);
123
124 void BEZIER_activarContinuidad(listaBezier *lista);
125
126 void BEZIER_desactivarContinuidad(listaBezier *lista);
127
128
129 /*
130  * Agrega un segmento nuevo de Bézier
131  * al final de la Lista especificada
132  * */
133 int BEZIER_agregarNodoNuevoaLista(listaBezier *lista, int valorInicial);
134
135 /*
136  * Agrega un segmento especificado de Bézier
137  * al final de la Lista especificada
```

```
138  * */
139 int BEZIER_agregarNodoaLista(listaBezier *lista, nodoBezier *nb);
```

Listing 9.11: Código fuente de funciones de secuencias de segmentos de Bézier

```
1  // /c09/bezier/bezier2.c
2  #include "bezier2.h"
3  #include <stdlib.h>
4
5  void BEZIER_imprimir2(listaBezier *lista, FILE *f){
6      nodoBezier *nb = NULL;
7      int i,j;
8      if(lista == NULL)
9          return;
10     fprintf(f,"\nPuntos de control de Bézier:\n");
11     fprintf(f,"i\tx_i\ty_i\tz_i\n");
12
13     //imprimir primer nodo
14     fprintf(f,"%d\t%3.2g\t%3.2g\t%3.2g\n",
15         i=1, lista->x, lista->y, lista->z);
16
17     //'primero' no es un puntero
18     for(nb = &(lista->primero); nb!=NULL; nb=nb->sig){
19         fprintf(f, "---\n");
20         for(j=0; j<3; j++)
21             fprintf(f,"%d\t%3.2g\t%3.2g\t%3.2g\n", ++i, nb->x[j], nb->y[j
                   ], nb->z[j]);
22     }
23 }
24
25 int BEZIER_crearLista(listaBezier *lista, int numNodos, double
       valorInicial){
26     if(lista == NULL || numNodos < 1)
27         return BEZIER_COD_ERROR_PARAMETROS;
28
29     nodoBezier *temp=NULL, *temp2=NULL;
30
31     lista->numNodos = numNodos;
32     lista->x = lista->y = lista->z =
33     lista->primero.x[0] = lista->primero.x[1] = lista->primero.x[2] =
34     lista->primero.y[0] = lista->primero.y[1] = lista->primero.y[2] =
           valorInicial;
35     lista->continuidad = 1;
36     lista->primero.sig=lista->primero.ant=NULL;
37
38     for(numNodos--;numNodos>0; numNodos--){
39         if(!(temp = (nodoBezier*)malloc(sizeof(nodoBezier))))
40             return BEZIER_COD_ERROR_MEMORIA;
41         if(temp2==NULL){ //es el primero creado dinámicamente
42             lista->primero.sig = temp2 = temp;
```

```
43              temp->sig = NULL;
44              temp->ant = &(lista->primero);
45          }else{  //ya hay un nodo anterior en 'temp2'
46              temp->sig = NULL;
47              temp->ant = temp2;
48              temp2->sig = temp;
49              temp2 = temp;
50          }
51          temp->x[0]=temp->x[1]=temp->x[2]=
52          temp->y[0]=temp->y[1]=temp->y[2]= valorInicial;
53          temp = NULL;
54      }
55      return BEZIER_COD_EXITO;
56  }
57
58  int BEZIER_modificarLista2d(listaBezier *lista, double *x, double *y){
59      if(lista == NULL || x == NULL || y == NULL)
60          return BEZIER_COD_ERROR_PARAMETROS;
61      nodoBezier *temp=NULL;
62      int i=0, j;
63
64      lista->x = x[i];
65      lista->y = y[i++];
66      lista->z = 0.0;
67
68      for(temp = &(lista->primero); temp!=NULL; temp=temp->sig)
69      for(j=0; j<3; j++){
70          temp->x[j] = x[i];
71          temp->y[j] = y[i++];
72          temp->z[j] = 0.0;
73      }
74      if(lista->continuidad)
75          BEZIER_acomodarContinua(lista);
76      return BEZIER_COD_EXITO;
77  }
78
79  int BEZIER_modificarLista(listaBezier *lista, double *x, double *y,
        double *z){
80      if(lista == NULL || x == NULL || y == NULL || z == NULL)
81          return BEZIER_COD_ERROR_PARAMETROS;
82      nodoBezier *temp=NULL;
83      int i=0, j;
84
85      lista->x = x[i];
86      lista->y = y[i];
87      lista->z = z[i++];
88
89      for(temp = &(lista->primero); temp!=NULL; temp=temp->sig)
90      for(j=0; j<3; j++){
91          temp->x[j] = x[i];
```

```
92          temp->y[j] = y[i];
93          temp->z[j] = z[i++];
94      }
95      if(lista->continuidad)
96          BEZIER_acomodarContinua(lista);
97      return BEZIER_COD_EXITO;
98  }
99
100 void BEZIER_liberarLista(listaBezier *lista){
101     if(lista == NULL)
102         return;
103     nodoBezier *nb = lista->primero.sig, *temp=NULL;
104     while(nb){
105         temp = nb->sig;
106         free(nb);
107         nb=temp;
108     }
109 }
110
111 int BEZIER_calculardeLista2d(listaBezier *lista, double t, double *x,
        double *y){
112     if(lista == NULL || x == NULL || y == NULL)
113         return BEZIER_COD_ERROR_PARAMETROS;
114     if(t<0.0 || t>1.0)
115         return BEZIER_COD_ERROR_DOMINIO;
116
117     nodoBezier *temp=NULL, *nb=NULL;
118     int i=0, j;
119
120     //calcular el índice del nodo 'nodoBezier'
121     //con el que debe calcularse este 't'
122     int ixBezier = (int) (t*lista->numNodos);
123     if(ixBezier==lista->numNodos) ixBezier--;
124
125     t = lista->numNodos * t - ixBezier;
126
127     double  t1 = t;
128     double  t2 = t*t;
129     double  t3 = t2*t;
130     double  _1_t1 = 1-t;
131     double  _1_t2 = _1_t1 * _1_t1;
132     double  _1_t3 = _1_t1 * _1_t2;
133
134     nb = temp = &(lista->primero);
135     if(ixBezier==0){    //es el primer 'nodoBezier'
136         *x = _1_t3 * lista->x + 3*t1*_1_t2 * nb->x[0] + 3*t2*_1_t1 * nb->
                x[1] + t3 * nb->x[2];
137         *y = _1_t3 * lista->y + 3*t1*_1_t2 * nb->y[0] + 3*t2*_1_t1 * nb->
                y[1] + t3 * nb->y[2];
138     }else{  //no es el primer 'nodoBezier' con el que se calculará 't'
```

```
139         i++;
140         for(temp = temp->sig; temp!=NULL; temp=temp->sig)
141             if(i==ixBezier){
142                 nb = temp;
143                 break;
144             }
145             else i++;
146         //aquí se asume que siempre lo encuentra...
147         *x = _1_t3 * nb->ant->x[2] + 3*t1*_1_t2 * nb->x[0] + 3*t2*_1_t1 *
                nb->x[1] + t3 * nb->x[2];
148         *y = _1_t3 * nb->ant->y[2] + 3*t1*_1_t2 * nb->y[0] + 3*t2*_1_t1 *
                nb->y[1] + t3 * nb->y[2];
149     }
150
151     return BEZIER_COD_EXITO;
152 }
153
154 int BEZIER_calculardeLista(listaBezier *lista, double t, double *x,
        double *y, double *z){
155     if(lista == NULL || x == NULL || y == NULL || z == NULL)
156         return BEZIER_COD_ERROR_PARAMETROS;
157     if(t<0.0 || t>1.0)
158         return BEZIER_COD_ERROR_DOMINIO;
159
160     nodoBezier *temp=NULL, *nb=NULL;
161     int i=0, j;
162
163     //calcular el índice del nodo 'nodoBezier'
164     //con el que debe calcularse este 't'
165     int ixBezier = (int) (t*lista->numNodos);
166     if(ixBezier==lista->numNodos) ixBezier--;
167
168     t = lista->numNodos * t - ixBezier;
169
170     double  t1 = t;
171     double  t2 = t*t;
172     double  t3 = t2*t;
173     double  _1_t1 = 1-t;
174     double  _1_t2 = _1_t1 * _1_t1;
175     double  _1_t3 = _1_t1 * _1_t2;
176
177     nb = temp = &(lista->primero);
178     if(ixBezier==0){    //es el primer 'nodoBezier'
179         *x = _1_t3 * lista->x + 3*t1*_1_t2 * nb->x[0] + 3*t2*_1_t1 * nb->
                x[1] + t3 * nb->x[2];
180         *y = _1_t3 * lista->y + 3*t1*_1_t2 * nb->y[0] + 3*t2*_1_t1 * nb->
                y[1] + t3 * nb->y[2];
181         *z = _1_t3 * lista->z + 3*t1*_1_t2 * nb->z[0] + 3*t2*_1_t1 * nb->
                z[1] + t3 * nb->z[2];
182     }else{  //no es el primer 'nodoBezier' con el que se calculará 't'
```

```
183         i++;
184         for(temp = temp->sig; temp!=NULL; temp=temp->sig)
185             if(i==ixBezier){
186                 nb = temp;
187                 break;
188             }
189             else i++;
190         //aquí se asume que siempre lo encuentra...
191         *x = _1_t3 * nb->ant->x[2] + 3*t1*_1_t2 * nb->x[0] + 3*t2*_1_t1 *
                nb->x[1] + t3 * nb->x[2];
192         *y = _1_t3 * nb->ant->y[2] + 3*t1*_1_t2 * nb->y[0] + 3*t2*_1_t1 *
                nb->y[1] + t3 * nb->y[2];
193         *z = _1_t3 * nb->ant->z[2] + 3*t1*_1_t2 * nb->z[0] + 3*t2*_1_t1 *
                nb->z[1] + t3 * nb->z[2];
194     }
195
196     return BEZIER_COD_EXITO;
197 }
198
199 int BEZIER_actualizarNodo2d(listaBezier *lista, int ix, double x, double
        y){
200     return BEZIER_actualizarNodo(lista, ix, x, y, 0.0);
201 }
202 int BEZIER_actualizarNodo(listaBezier *lista, int ix, double x, double y,
         double z){
203     if(lista == NULL || ix < 0)
204         return BEZIER_COD_ERROR_PARAMETROS;
205
206     nodoBezier *temp=NULL;
207     int i=0, j;
208
209     if(ix==0){
210         lista->x=x;
211         lista->y=y;
212         lista->z=z;
213         return BEZIER_COD_EXITO;
214     }
215     i++;
216     for(temp = &(lista->primero); temp!=NULL; temp=temp->sig)
217     for(j=0; j<3; j++)
218         if(i==ix){
219             //Mecanismo de coordinación con los puntos adyacentes:
220             if(lista->continuidad)
221             switch(j){
222                 case 0:{
223                     /*
224                     Desplazar el penúltimo punto
225                     del nodo anterior,
226                     si hay nodo anterior.
227                     Esto es para garantizar la colinealidad
```

```
228                     en las uniones de los segmentos de Bézier.
229                     */
230                     if(temp->ant){
231                         temp->ant->x[1] = temp->ant->x[2] - (x-temp->ant
                                ->x[2]);
232                         temp->ant->y[1] = temp->ant->y[2] - (y-temp->ant
                                ->y[2]);
233                         temp->ant->z[1] = temp->ant->z[2] - (z-temp->ant
                                ->z[2]);
234                     }
235                 }
236                 break;
237                 case 1:{
238                     /*
239                     Desplazar el primer punto
240                     del nodo siguiente,
241                     si hay nodo siguiente.
242                     Esto es para garantizar la colinealidad
243                     en las uniones de los segmentos de Bézier.
244                     */
245                     if(temp->sig){
246                         temp->sig->x[0] = temp->x[2] - (x-temp->x[2]);
247                         temp->sig->y[0] = temp->y[2] - (y-temp->y[2]);
248                         temp->sig->z[0] = temp->z[2] - (z-temp->z[2]);
249                     }
250                 }
251                 break;
252                 case 2:{
253                     /*
254                     Desplazar el punto anterior
255                     y el siguiente, si es que
256                     hay un siguiente.
257                     */
258                     if(temp->sig){
259                         temp->x[1] += x - temp->x[2];
260                         temp->y[1] += y - temp->y[2];
261                         temp->z[1] += z - temp->z[2];
262                         temp->sig->x[0] += x - temp->x[2];
263                         temp->sig->y[0] += y - temp->y[2];
264                         temp->sig->z[0] += z - temp->z[2];
265                     }
266                 }
267                 break;
268             }
269             temp->x[j]=x;
270             temp->y[j]=y;
271             temp->z[j]=z;
272             return BEZIER_COD_EXITO;
273         }
274         else i++;
```

```
275     return BEZIER_COD_ERROR_DOMINIO;
276 }
277
278 int BEZIER_recuperarNodo2d(listaBezier *lista, int ix, double *x, double
        *y){
279     if(lista == NULL || ix < 0 || x == NULL || y == NULL)
280         return BEZIER_COD_ERROR_PARAMETROS;
281
282     nodoBezier *temp=NULL;
283     int i=0, j;
284
285     if(ix==0){
286         *x=lista->x;
287         *y=lista->y;
288         return BEZIER_COD_EXITO;
289     }
290     i++;
291     for(temp = &(lista->primero); temp!=NULL; temp=temp->sig)
292     for(j=0; j<3; j++)
293         if(i==ix){
294             *x=temp->x[j];
295             *y=temp->y[j];
296             return BEZIER_COD_EXITO;
297         }
298         else i++;
299     return BEZIER_COD_ERROR_DOMINIO;
300 }
301
302 int BEZIER_recuperarNodo(listaBezier *lista, int ix, double *x, double *y
        , double *z){
303     if(lista == NULL || ix < 0 || x == NULL || y == NULL || z == NULL)
304         return BEZIER_COD_ERROR_PARAMETROS;
305
306     nodoBezier *temp=NULL;
307     int i=0, j;
308
309     if(ix==0){
310         *x=lista->x;
311         *y=lista->y;
312         *z=lista->z;
313         return BEZIER_COD_EXITO;
314     }
315     i++;
316     for(temp = &(lista->primero); temp!=NULL; temp=temp->sig)
317     for(j=0; j<3; j++)
318         if(i==ix){
319             *x=temp->x[j];
320             *y=temp->y[j];
321             *z=temp->z[j];
322             return BEZIER_COD_EXITO;
```

```
323         }
324         else i++;
325     return BEZIER_COD_ERROR_DOMINIO;
326 }
327 
328 int BEZIER_recuperarNodos(listaBezier *lista, double *x, double *y,
        double *z){
329     if(lista == NULL || x == NULL || y == NULL || z == NULL)
330         return BEZIER_COD_ERROR_PARAMETROS;
331 
332     nodoBezier *temp=NULL;
333     int i=0, j;
334 
335     x[i]  =lista->x;
336     y[i]  =lista->y;
337     z[i++]=lista->z;
338     for(temp = &(lista->primero); temp!=NULL; temp=temp->sig)
339     for(j=0; j<3; j++){
340         x[i]  =temp->x[j];
341         y[i]  =temp->y[j];
342         z[i++]=temp->z[j];
343     }
344     return BEZIER_COD_EXITO;
345 }
346 
347 int BEZIER_recuperarNodos2d(listaBezier *lista, double *x, double *y){
348     if(lista == NULL || x == NULL || y == NULL)
349         return BEZIER_COD_ERROR_PARAMETROS;
350 
351     nodoBezier *temp=NULL;
352     int i=0, j;
353 
354     x[i]  =lista->x;
355     y[i++]=lista->y;
356     for(temp = &(lista->primero); temp!=NULL; temp=temp->sig)
357     for(j=0; j<3; j++){
358         x[i]  =temp->x[j];
359         y[i++]=temp->y[j];
360     }
361     return BEZIER_COD_EXITO;
362 }
363 
364 int BEZIER_acomodarContinua(listaBezier *lista){
365     if(lista == NULL)
366         return BEZIER_COD_ERROR_PARAMETROS;
367 
368     nodoBezier *temp=NULL;
369 
370     for(temp = (&(lista->primero))->sig; temp!=NULL; temp=temp->sig){
371         /*
```

```
372             Desplazar el primer punto
373             del nodo actual.
374             Esto es para garantizar la colinealidad
375             en las uniones de los segmentos de Bézier.
376             */
377             temp->x[0] = temp->ant->x[2] - (temp->ant->x[1]-temp->ant->x[2]);
378             temp->y[0] = temp->ant->y[2] - (temp->ant->y[1]-temp->ant->y[2]);
379             temp->z[0] = temp->ant->z[2] - (temp->ant->z[1]-temp->ant->z[2]);
380         }
381
382         return BEZIER_COD_EXITO;
383     }
384
385     void BEZIER_activarContinuidad(listaBezier *lista){
386         lista->continuidad = 1;
387     }
388
389     void BEZIER_desactivarContinuidad(listaBezier *lista){
390         lista->continuidad = 0;
391     }
392
393
394     int BEZIER_agregarNodoNuevoaLista(listaBezier *lista, int valorInicial){
395         if(lista == NULL)
396             return BEZIER_COD_ERROR_PARAMETROS;
397
398         nodoBezier *temp=NULL, *temp2=NULL;
399
400         if(!(temp = (nodoBezier*)malloc(sizeof(nodoBezier))))
401             return BEZIER_COD_ERROR_MEMORIA;
402         for(temp2=&(lista->primero); temp2->sig; temp2=temp2->sig);
403
404         temp->ant=temp->sig=NULL;
405
406         temp2->sig=temp;
407         temp->ant = temp2;
408         lista->numNodos++;
409
410         temp->x[0]=temp->x[1]=temp->x[2]=
411         temp->y[0]=temp->y[1]=temp->y[2]= valorInicial;
412
413         return BEZIER_COD_EXITO;
414     }
415
416     int BEZIER_agregarNodoaLista(listaBezier *lista, nodoBezier *nb){
417         if(lista == NULL || nb == NULL)
418             return BEZIER_COD_ERROR_PARAMETROS;
419
420         nodoBezier *temp2=NULL;
421
```

```
422     for(temp2=&(lista->primero); temp2->sig; temp2=temp2->sig);
423
424     nb->ant=nb->sig=NULL;
425
426     temp2->sig=nb;
427     nb->ant = temp2;
428     lista->numNodos++;
429
430     return BEZIER_COD_EXITO;
431 }
```

El siguiente archivo permite la manipulación de una unión de tres segmentos de Bézier, permitiendo alternar la continuidad en sus puntos de unión, entre C^1 y G^0:

Listing 9.12: Programa principal de manipulación de una secuencia de segmentos de Bézier

```
1  // /c09/bezier/main2.c
2  #include <SDL/SDL.h>
3  #include "bezier2.h"
4  #include <SDL/SDL_gfxPrimitives.h>
5
6  #define ANCHO 640
7  #define ALTO  480
8
9  #define ANCHO_RECTANGULO 15
10
11 #define XMIN  0.0
12 #define XMAX 15.0
13 #define YMIN  0.0
14 #define YMAX 15.0
15 #define TPASO 0.01
16
17 #define TAM 10
18 //Debe cumplirse la siguiente relación:
19 //numPuntos = 3*numNodos+1;
20 double x[TAM] = {1.3, 3.0, 6.0, 5.0, 4.0, 5.0, 6.0, 8.0, 10.0, 9.0};
21 double y[TAM] = {1.3, 5.0, 7.4, 9.2, 2.0, 3.0, 4.0, 5.0, 7.0, 4.0};
22 listaBezier lista;
23
24
25 static inline int x_real(double x_virtual){
26     return (int) (ANCHO * x_virtual / XMAX);
27 }
28 static inline int y_real(double y_virtual){
29     return (int) (ALTO * (1.0 - y_virtual / YMAX));
30 }
31 static inline double x_virtual(int x_real){
32     return (XMAX * x_real) / ANCHO;
33 }
```

```
34 static inline double y_virtual(int y_real){
35     return YMAX * (1.0 - y_real / (double)ALTO);
36 }
37 
38 Uint32 gfxColor(Uint8 r, Uint8 g, Uint8 b){
39     return
40         r << 24 |
41         g << 16 |
42         b << 8  |
43         255;     //este valor es la opacidad del color
44                  //y debe ser máxima para que sea sólido
45 }
46 
47 //Variables globales
48 
49 int i, j, k, l;
50 double tvar, xvar, yvar;
51 Uint32 color_fondo, color_curva, color_lineas, color_union_nodos,
       color_union;
52 int mostrar_lineas, mostrar_cuadrados;
53 
54 void dibujo(SDL_Surface *pantalla){
55     i=j=k=l=0;
56     SDL_Rect rect;
57 
58     //Borra la pantalla
59     SDL_FillRect(pantalla, NULL, color_fondo);
60 
61     //dibujar los rectángulos de control
62     rect.w = rect.h = ANCHO_RECTANGULO;
63     if(mostrar_cuadrados)
64     for(i=0; i<TAM; i++){
65         rect.x = x_real(x[i])-ANCHO_RECTANGULO/2;
66         rect.y = y_real(y[i])-ANCHO_RECTANGULO/2;
67         if(lista.continuidad)
68             switch(i%3){
69                 case 0:
70                     rectangleColor(pantalla, rect.x, rect.y, rect.x +
                           rect.w, rect.y + rect.h, color_union_nodos);
71                 break;
72                 default:
73                     rectangleColor(pantalla, rect.x, rect.y, rect.x +
                           rect.w, rect.y + rect.h, color_union);
74                 break;
75             }
76         else
77             rectangleColor(pantalla, rect.x, rect.y, rect.x + rect.w,
                   rect.y + rect.h, color_union_nodos);
78     }
79 
```

```
80      //dibujar líneas entre los nodos
81      if(mostrar_lineas)
82      for(i=1; i<TAM; i++){
83          lineColor(pantalla,
84              x_real(x[i-1]), y_real(y[i-1]),
85              x_real(x[i])  , y_real(y[i])  , color_lineas);
86      }
87
88      //calcular los puntos de la curva
89      tvar = 0.0;
90      BEZIER_calculardeLista2d(&lista, tvar, &xvar, &yvar);
91      i=x_real(xvar);
92      j=y_real(yvar);
93      for(tvar = TPASO; tvar <= 1.0; tvar+=TPASO){
94          if(!BEZIER_calculardeLista2d(&lista, tvar, &xvar, &yvar)){
95              lineColor(pantalla, i, j, k=x_real(xvar), l=y_real(yvar),
                    color_curva);
96              i=k; j=l;
97          }
98      }
99      BEZIER_calculardeLista2d(&lista, tvar, &xvar, &yvar);
100     lineColor(pantalla, i, j, k=x_real(xvar), l=y_real(yvar), color_curva
            );
101
102     //vuelca el buffer en la pantalla:
103     SDL_Flip (pantalla);
104 }
105
106 int main(int argc, char *argv[]){
107     SDL_Surface *pantalla = NULL;
108     SDL_Event evento;
109     int profundidad_color;
110     const SDL_VideoInfo *info;
111     Uint32 color;
112     int i;
113
114     int corriendo = 1;
115     int seleccionado = 0;
116
117     mostrar_cuadrados = 1;
118     mostrar_lineas = 1;
119
120     if(SDL_Init(SDL_INIT_VIDEO) < 0 ) {
121         fprintf(stderr, "No␣se␣puede␣iniciar␣SDL:␣%s\n", SDL_GetError());
122         exit(1);
123     }
124     atexit(SDL_Quit);
125
126     //Este if es importante para poder usar SDL_gfx
127     info = SDL_GetVideoInfo();
```

```
128     if ( info->vfmt->BitsPerPixel > 8 ) {
129         profundidad_color = info->vfmt->BitsPerPixel;
130         //printf("%d\n", profundidad_color);
131     } else {
132         profundidad_color = 16;
133     }
134 
135     pantalla = SDL_SetVideoMode(ANCHO, ALTO, profundidad_color,
            SDL_SWSURFACE);
136     if(pantalla == NULL)
137     {
138         fprintf(stderr, "No se puede establecer el modo de video %dx%d: %
                s\n",
139             ANCHO, ALTO, SDL_GetError());
140         exit(1);
141     }
142     SDL_WM_SetCaption("Segmentos de Bezier", NULL);
143 
144     color_fondo = SDL_MapRGB (pantalla->format,0,0,0);
145     color_curva = gfxColor(255,255,255); //línea
146     color_lineas = gfxColor(0,0,255);      //unión puntos
147     color_union_nodos = gfxColor(255,0,0);      //unión de nodos
148     color_union = gfxColor(255,255,0);    //puntos adyacentes
149 
150     BEZIER_crearLista(&lista, (TAM-1)/3,0.0);
151     BEZIER_modificarLista2d(&lista, x, y);
152     BEZIER_recuperarNodos2d(&lista, x, y);
153     BEZIER_imprimir2(&lista, stdout);
154 
155     dibujo(pantalla);
156 
157     while(corriendo) {
158         while(SDL_PollEvent(&evento)) {
159             switch(evento.type){
160                 case SDL_MOUSEMOTION:{
161                     if(seleccionado){
162                         //actualizar el punto
163                         x[i]=x_virtual(evento.button.x);
164                         y[i]=y_virtual(evento.button.y);
165                         BEZIER_actualizarNodo2d(&lista, i, x[i], y[i]);
166                         BEZIER_recuperarNodos2d(&lista, x, y);
167                         dibujo(pantalla);
168                     }
169                 }
170                     break;
171                 case SDL_MOUSEBUTTONDOWN:{
172                     for(i=0; i<TAM; i++){
173                         if(evento.button.button == SDL_BUTTON_LEFT && //
                                si hace clic sobre un nodo...
174                             ((evento.button.x > x_real(x[i])-
```

```
                                ANCHO_RECTANGULO/2) &&
175                          (evento.button.y > y_real(y[i])-
                                ANCHO_RECTANGULO/2) &&
176                          (evento.button.x < x_real(x[i])+
                                ANCHO_RECTANGULO/2) &&
177                          (evento.button.y < y_real(y[i])+
                                ANCHO_RECTANGULO/2))
178                     ){
179                         //se selecciona y el índice queda en 'i'
180                         seleccionado = 1;
181                         //printf("seleccionado\n");
182                         break;
183                     }
184                 }
185             }
186                 break;
187             case SDL_MOUSEBUTTONUP:
188                 seleccionado = 0;
189                 break;
190
191             case SDL_KEYDOWN:
192                 switch(evento.key.keysym.sym){
193                     case SDLK_i: {
194                         BEZIER_imprimir2(&lista, stdout);
195                     }
196                     break;
197                     case SDLK_l: {
198                         mostrar_lineas = !mostrar_lineas;
199                         dibujo(pantalla);
200                     }
201                     break;
202                     case SDLK_c: {
203                         mostrar_cuadrados = !mostrar_cuadrados;
204                         dibujo(pantalla);
205                     }
206                     break;
207                     case SDLK_r: {
208                         dibujo(pantalla);
209                     }
210                     break;
211                     case SDLK_x: {
212                         lista.continuidad = !lista.continuidad;
213                         if(lista.continuidad){
214                             BEZIER_acomodarContinua(&lista);
215                             BEZIER_recuperarNodos2d(&lista, x, y);
216                         }
217                         dibujo(pantalla);
218                     }
219                     break;
220                 }
```

```
221                         break;
222
223                     case SDL_QUIT:
224                         corriendo = 0;
225                         break;
226                 }
227             }
228         }
229
230         SDL_Quit();
231         return 0;
232 }
```

editorBezier.py

Este programa fue implementado con el doble propósito de servir como ejemplo para este capítulo y para hacer la parte principal de la carátula de este libro.

Sirve para crear una serie de Secuencias de segmentos de Bézier. Las opciones de interacción son:

arrastre con clic izquierdo Sobre los cuadros que representan los puntos de control, permite moverlos.
Sobre el fondo blanco, mueve el marco virtual de todas las secuencias.

arrastre con clic izquierdo+CTRL Sobre algún punto de control, mueve toda la secuencia a la que pertenece el punto.

ruedas del ratón Implementa acercamiento/alejamiento.

ruedas del ratón + tecla x/y Expande/Contrae el marco virtual horizontalmente/verticalmente.

CTRL+tecla L Alterna dibujo de las líneas rectas que unen los puntos de control.

CTRL+tecla C Alterna dibujo de los rectángulos que marcan los puntos de control.

CTRL+tecla A Agrega una nueva Secuencia de segmentos de Bézier de manera aleatoria.

CTRL+tecla X Alterna entre forzar continuidad C^1 entre los segmentos (por defecto) o mantener sólo continuidad G^0. Cuando se activa la continuidad C^1 la posición de algunos nodos es forzada para garantizarla. Igual que en la aplicación anterior.

CTRL+tecla G Guarda el archivo (interfaz de línea de comandos).

clic derecho Sobre un punto de control, agrega tres puntos más a partir de ahí.

clic central Sobre un punto de control, borra todo un segmento de una secuencia.

ruedas del ratón+CTRL Aumento/disminución del grosor de la curva.

ruedas del ratón+SHIFT Aumento/disminución del tamaño de los cuadros de control.

CTRL+tecla Arriba/Abajo Aumento/disminución del grueso de los cuadros de control.

SHIFT+tecla Arriba/Abajo Aumento/disminución del detalle/finura de las curvas

9.5. Splines vs. Bézier

Las curvas de Bézier no requieren de cálculos previos para el cálculo de los puntos que la componen, a diferencia de las curvas splines. Además, el cálculo de cada punto no requiere una búsqueda como en el caso de splines.

Esto permite que las curvas de Bézier sean más atractivas para el uso de diseño interactivo que las splines. Aunque hay que considerar que los puntos de control de Bézier, no pertenecen todos a la curva generada, y en consecuencia, su uso es menos natural que sus contrapartes splines. Además, el usuario tendrá que distinguir cuáles puntos de control sí interpolan la curva y cuáles controlan los vectores tangentes. Para poder lograr esto, habrá que enriquecer más la interfaz, lo que requiere cálculos adicionales.

Para que un usuario no matemático modele algo usando splines, sólo tendría que aprender a agregar y eliminar puntos de control de la curva, mientras que si usa Bézier, tendrá que pasar por un proceso de aprendizaje mayor para entender los efectos de cambiar los puntos de control sobre las curvas o superficies.

9.6. Ejercicios

1. Mencione las diferencias entre las curvas paramétricas Spline y Bézier.
2. Sea $\psi(t) = 2t\hat{i} + \frac{5}{2}t^2\hat{j}$ para $0 \leqslant t \leqslant 1$ y sea $\omega(t) = (4t-2)\,\hat{i} - \left(t^4 + 3t^2 - \frac{3}{2}\right)\hat{j}$ para $1 \leqslant t \leqslant 2$. Observe que $\psi(1) = \left(2, \frac{5}{2}\right) = \omega(1)$, de manera que ambas curvas se unen con continuidad C^0.
   a) Grafique $\psi(t)$ y $\omega(t)$ para $0 \leqslant t \leqslant 1$ y $1 \leqslant t \leqslant 2$ respectivamente.
   b) Determine si $\psi(t)$ y $\omega(t)$ cumplen con continuidad G^1 en el punto de unión. Recuerde que deberá evaluar $\frac{d}{dt}\psi(t=1)$ y $\frac{d}{dt}\omega(t=1)$.
   c) Determine si $\psi(t)$ y $\omega(t)$ cumplen con continuidad C^1 en el punto de unión.
3. Muestre que las dos curvas
   $f(t) = \left(\frac{1}{2}t^4 - \frac{2}{9}t^3 + \frac{7}{6}t^2 + \frac{1}{3}t\right)\hat{i} + \left(\frac{5}{3}t^3 - t^2 - t + \frac{\sqrt{3}\pi - 18}{3\sqrt{3}\pi}\right)\hat{j}$, con $0 \leqslant t \leqslant 1$ y
   $g(t) = \left(\frac{2}{\pi}\sin(2\pi t) + \frac{16}{9}\right)\hat{i} - \frac{12}{\sqrt{3}\pi}\cos\left(\frac{\pi}{3}t\right)\hat{j}$, con $1 \leqslant t \leqslant 2$ tienen continuidad C^1 y G^1 cuando se unen en $f(1) = g(1)$.

4. Modifique el programa presentado en la subsección 9.3.7 en la página 178, para que permita al usuario agregar y eliminar nodos de control.
5. Modifique el programa presentado en el apartado *bezier2* de la subsección 9.4.6 en la página 221, para que además de permitir elegir entre mantener continuidad C^1 y G^0 también permita mantener continuidad G^1 (sin C^1, obviamente).

10 Superficies paramétricas

10.1. Representación algebráica de Superficies paramétricas

De manera análoga a las curvas paramétricas, las superficies paramétricas se pueden representar de las siguientes maneras:

$$\vec{r}(s,t) = x(s,t)\hat{i} + y(s,t)\hat{j} + z(s,t)\hat{k}$$

$$\vec{r}(u,v) = x(u,v)\hat{i} + y(u,v)\hat{j} + z(u,v)\hat{k}$$

$$\begin{cases} x = x(s,t) \\ y = y(s,t) \\ z = z(s,t) \end{cases}, \text{«}restricciones\text{»}$$

Veamos algunos ejemplos de superficies paramétricas:

Cilindro hueco de altura H y radio R:

$$\begin{cases} x = R\cos\theta \\ z = R\sin\theta \\ y = v \end{cases}, \begin{array}{l} 0 \leq \theta \leq 2\pi \\ 0 \leq v \leq H \end{array}$$

En la figura 10.1 en la página siguiente se presenta un cilindro con $R = 4$ y $H = 6$, generado con el comando `plot3d([4*cos(th),v,4*sin(th)], [th, 0, 2*%pi], [v, 0, 6]);` de Maxima.

Toroide con radio central R y radio lateral r (ver figura 10.2):

$$\begin{cases} x = (r\cos\theta + R)\cos\phi \\ y = r\sin\theta \\ z = (r\cos\theta + R)\sin\phi \end{cases}, \begin{array}{l} 0 \leq \theta \leq 2\pi \\ 0 \leq \phi \leq 2\pi \end{array}$$

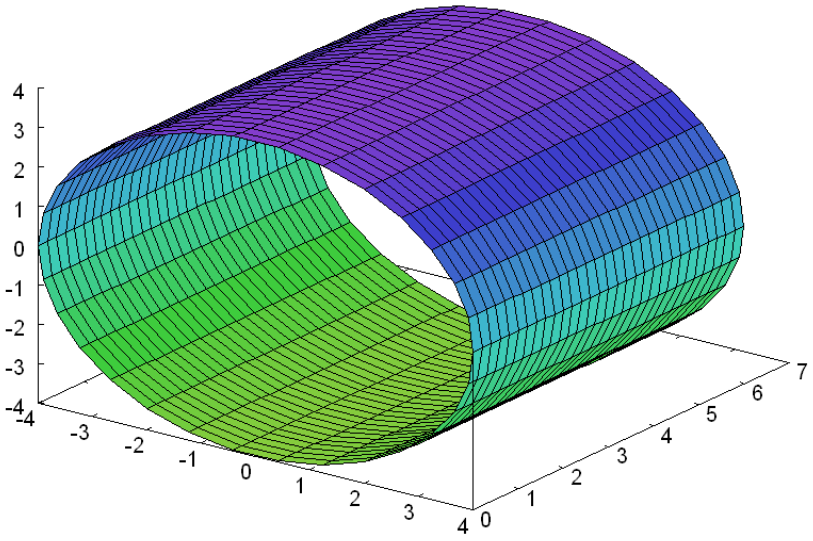


Figura 10.1: Cilindro generado con Maxima

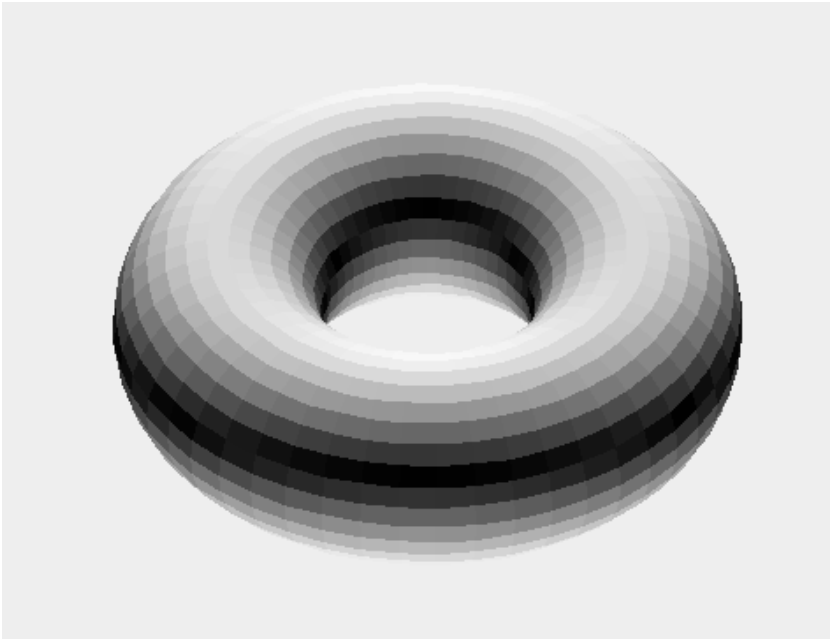


Figura 10.2: Toroide

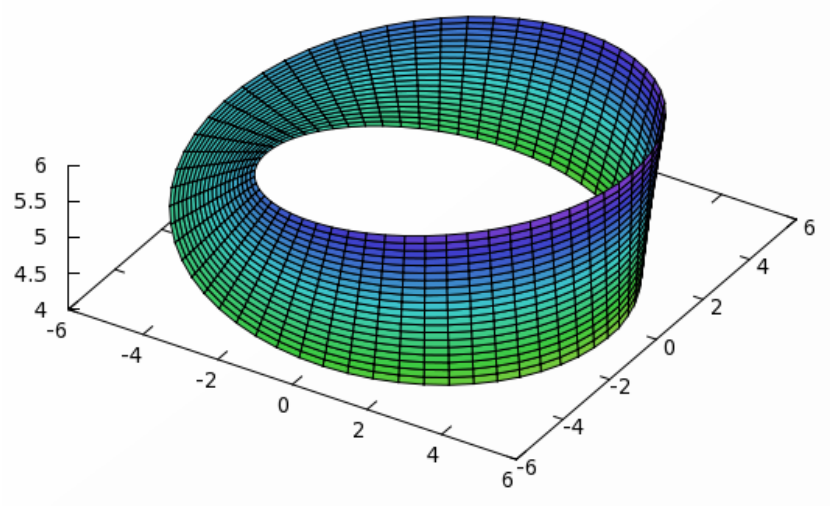


Figura 10.3: Cinta de Möbius

Cinta de *Möbius* con radio de la cinta R y un ancho de la cinta A:

$$\begin{cases} x = \left(Av\sin\frac{u}{2} + R\right)\cos u \\ y = \left(Av\sin\frac{u}{2} + R\right)\sin u \\ z = Av\cos\frac{u}{2} + R \end{cases} , \quad \begin{array}{l} 0 \le u \le 2\pi \\ -\frac{1}{2} \le v \le \frac{1}{2} \end{array}$$

En la figura 10.3 se presenta una cinta de Möbius con $A = 2$ y $R = 5$, generada con el script de Maxima:

```
A: 2$
R: 5$
x: (A*v*sin(u/2)+R)*cos(u)$
y: (A*v*sin(u/2)+R)*sin(u)$
z: A*v*cos(u/2)+R$
plot3d([x,y,z], [u, 0,2*%pi], [v, -1/2, 1/2], ['grid, 60,20]);
```

Superficie paramétrica suave

$\vec{r}(r,t)$ es una curva suave en $\{[a,b]\times[c,d]\}$ si la derivada direccional $D_{\hat{u}}\vec{r}(s,t)$ es continua en todos los puntos de $\{[a,b]\times[c,d]\}$ para cualquier dirección $\hat{u}$.

10.2. Ejercicios

1. Escriba la ecuación paramétrica de un sorbete que está formado por la mitad superior de una bola con centro en el origen y radio 5, y un cono circular con la

base en el plano xy y la punta en $(0, 0, -15)$.
Sugerencia 1: Hacer la parametrización en coordenadas cilíndricas.
Sugerencia 2: Es una ecuación seccionada.

11 Mallas Poligonales

Una manera alternativa de modelar cuerpos tridimensionales arbitrarios, es aproximando sus superfices con lo que se conoce como Mallas Poligonales. Las mallas poligonales son conjuntos de puntos, aristas (líneas) y polígonos relacionados entre sí con el objetivo de aproximar un cuerpo o una superficie.

Existen diversas maneras de implementar tales estructuras de datos. Algunas requieren más memoria que otras, algunas requieren algoritmos más sofisticados para operarlas que otras, algunas posibilitan ciertos análisis que otras no. Todo dependerá de las necesidades concretas de la aplicación a desarrollar.

En este punto es conveniente volver a tener frescos los conceptos de geometría analítica vectorial relacionados con planos. También recomendamos leer el capítulo 14 en la página 283 para poder comprender mejor las descripciónes de las estructuras de datos.

A continuación se presentarán algunas representaciones genéricas diferentes de mallas poligonales (las primeras tres, adaptadas de [Foley et al., p. 366-367]):

11.1. Representación explícita

En esta representación, los objetos tridimensionales se representan como una lista de polígonos; y cada polígono se representa como una lista propia de los puntos que lo conforman.

El diagrama de clases correspondiente puede apreciarse en la figura 11.1.

Los objetos tridimensionales se modelan como una "composición" de uno o más polígonos. Y cada polígono se modela como una secuencia lineal de puntos. Obviamente estos son los vértices que delimitan cada polígono.

Esta representación tiene el problema de la redundancia de información. Veámoslo con un ejemplo:

En la figura 11.2 se presenta un "objeto tridimensional" de cuatro vértices dispuestos en dos polígonos adyacentes. Su representación en diagrama de objetos (de acuerdo al diagrama de clases de la figura 11.1) está en la figura 11.3.

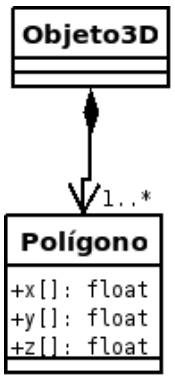


Figura 11.1: Diagrama de clases de una malla poligonal en representación explícita

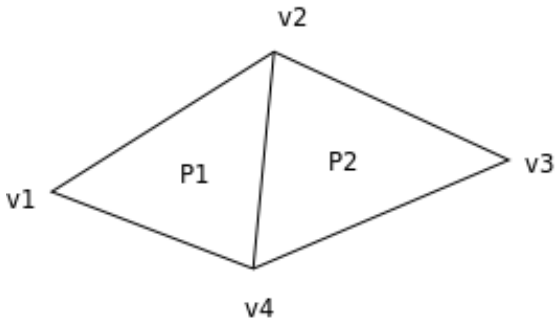


Figura 11.2: Objeto tridimensional simple de ejemplo

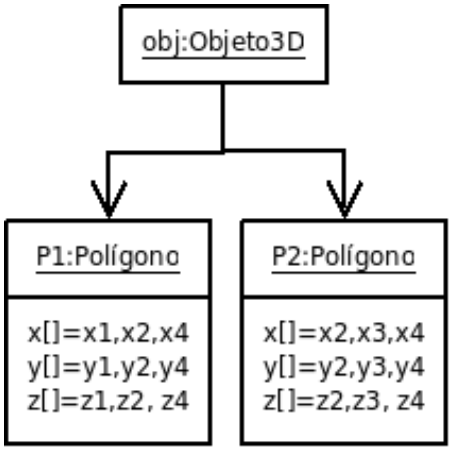


Figura 11.3: Diagrama de objetos del objeto de la figura 11.2

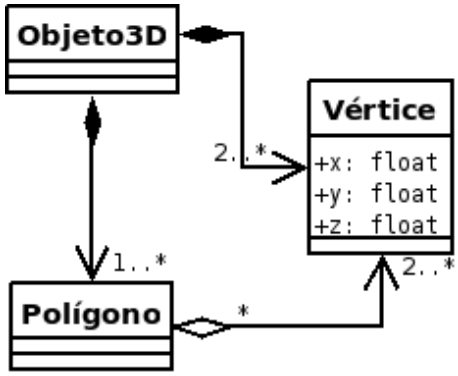


Figura 11.4: Diagrama de clases de una malla poligonal en representación de apuntadores a una lista de vértices

Es de notar que las coordenadas de los puntos – o vértices – $v2$ y $v4$ están repetidos en ambos polígonos, provocando una redundancia innecesaria. Imagine tal redundacia en un objeto altamente complejo, como el toroide de la figura 10.2 en la página 244. Además, como efecto colateral, se genera un grave problema en el caso de querer trasladar un punto, puesto que la búsqueda del punto se haría por igualdad entre tres pares de flotantes, lo cual es altamente riesgoso.

11.2. Apuntadores a una lista de vértices

En esta representación, los objetos tridimensionales se representan como una lista de polígonos y una lista de vértices. Cada vértice del objeto está sólo una vez en la lista de vértices, y cada polígono contiene una lista de apuntadores a los vértices que lo conforman.

El diagrama de clases correspondiente puede apreciarse en la figura 11.4.

Los objetos tridimensionales se modelan como una composición de uno o más polígonos y al mismo tiempo como una composición de varios vértices únicos. Cada polígono se modela como una secuencia lineal de referencias a vértices.

Esta representación resuelve el problema de la redundancia de información, pero presenta el problema siguiente: El proceso de dibujado sería recorrer todos los polígonos y dibujar todas las lineas de unión; pero las lineas – o aristas – que son compartidas por dos polígonos, se dibujarían dos veces (en realidad tantas veces como polígonos unan un mismo par de vértices). Entonces, para una figura compleja, habría que ejecutar el código de dibujo de líneas una alta cantidad de veces sin necesidad, porque ya habrían

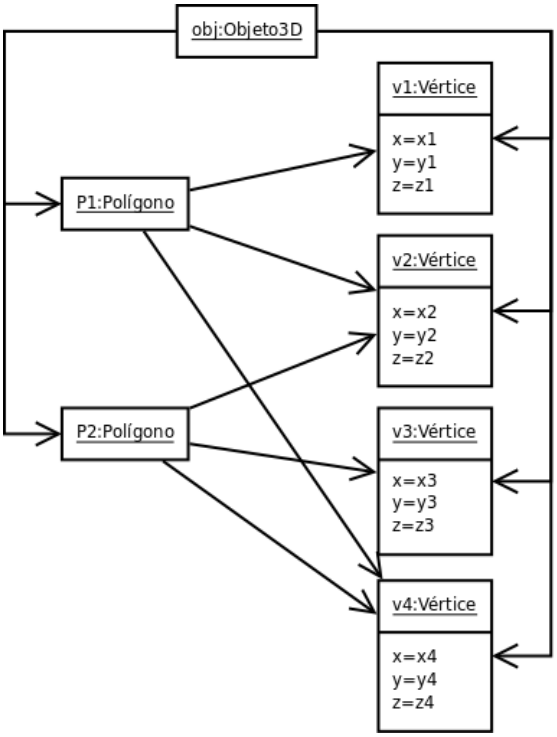


Figura 11.5: Otro diagrama de objetos del objeto de la figura 11.2

sido dibujadas. De hecho, la cantidad de líneas innecesariamente dibujadas ronda la mitad de todas las líneas.

Veamos en la figura 11.5 la representación en diagrama de objetos (de acuerdo al diagrama de clases de la figura 11.4) del objeto de la figura 11.2 en la página 248.

En este caso, al trasladar un punto − o alterarlo de cualquier manera −, se hace sólo una modificación y automáticamente todos los polígonos que incluyen tal punto estarán actualizados.

11.3. Apuntadores a una lista de aristas

En esta representación, los objetos tridimensionales se representan como una lista de polígonos, una lista de vértices y una lista de aristas. Cada polígono se representa como una lista de referencias a las aristas que lo conforman, y cada arista es un par de referencias a los vértices que unen.

El diagrama de clases correspondiente puede apreciarse en la figura 11.6.

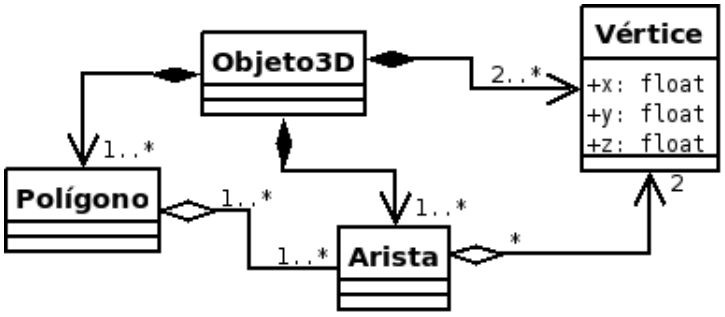


Figura 11.6: Diagrama de clases de una malla poligonal en representación de apuntadores a una lista de aristas

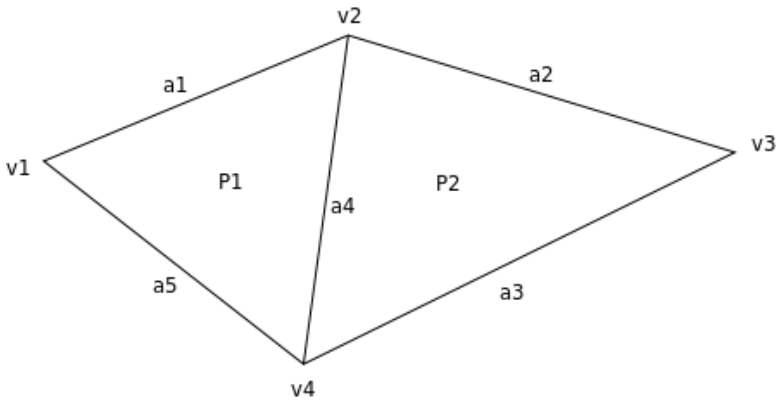


Figura 11.7: Objeto tridimensional de ejemplo para representación de apuntadores a una lista de aristas

Los objetos tridimensionales se modelan como una composición de uno o más polígonos, como una composición de varias aristas únicas y como una composición de varios vértices únicos. Cada polígono se modela como una secuencia lineal de referencias a las aristas que lo conforman. Cada arista contiene una referencia a los dos vértices que une y contiene una referenia a los polígonos a los que pertenece. Cada vértice por supuesto es único.

Veámos un ejemplo:

En la figura 11.7 se presenta el mismo objeto de la figura 11.2 en la página 248 pero con información sobre las aristas. Su representación en diagrama de objetos (de acuerdo al diagrama de clases de la figura 11.6) está en la figura 11.8.

En este caso, existe la manera de garantizar que cada línea – o arista – será dibujada sólo una vez, ya que el recorrido del algoritmo de dibujo puede hacerse sobre la lista de aristas y no sobre la de polígonos. Además no hay redundancia de vértices. Por otro

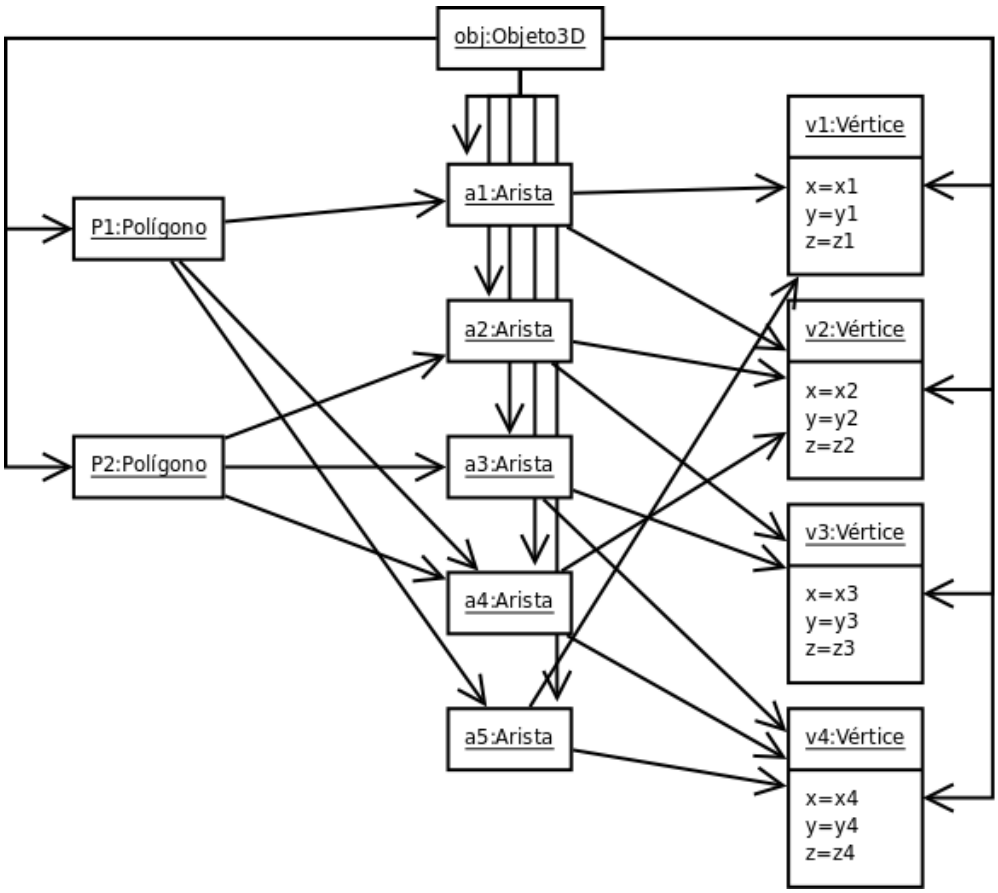


Figura 11.8: Diagrama de objetos del objeto de la figura 11.7

lado, gracias a la referencia a los polígonos a los que pertenecen las aristas, es posible discriminar algunos polígonos para que no sean dibujados, haciendo siempre el recorrido sobre la lista de aristas.

En la figura en la página siguiente se presenta un ejemplo con los detalles de implementación de este tipo de representación. En ese caso, las "listas" se implementan como listas circulares dobles con nodo de control.

11.4. Apuntadores sólo a una lista de aristas

Como se dijo al principio del capítulo, existen diversas maneras de modelar cuerpos tridimensionales y que el modelo a implementar depende de las necesidades concretas de las aplicaciones. Así, la estructura de datos usada para implementar las aplicaciones `transformaciones3D.jar` y `perspectiva3D.jar` de los capítulos 6 y 7 no es de ninguno de los tipos presentados anteriormente.

Dado que para esas aplicaciones no es relevante el concepto de polígono (ya que cada objeto tridimensional es simplemente una agrupación de vértices y aristas), tal clase de objetos no existe. En la figura 11.10 se presenta la estructura de datos usada en ese caso.

A continuación se presenta el código (en lenguaje java) que implementa dicha estructura de datos:

Listing 11.1: Código de Objeto3DSimple.java

```
1  /* c06/transformaciones3d/Objeto3DSimple.java
2   * Clases 3D básicas
3  */
4  public class Objeto3DSimple implements Config3D{
5      VerticeSimple puntos[];
6      AristaSimple aristas[];
7      boolean visible = true;
8
9      //Transformar todos los vértices mediante una matriz
10     //para ser proyectados en un portal de visión
11     public void transformarProyeccion(Matriz3d m, PortaldeVision portal){
12         for(int i=0; i<puntos.length; i++){
13             puntos[i].puntoProyectado.x = m.e[0][0] * puntos[i].puntoReal
                   .x + m.e[0][1] * puntos[i].puntoReal.y + m.e[0][2] *
                   puntos[i].puntoReal.z + m.e[0][3];
14             puntos[i].puntoProyectado.y = m.e[1][0] * puntos[i].puntoReal
                   .x + m.e[1][1] * puntos[i].puntoReal.y + m.e[1][2] *
                   puntos[i].puntoReal.z + m.e[1][3];
15             puntos[i].puntoProyectado.z = m.e[2][0] * puntos[i].puntoReal
                   .x + m.e[2][1] * puntos[i].puntoReal.y + m.e[2][2] *
                   puntos[i].puntoReal.z + m.e[2][3];
16         }
```

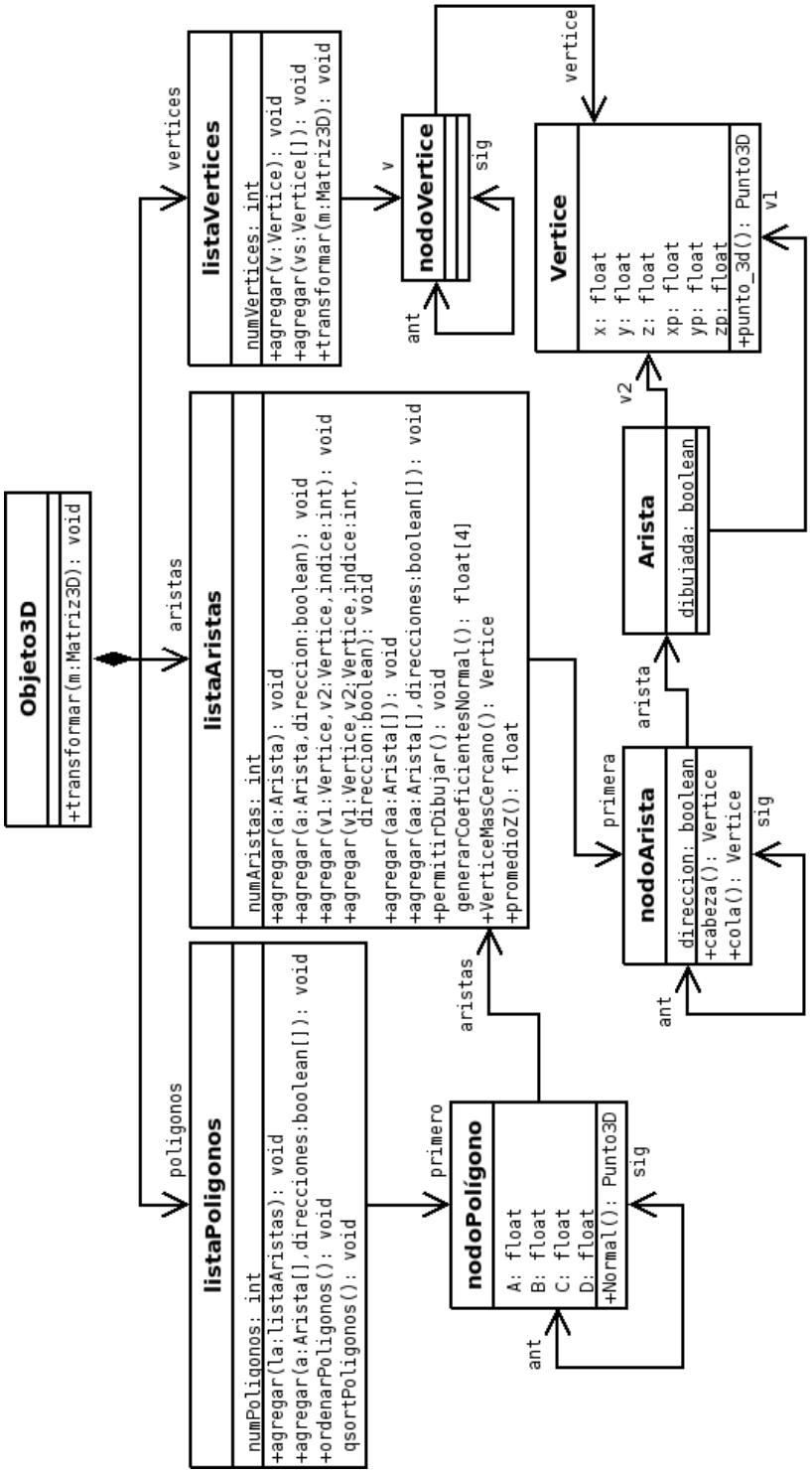


Figura 11.9: Detalles de implementación de una malla poligonal en representación de apuntadores a una lista de aristas

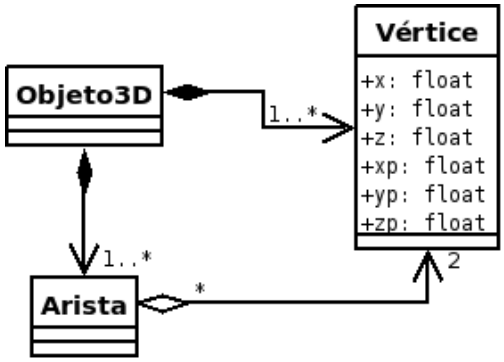


Figura 11.10: Diagrama de clases de las aplicaciones `transformaciones3D.jar` y `perspectiva3D.jar`

```
17      }
18
19      //  Transformar todos los vértices mediante una matriz
20      //para ser modificados en su universo virtual
21      public void transformar(Matriz3d m){
22          float nx, ny, nz;
23          for(int i=0; i<puntos.length; i++){
24              nx = m.e[0][0] * puntos[i].puntoReal.x + m.e[0][1] * puntos[i
                    ].puntoReal.y + m.e[0][2] * puntos[i].puntoReal.z + m.e
                    [0][3];
25              ny = m.e[1][0] * puntos[i].puntoReal.x + m.e[1][1] * puntos[i
                    ].puntoReal.y + m.e[1][2] * puntos[i].puntoReal.z + m.e
                    [1][3];
26              nz = m.e[2][0] * puntos[i].puntoReal.x + m.e[2][1] * puntos[i
                    ].puntoReal.y + m.e[2][2] * puntos[i].puntoReal.z + m.e
                    [2][3];
27              puntos[i].puntoReal.x = nx;
28              puntos[i].puntoReal.y = ny;
29              puntos[i].puntoReal.z = nz;
30          }
31      }
32
33  }
34
35  class VerticeSimple {
36      Punto3d puntoReal, puntoProyectado;
37      public VerticeSimple(){
38          puntoProyectado = new Punto3d(0f,0f,0f);
39      }
40  }
41  class AristaSimple {
42      VerticeSimple punto1, punto2;
```

```
43      java.awt.Color color;
44  }
```

11.5. Ejercicios

1. Construya un algoritmo que genere una superficie de malla poligonal cilíndrica (Obviamente hay que elegir la representación que tendrá).
2. Construya un algoritmo que genere una superficie de malla poligonal esférica (se sugiere considerar una división en meridianos y paralelos). La finura de la malla deberá ser controlada por parámetros de entrada al algoritmo.

12 Introducción a los Fractales

En este capítulo no haremos un estudio formal de la matemática involucrada con los fractales, sino más bien un breve recorrido por algunas familias de fractales que son *relativamente* fáciles de graficar.

12.1. Características de los fractales

En realidad no existe una definición específica de *Fractal*, sino más bien, un conjunto de características asociadas a tal definición. De tal manera que cuando algo tiene algunas de esas características[1], se dice que ese algo es un fractal.

Las principales características son las siguientes:

- Tener una intrincada (y aparentemente sofisticada) geometría, tal que no puede ser descrito en términos de geometría euclideana normal.
- Poseer el mismo nivel de detalle a cualquier escala.
- Tener una descripción geométrica recursiva.
- Tener "autosimilitud", determinística o probabilística en su apariencia. O sea, que el todo sea igual o muy parecido a una de sus partes.
- Tener una dimensión de Hausdorff-Besicovitch mayor que la propia dimensión topológica.

Veamos algunos ejemplos de objetos con esas características en la naturaleza, en las figuras 12.1[2], 12.2 y 12.3.

Ahora veamos algunas figuras fractales generadas por computadora. La 12.4 muestra un helecho que efectivamente parece real. Las figuras 12.5[3] y 12.6[4] muestran perfectamente la característica de la autosimilitud. La 12.7 muestra "relámpagos fractales" a partir del conjunto de Mandelbrot.

[1] al menos dos de ellas
[2] Ruta completa:
http://www.ubcbotanicalgarden.org/potd/2006/02/brassica_oleracea_botrytis_group_romanesco.php
[3] Fuente: http://commons.wikimedia.org/wiki/User:Wolfgangbeyer
[4] Realizado con el programa libre XaoS: http://xaos.sf.net/

Figura 12.1: Fractal natural: *Brassica oleracea*, un Romanescu fresco, cortesía del programa *Botany Photo of the Day* de http://www.ubcbotanicalgarden.org/.

Figura 12.2: Las ramas de los árboles siguen leyes fractales de distribuión volumétrica.

Figura 12.3: Las hojas de casi todos los helechos tienen la característica de la autosimilitud finita.

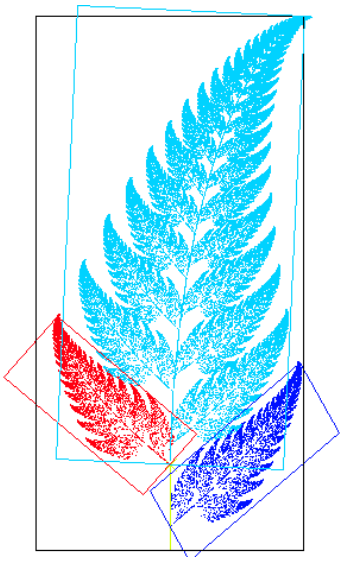


Figura 12.4: Helecho fractal

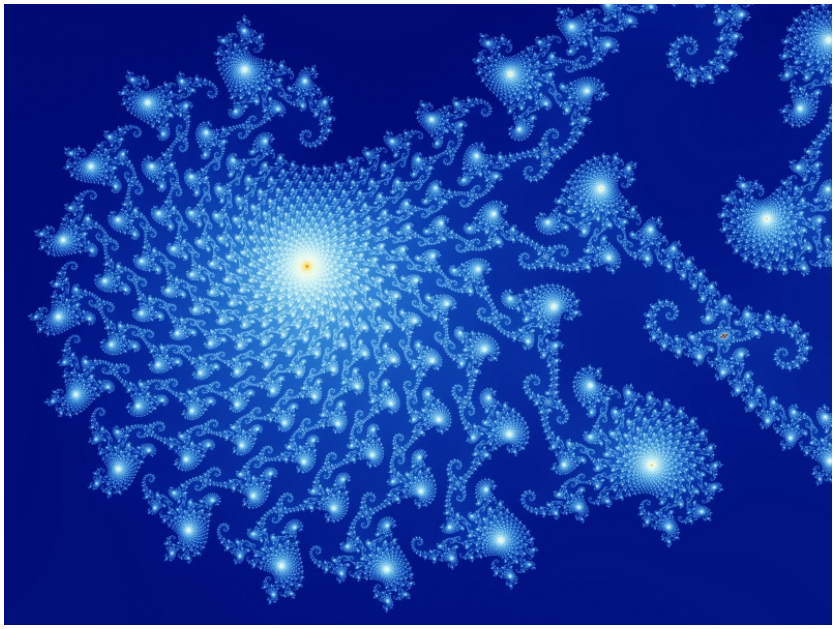


Figura 12.5: Espiral de satélites con islas de Julia, cortesía del Dr. Wolfgang Beyer

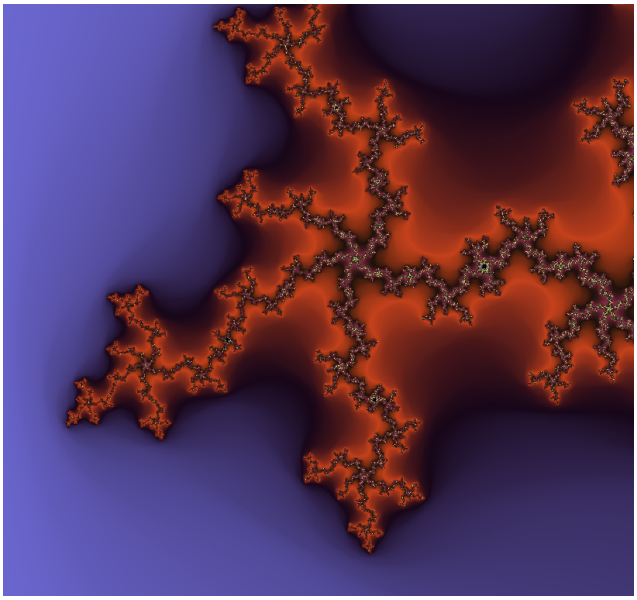


Figura 12.6: Acercamiento del conjunto de Mandelbrot realzado por el autor de este libro con el programa XaoS.

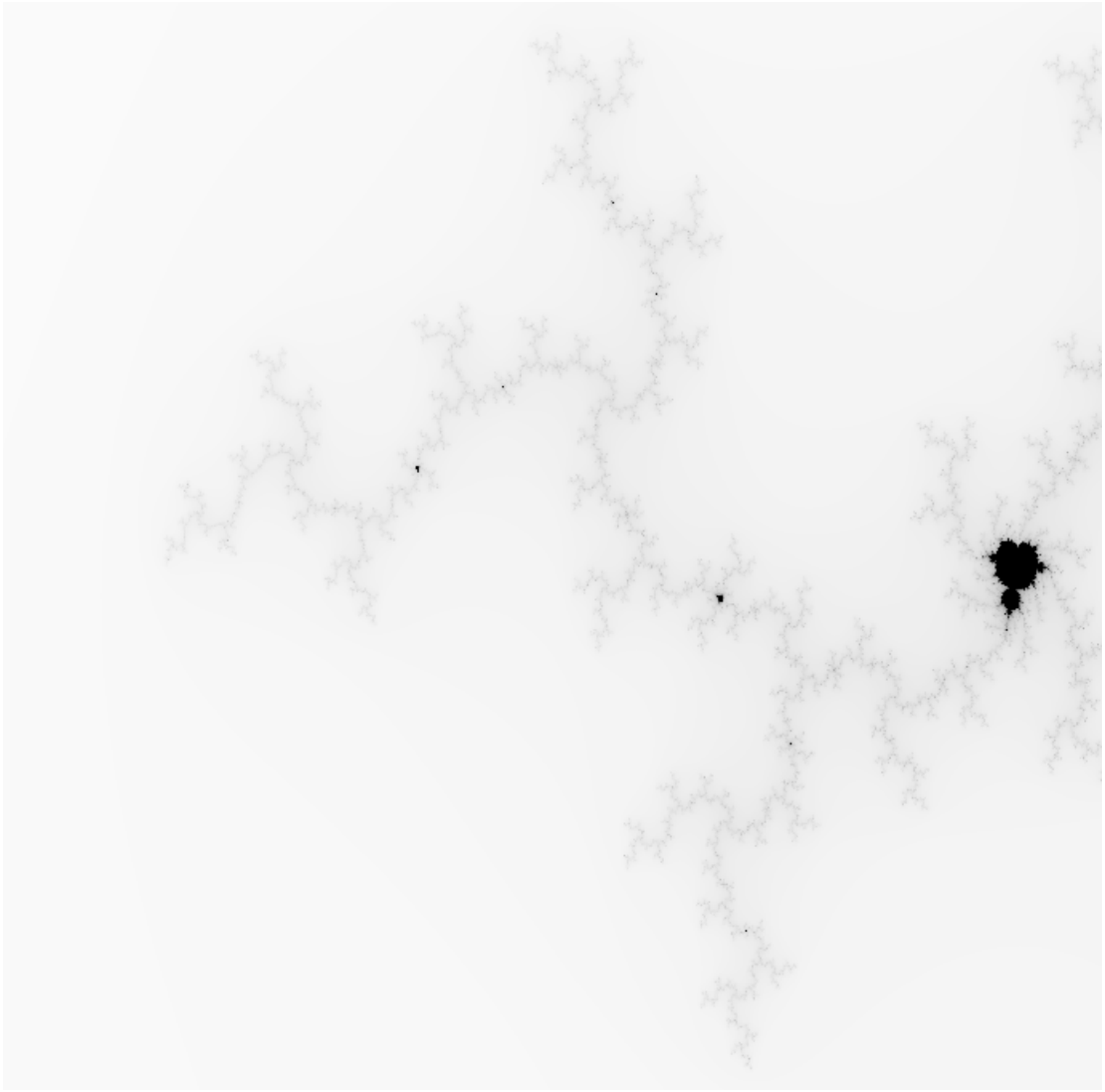


Figura 12.7: Otro acercamiento del conjunto de Mandelbrot realizado con la aplicación `mandelbrot.out` presentada más adelante en este capítulo.

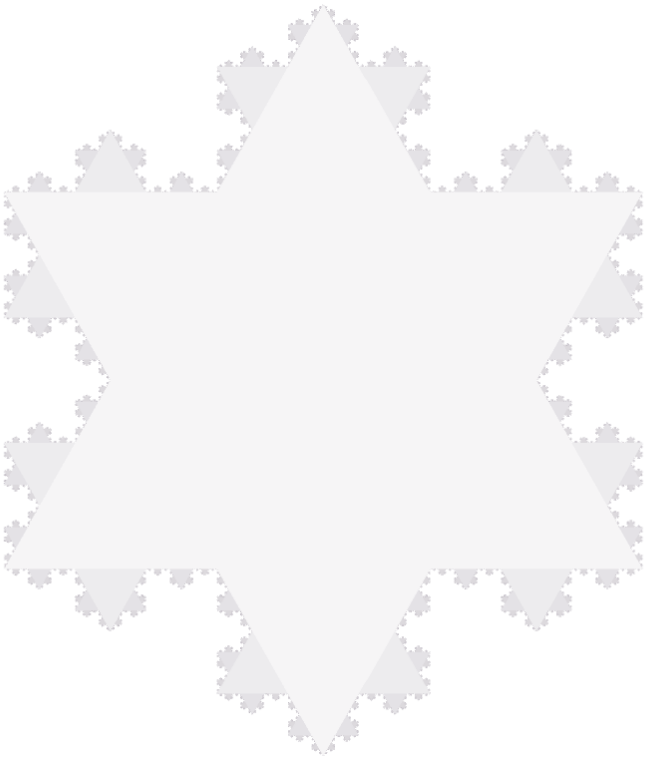


Figura 12.8: Copo de nieve de von Koch

12.2. El copo de nieve de *von Koch*[5]

El copo de nieve de **Niels Fabian Helge von Koch**, es una figura sencilla que exhibe las características de estar geométricamente definido por un algoritmo recursivo y por que su dimensión de Hausdorff-Besicovitch es mayor que su dimensión topológica. La explicación formal de esto último está fuera del alcance actual de esta obra, pero podemos enunciar la consecuencia directa de tal formalismo: *La longitud de la curva es infinita, pero el área que encierra es finita.*

Podemos apreciar una representación hecha con la aplicación XaoS mencionada anteriormente en la figura 12.8.

[5] léase "fon koj"

12.3. El triángulo de *Sierpiński*[6]

Presentamos brevemente las famosas figuras de **Wacław Sierpiński**, el triángulo de Sierpiński y la carpeta de Sierpiński. Estas pueden verse en las figuras 12.9 y 12.10. Las ideas básicas de estas figuras planas se pueden extender para figuras tridimensionales, como vemos en la figura 12.11 en la página 266.

12.4. Los Conjuntos *Julia-Fatou*

12.4.1. Definición

Llamados así, en honor de **Gaston Julia** y **Pierre Fatou**, los conjuntos *Julia* no son figuras concretas, sino *una familia de figuras fractales*, con todo el esplendor de la expresión. Se obtienen al analizar el acotamiento de ciertas funciones recursivas en el dominio de $\mathbb{C}$ (sí, de los números complejos).

El conjunto *Julia* de una función $f_c(z)$ con semilla $c \in \mathbb{C}$, denotado por $J_c(f)$, es el conjunto de todos los valores z, tales que la siguiente sucesión sea acotada:

$$\begin{cases} z_0 = z \\ z_{n+1} = f_c(z_n) \end{cases}$$

Típicamente se calculan los conjuntos $J_c(f)$ con $f_c(z) = z^2 + c$.

Por otro lado, el conjunto *Fatou*, $F_c(f)$ es el complemento de $J_c(f)$, $F_c(f) = \mathbb{C} - J_c(f)$. Es decir, $F_c(f)$ contiene todos los z para los que la sucesión antes descrita, no es acotada.

Ya encaminándonos a la implementación, se puede demostrar que si $|z_n| > 2$ entonces la sucesión no es acotada y $z \notin J_c(f)$. Ese es el critero a usar para saber si la sucesión diverge. Y si no es acotada, no se llegará al valor de 2, por lo que debe haber un número máximo de iteraciones a evaluar. Si el valor de n llega a un cierto límite sin pasar de 2, consideraremos que dicha sucesión es acotada. Vale recalcar que mientras mayor sea el n máximo, más nos acercaremos al conjunto real (el cual, por supuesto, es imposible de alcanzar).

A continuación presentamos algunas imágenes del programa `julia.out`. En el título de las ventanas aparecen los valores c usados como semilla. Son las figuras 12.12, 12.13, 12.14 y 12.15.

[6] léase "sierpiñski"

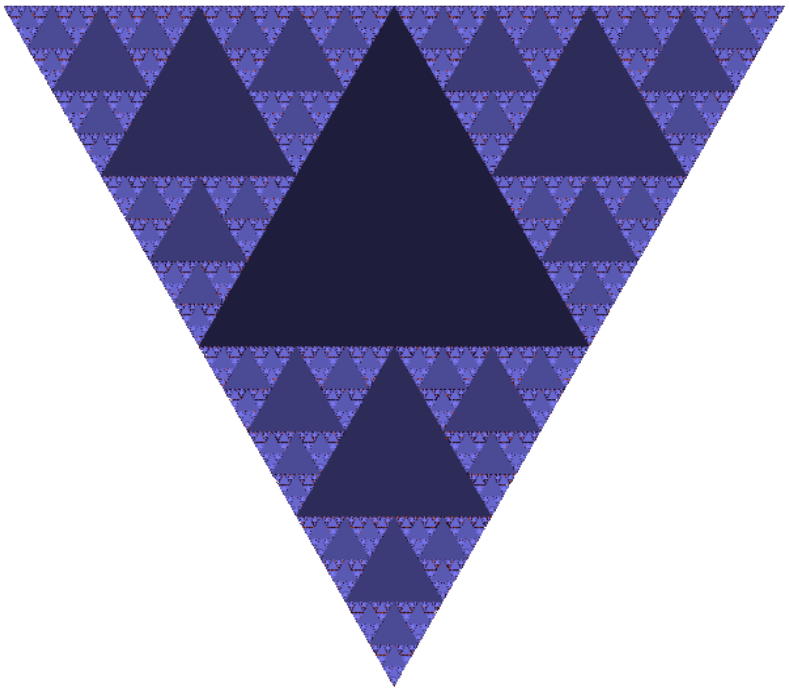


Figura 12.9: Triángulo de Sierpiński

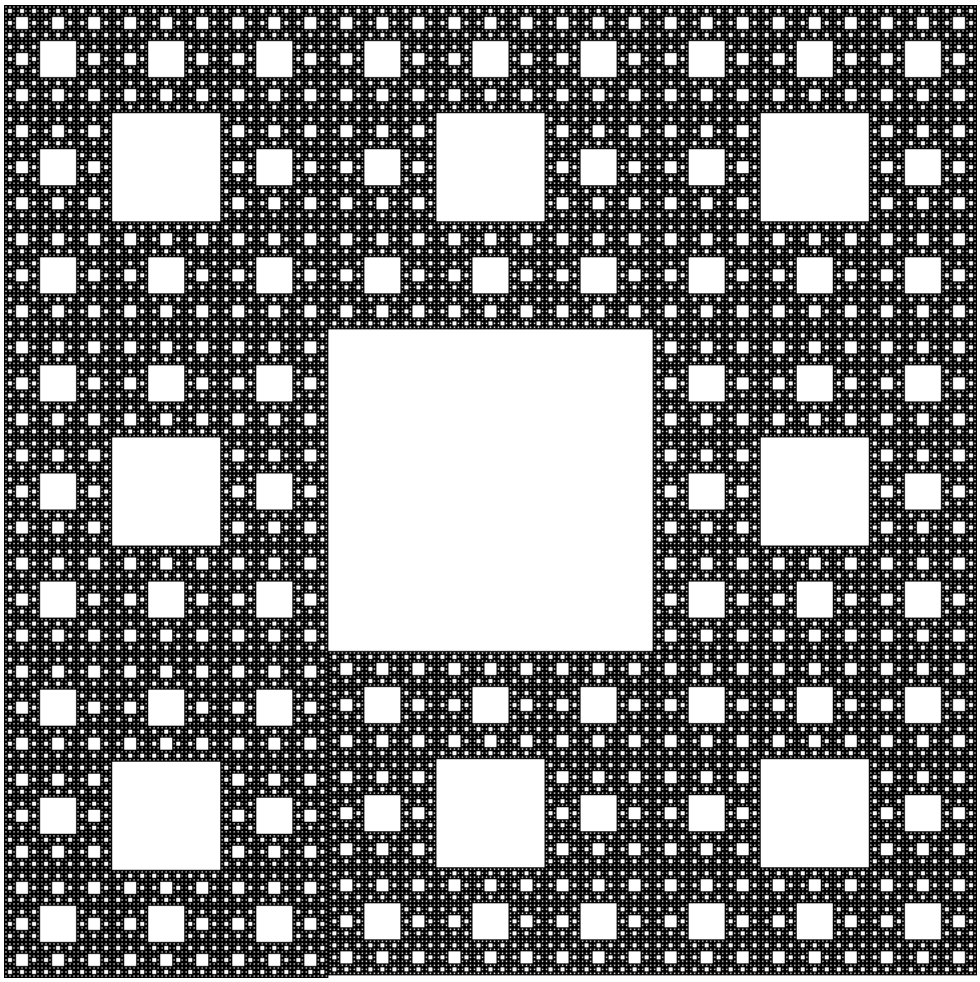


Figura 12.10: Carpeta de Sierpiński

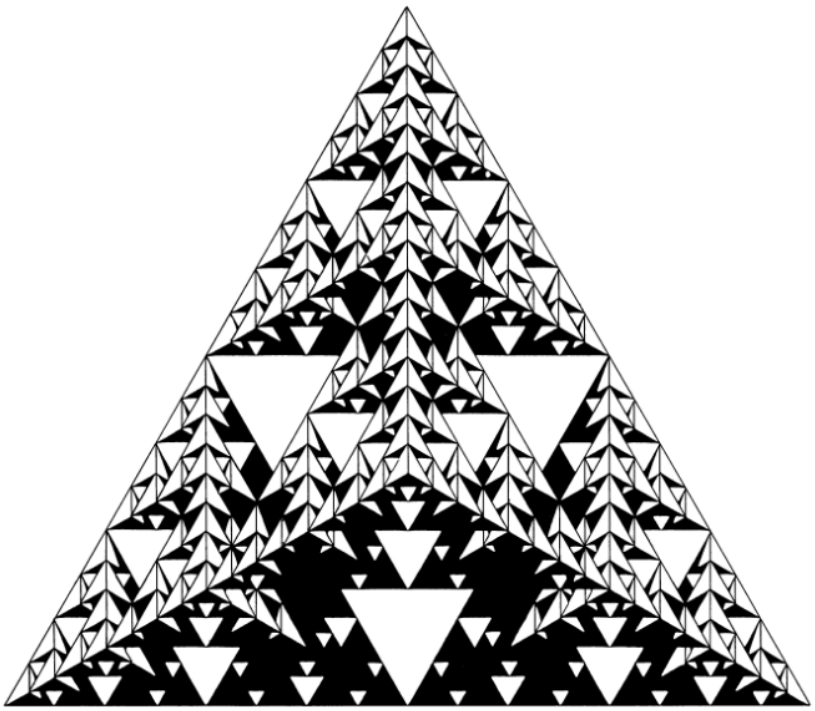


Figura 12.11: Pirámide de Sierpiński

Figura 12.12: Imágen del programa `julia.out`

Figura 12.13: Segunda imágen del programa `julia.out`

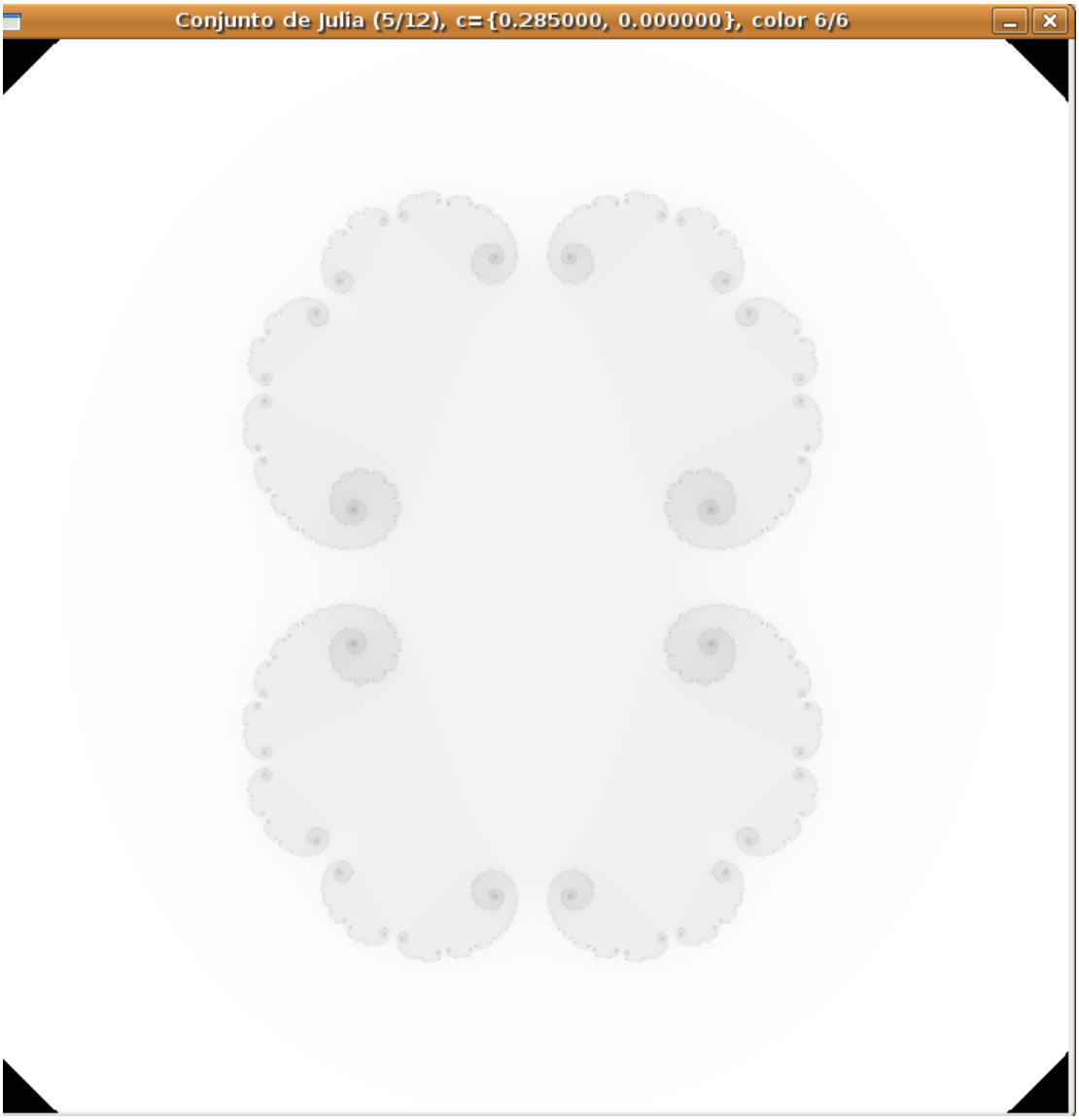


Figura 12.14: Tercera imágen del programa `julia.out`

Figura 12.15: Cuarta imágen del programa `julia.out`

12.4.2. Implementación

En el material adjunto a este libro se encuentra la aplicación `julia.out`, cuyo funcionamiento se describe a continuación:

clic izquerdo El primer clic, define una de las esquinas de un rectángulo que será usado como área de aumento. El segundo clic, define la esquina opuesta del rectángulo y se efectúa el aumento correspondiente.

clic derecho Cambia la semilla usada para calcular el conjunto y regresa la escala a sus valores por defecto. Actualmente el programa contiene 12 semillas diferentes (algunas no se aprecian bien con algunos esquemas de color). El valor de la semilla usada se muestra en la barra de título de la ventana.

rueda del ratón Cambia el algoritmo de coloreado para el conjunto actual con la escala actual. Por el momento hay 6 algoritmos de coloreado. El índice del algoritmo aparece en la barra de título de la ventana gráfica con el formato "`color i/6`".

Debido a la alta complejidad del algoritmo que decide si cada pixel pertenece o no al conjunto, la respuesta de la aplicación no es inmediata. Dependiendo del procesador en el que se ejecute, la aplicación puede ser un poco lenta para responder a la rueda del ratón.

A continuación se presenta una de las funciones de dibujo de la aplicación mencionada (tomada del archivo `calculos.c`):

Listing 12.1: Función de dibujo de Conjunto de Julia

```
165 void dibujarFractalJulia6(SDL_Surface *pantalla, complejo *c, escala *e){
166     int i, j;
167     complejo z0, z;
168     int numPasos;
169 
170     for(i=0; i<e->anchoReal; i++){
171         z0.i = tV_Ry(e, i);
172 
173         for(j=0; j<e->altoReal; j++){
174             z0.r = tV_Rx(e, j);
175 
176             z = z0;
177             numPasos = 0;
178             while(numPasos<MAXPASOS && (sq(z.r)+sq(z.i)<LIMITEMODULO2)){
179                 CX_suma(CX_cuadrado(&z), c);
180                 numPasos++;
181             }
182             if(numPasos==MAXPASOS){
183                 pixelColor(pantalla, j, i, COLOR_ADENTRO);
184             }
185             else{
```

```
186                     pixelColor(pantalla, j, i, colorGfx(
187                         (MAXPASOS-numPasos)*256/MAXPASOS,
188                         (MAXPASOS-numPasos)*256/MAXPASOS,
189                         (MAXPASOS-numPasos)*256/MAXPASOS));
190                 }
191             }
192         }
193 }
```

12.5. Conjunto de *Mandelbrot*

12.5.1. Definición

En honor a **Benoît Mandelbrot**, es un conjunto específico, bien definido, estrechamente relacionado con los conjuntos *Julia*.

El conjunto *Mandelbrot*, denotado por M, es el conjunto de todos los valores $c \in \mathbb{C}$ tales que la siguiente sucesión sea acotada:

$$\begin{cases} z_0 = 0 + 0i \\ z_{n+1} = {z_n}^2 + c \end{cases}$$

Y se utilizan los mismos criterios de selección que para los conjuntos Julia.

Presentamos la forma clásica de este famoso conjunto fractal en la figura 12.16 y una versión estilizada con XaoS en la figura 12.17.

12.5.2. Implementación

En el material adjunto a este libro se encuentra la aplicación `mandelbrot.out`, cuyo funcionamiento se describe a continuación:

clic izquerdo El primer clic, define una de las esquinas de un rectángulo que será usado como área de aumento. El segundo clic, define la esquina opuesta del rectángulo y se efectúa el aumento correspondiente.

clic derecho Cambia el algoritmo de coloreado para el conjunto actual con la escala actual. Por el momento hay 8 algoritmos de coloreado. El índice del algoritmo aparece en la barra de título de la ventana gráfica con el formato "`color i/8`".

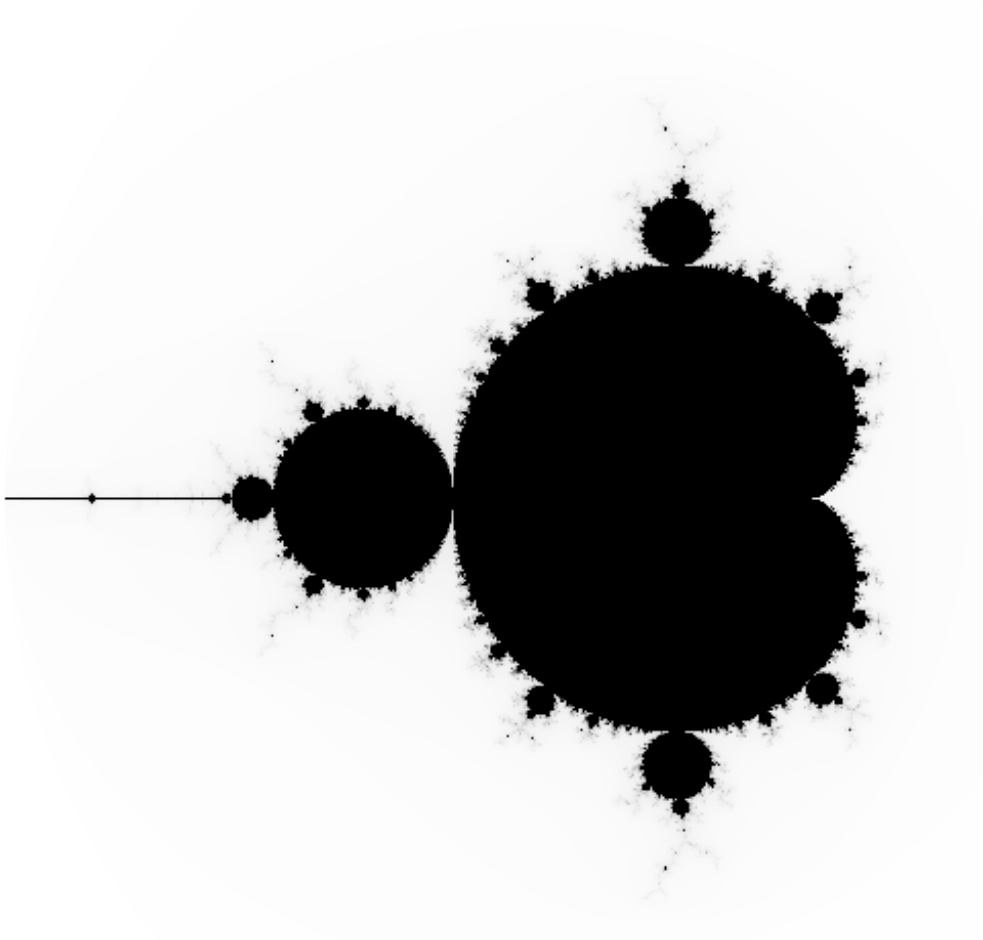


Figura 12.16: Forma clásica del conjunto Mandelbrot, generado con `mandelbrot.out`

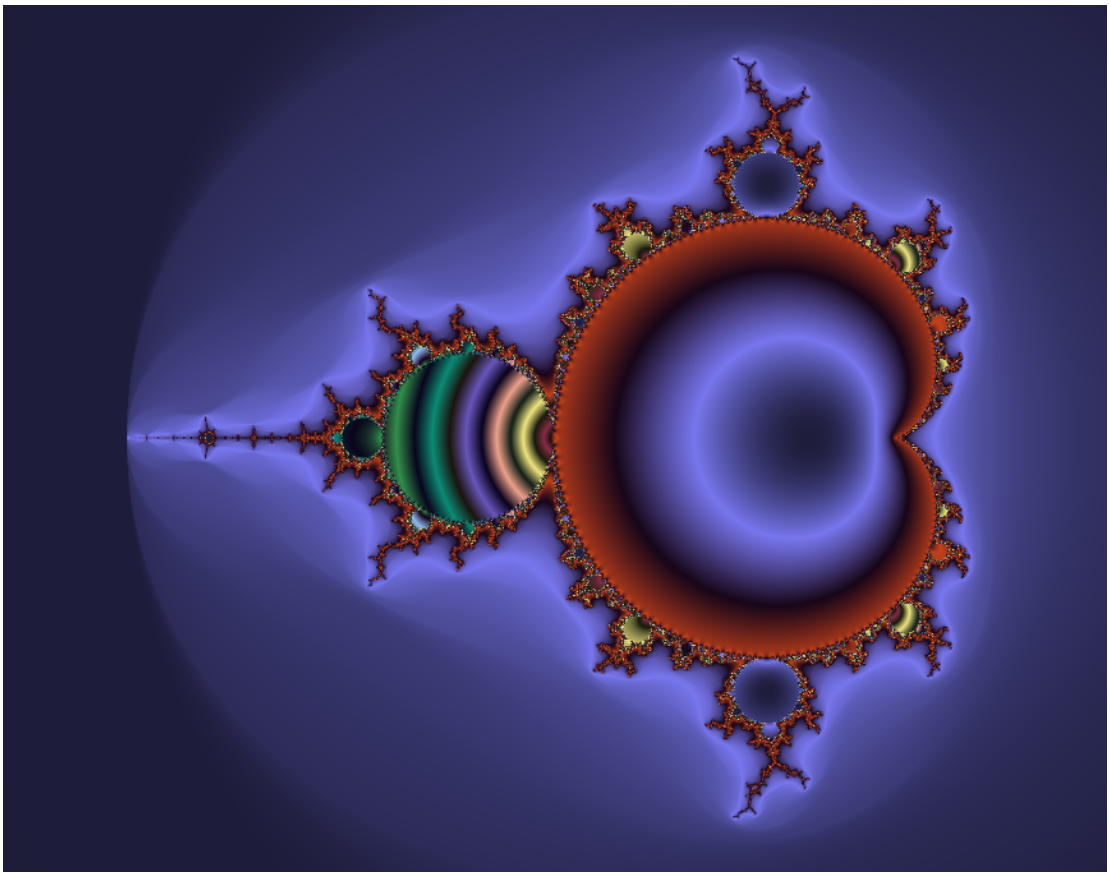


Figura 12.17: Conjunto Mandelbrot con "suavización" de color interna y externa, generado con la aplicación XaoS

El algoritmo que decide si cada pixel pertenece o no al conjunto de Mandelbrot es de la misma complejidad que el del caso de los conjuntos de Julia, por lo que la respuesta de la aplicación tiene en general, la misma velocidad.

A continuación se presenta una de las funciones de dibujo de la aplicación mencionada (tomada del archivo `calculos.c`):

Listing 12.2: Función de dibujo de Conjunto de Mandelbrot

```
371 void dibujarFractalMandelbrot7(SDL_Surface *pantalla, escala *e){
372     int i, j;
373     complejo z={0.0, 0.0}, c;
374     int numPasos;
375 
376     for(i=0; i<e->anchoReal; i++){
377         c.i = tV_Ry(e, i);
378 
379         for(j=0; j<e->altoReal; j++){
380             c.r = tV_Rx(e, j);
381 
382             numPasos = 0;
383             z.r = z.i = 0.0;
384             while(numPasos<MAXPASOS && (sq(z.r)+sq(z.i) < LIMITEMODULO2))
                    {
385                 CX_suma(CX_cuadrado(&z), &c);
386                 numPasos++;
387             }
388             if(numPasos==MAXPASOS){
389                 pixelColor(pantalla, j, i, COLOR_ADENTRO);
390             }
391             else
392                 pixelColor(pantalla, j, i, colorGfx(
393                     numPasos*256/MAXPASOS,
394                     (MAXPASOS-numPasos)*256/MAXPASOS,
395                     numPasos*256/MAXPASOS));
396         }
397     }
398 }
```

12.6. Ejercicios

1. Investigue cuál es la relación entre M y $J_c(f)$.
2. Construya un algoritmo que dibuje el fractal de la figura 12.18.
3. Construya un algoritmo que dibuje el fractal de la figura 12.19.

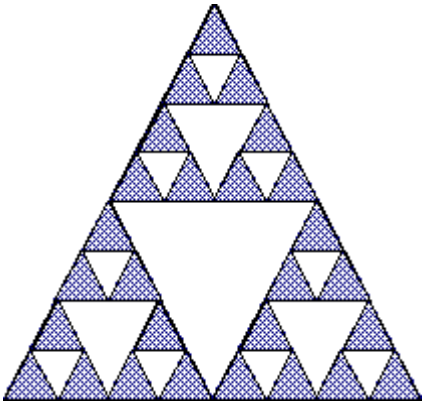


Figura 12.18: Fractal de ejercicio

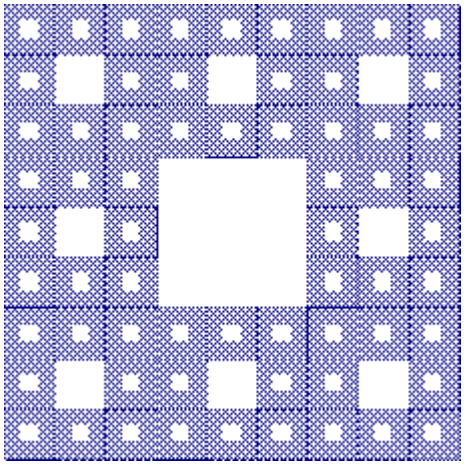


Figura 12.19: Fractal de otro ejercicio

Parte II

Otras Yerbas

13 Compilación desde Múltiples archivos fuente (en lenguaje C)

Supongamos que tenemos nuestro código separado en diversos archivos fuente, tal como el siguiente ejemplo:

Listing 13.1: Programa principal

```
/* c13/principal.c
 * */
#include "otro.h"

int main(int argc, char *argv[]){
    funcion();
    return 0;
}
```

Listing 13.2: Cabecera de otro código

```
/* c13/otro.h
 * */
int funcion(void);
```

Listing 13.3: Otro código fuente

```
/* c13/otro.c
 * */
#include "otro.h"
#include <stdio.h>

int funcion(void){
    printf("hola␣a␣todos␣y␣todas\n");
    return 0;
}
```

El programa es muy simple, su salida es completamente previsible, por lo que es perfecto para ilustrar cómo podemos apoyarnos en la utilería `make` para compilarlo de forma automática:

Debemos crear en ese mismo directorio un archivo `Makefile` para orientar a `make`. El archivo de ayuda a la compilación debería contener algo parecido a lo siguiente:

Listing 13.4: Makefile para varios archivos fuente

```
# c13/Makefile

#Esto es un comentario

#El comando para borrar archivos
RM = /bin/rm -f

#Un '*.o' por cada '*.c' que pertenezca al proyecto
OBJS = principal.o otro.o

#Nombre del programa ejecutable:
PROG = programa

#Esto indica que los siguientes identificadores,
#no son archivos, sino comandos de make:
.PHONY: limpiar
.PHONY: limpiartodo
.PHONY: all
#se puede, por ejemplo, ejecutar en la consola
#lo siguiente:
#'$ make limpiartodo', etc.

#Cuando se ejecuta '$ make', se evalúan
# las reglas '$(PROG)' y 'limpiar':
all: $(PROG) limpiar


#Esta regla compila todo el código y lo enlaza:
$(PROG): $(OBJS)
        gcc -o $(PROG) $(OBJS)

#Esta regla borra todos los archivos intermedios
# y de copia de seguridad:
limpiar:
        $(RM) *~ $(OBJS)

#Esta regla borra todos los archivos intermedios
# y el programa ejecutable, si es que existe
limpiartodo:
        make limpiar
        $(RM) $(PROG)
```

La presencia de dicho archivo y su contenido nos permiten ejecutar las siguientes órdenes en esa carpeta:

- `$ make limpiar`
  Borra los archivos de copia de seguridad que se hayan creado y también borra los archivos de código objeto intermedios (los `*.o`) que se crean al compilar los

respectivos archivos de código fuente (los `*.c`).

- `$ make limpiartodo`
  Invoca la instrucción anterior y además borra el programa ensamblado o ejecutable (su nombre depende de lo que hayamos puesto en la variable `PROG`.
- `$ make`
  Se ejecuta lo que hayamos indicado en la regla `all`. En este caso, invoca la regla `PROG`, que a su vez invoca la compilación individual de cada archivo fuente (los `*.c`) y posteriormente invoca el ensamblaje de estos con el comando `gcc`. Posteriormente invoca la regla `limpiar`.

Esta pequeña guía no pretende ser altamente exhaustiva. Simplemente pretende orientar para la compilación asistida en proyectos de programación en lenguaje C estándar de mediana escala en ambientes tipo UNIX. Si se necesita mayor detalle o explicación, por favor refiérase el lector a la documentación apropiada. Por ejemplo:

```
$ man make
```

O en los sitios siguientes:

http://www.chuidiang.com/clinux/herramientas/makefile.php
http://www.calcifer.org/documentos/make/makefile.html
http://atc1.aut.uah.es/~lsotm/Makefile.htm
http://en.wikipedia.org/wiki/Make_(software)
http://www.opussoftware.com/tutorial/TutMakefile.htm

14 Diagramas de Clases y Diagramas de Objetos (una untadita de UML)

Haremos en este capítulo un breve resumen de la notación UML (Unified Modeling Language) para diagramas de clases y objetos.

14.1. Notación de clases

En UML una clase se representa como un rectángulo con tres espacios. En el primero va el nombre de la clase, en el segundo sus atributos y en el tercero sus operaciones. Veamos un ejemplo en la figura 14.1.

La clase se llama **Clase**, tiene dos atributos y dos operaciones. Los atributos son *i*, un entero, y *h*, un flotante con valor por defecto de 4.0. El atributo *i* es público y *h* es privado. Eso indican los signos que preceden a los nombres.

Que un atributo sea privado significa que su ámbito de acceso está limitado al interior del código de la clase, y si es público, significa que se puede acceder a él desde cualquier ámbito desde el que se pueda alcanzar una instancia de esta clase.

La primera operación es privada, se llama *metodo*, retorna *void* y recibe un parámetro llamado *para1* de tipo **Clase** (sí, del mismo tipo). La segunda función se llama *funcion*, retorna un *String* y recibe dos parámetros: *semilla* que es entero con valor por defecto de 3, y *objeto* de tipo *Estructura*.

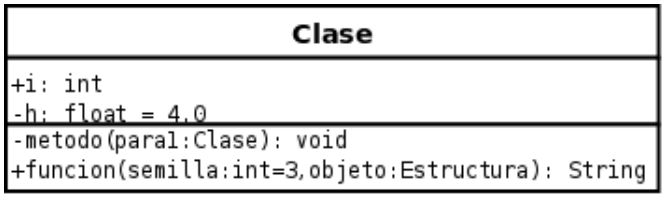


Figura 14.1: Representación de una clase

Figura 14.2: Diagrama de clase ocultando sus atributos y operaciones

Los niveles de acceso público y privado son los más usuales en los lenguajes de programación orientados a objetos, pero algunos lenguajes definen otros niveles de acceso. En el caso de Java, también existen los niveles de acceso *protegido* y de *paquete*, representados por los signos "#" y "~" respectivamente.

Por otro lado, la información de los atributos y de las operaciones de las clases a veces no es relevante o no es conveniente mostrarla (generalmente por cuestiones de espacio), por lo que pueden omitirse esas secciones y mostrar únicamente un rectángulo con el nombre de la clase, como vemos en la figura 14.2.

14.2. Notación de relaciones o asociaciones

En la figura 14.3 se presenta un resumen de la notación básica de asociaciones en diagramas de clases. Veamos cada uno de ellos:

1. Indica simplemente que hay una relación uno-a-uno entre una instancia de **Clase1** y una de **Clase2**.
2. Indica que hay una relación uno-a-uno entre las instancias, pero agrega semántica a la relación: indica que las instancias de **Clase1** "tienen" una instancia de la clase **Clase2**.
3. Indica que hay una relación uno-a-uno entre las clases, y el nombre de la relación, pero sin indicar la dirección de la relación.
4. Indica que una instancia de **Clase1** puede tener referencias a ningúna, una o muchas instancias de **Clase2** y que una instancia de **Clase2** tiene forzosamente 2 referencias a instancias de **Clase1**.
   Estos elementos que indican cantidad de referencias se llaman "*multiplicidades*".
5. Indica que una instancia de **Clase1** debe tener al menos una referencia a instancias de **Clase2** y que una instancia de **Clase2** tiene forzosamente entre 2 y 4 referencias a instancias de **Clase1**.
6. Indica que hay una relación uno-a-uno, pero indicando que para las instancias de **Clase1**, la referencia a la instancia de **Clase2** se llama *objetoUtil* y es privada. Dice además, que para las instancias de **Clase2**, la referencia a la instancia de **Clase1** se llama *apuntado* y es pública.
   Estos nombres se conocen como "*roles*".

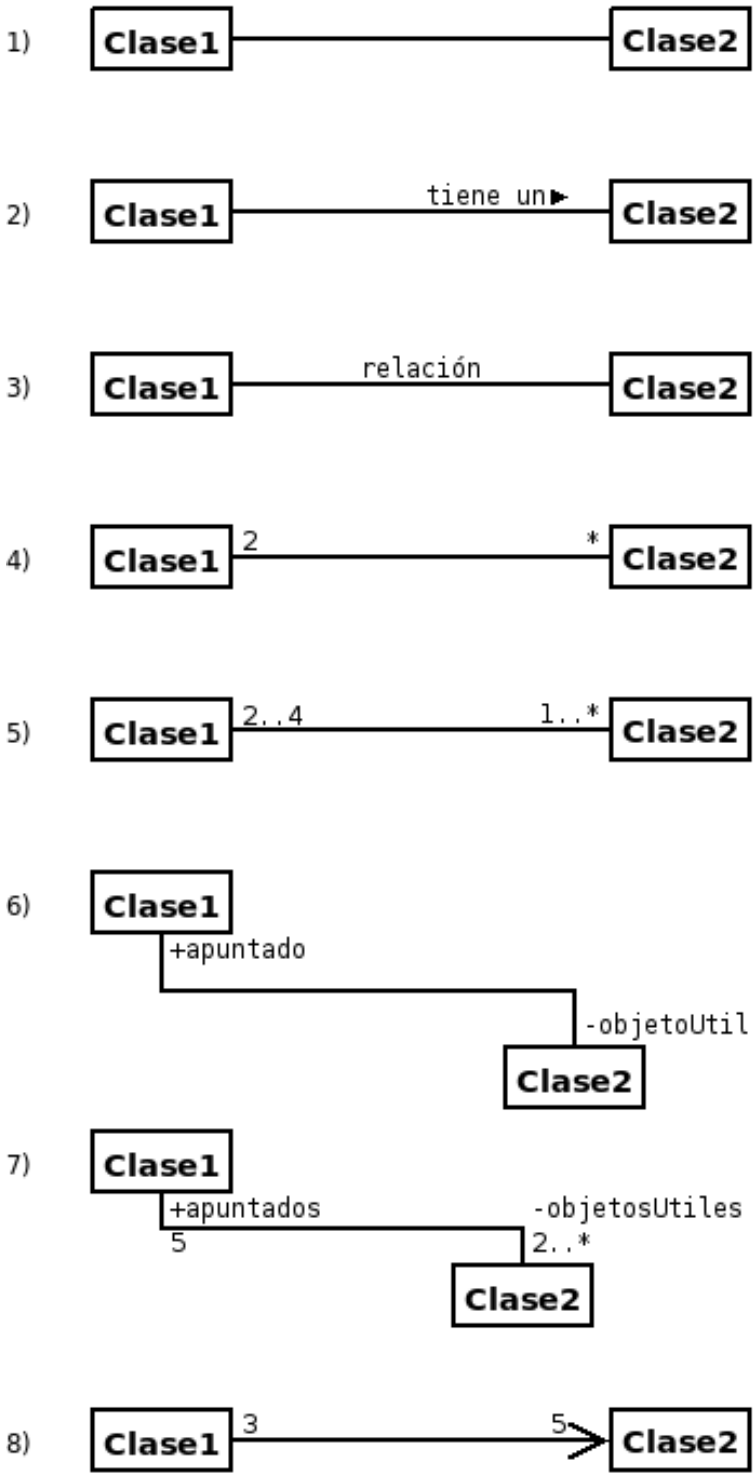


Figura 14.3: Relaciones bidireccionales entre clases

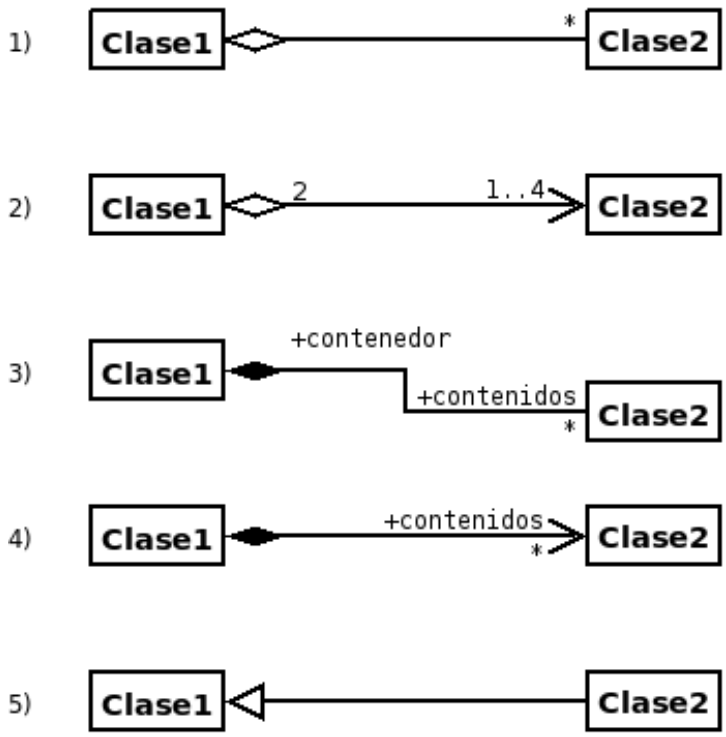


Figura 14.4: Otros tipos de relación entre clases

7. Indica que las instancias de **Clase1** deben tener una colección de referencias a al menos 2 instancias de **Clase2** y que esa colección se llama *objetosUtiles*. Dice además, que las instancias de **Clase2** tienen una colección de 5 referencias a instancias de **Clase1** que se llama *apuntados*. Indica además el nivel de acceso de dichas colecciones.
8. Indica que una instancia de **Clase1** tiene 5 referencias a instancias de **Clase2**, y que las instancias de **Clase2** son referenciadas por 3 instancias de **Clase1**. También dice que las instancias de **Clase2** no tienen referencias a las instancias relacionadas de **Clase1**.

Los diagramas de clase no obligan al diseñador a especificar cómo, en concreto, se implementarán tales referencias o colecciones de referencias. Se podrían implementar con arreglos, con listas, árboles, grafos, etc.

En la figura 14.4 se presentan más tipos de asociaciones en diagramas de clases:

1. Indica una "agregación", en la que las instancias de **Clase1** están formadas (entre otras cosas) por una colección de referencias a algunas instancias de **Clase2**, y estas tienen una referencia a la instancia de **Clase1** a la cual están agregadas.

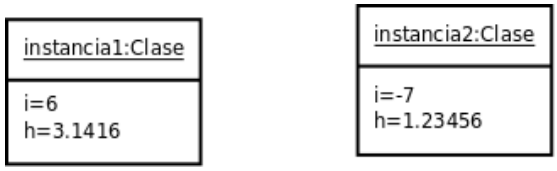


Figura 14.5: Representación de una instancia de una clase

También indica que si la instancia "agregada" de **Clase1** deja de existir, las instancias de **Clase2** de las cuales estaba fomada, no tienen por qué dejar de existir.

2. Indica una "agregación" en la que las instancias de **Clase1** están formadas (entre otras cosas) por una colección de entre 1 y 4 referencias a instancias de **Clase2**, y que estas conforman simultaneamente dos instancias de **Clase1**, pero no contienen referencia a ellas.
3. Indica una "composición", que es una agregación muy restrictiva, ya que indica que si la instancia "compuesta" de **Clase1** deja de existir, las instancias de **Clase2** que la componían, deben también dejar de existir. En consecuencia, las instancias de **Clase2** sólo pueden "componer" una instancia de **Clase1**.
4. Esta es una "composición" en la que las instancias de **Clase2** no tienen referencia a la instancia de **Clase1** que componen.
5. Indica que la clase **Clase2** hereda de **Clase1**. Algunos lenguajes de programación permiten herencia múltiple[1], otros no. Java no lo permite directamente, Python sí.

Hay más tipos de relaciones en la notación UML, pero como el título de este capítulo dice, es sólo una untadita.

14.3. Notación de objetos

Las instancias de una clase se representan como rectángulos con dos partes. La primera tiene el siguiente formato:
`<nombreInstancia>: <nombreClase>`

La otra parte está reservada para los valores de sus atributos. Al igual que en el caso de los diagramas de clases, esta parte se puede obviar cuando no es necesaria.

Podemos ver un ejemplo de un diagrama de objetos en la figura 14.5, en la que hay dos instancias de la clase **Clase**.

[1] es decir, que una clase pueda heredar de varias clases simultaneamente

En los diagramas de objetos también se pueden incluir las referencias entre sí, con o sin nombre y con o sin dirección, según sea el caso, tal como se ve en las figuras 11.3, 11.5 y 11.8.

Parte III

Apéndices

A Plantilla de Aplicación Gráfica en J2SE

A continuación se presenta el esqueleto de una aplicación gráfica en lenguaje java. Es la idea usada para las aplicaciones `transformaciones3D.jar` y `perspectiva3D.jar`.

Se sigue la idea de un diseño Modelo-Vista-Controlador como en la figura A.1.

Listing A.1: Clase controladora

```
1  /* cA/Controlador.java
2   * Objeto controlador
3  */
4  import java.awt.Graphics;
5  import java.awt.event.*;
6
7  /*
8   * Controlador de la aplicación
9   */
10 public class Controlador {
11
12     /**
13      * Función principal independiente de instancias.
14      */
15     public static void main(String args[]) {
16         new Controlador(args);
```

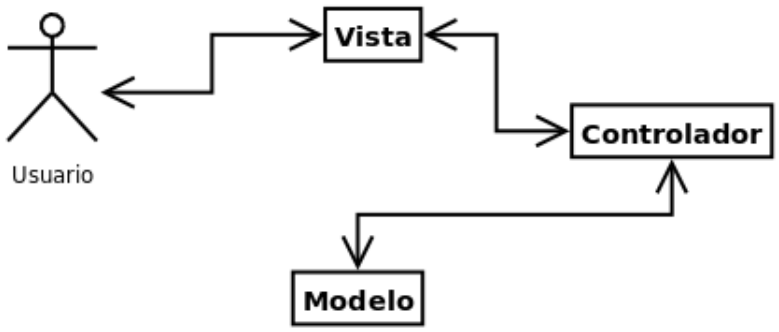


Figura A.1: Modelo de diseño Modelo-Vista-Controlador

```
17      }
18
19      Ventana vista;
20      Modelo modelo;
21
22      /**
23           Hilo principal
24      */
25      public Controlador (String args[]){
26           vista = new Ventana(this);
27           modelo = new Universo3D();
28
29           //mostrar la ventana
30           vista.setVisible(true);
31           System.out.println("Iniciando␣aplicación");
32
33           //acción principal o
34           //conjunto de acciones principales
35           modelo.hacerAlgo();
36
37           //cuando ya acabamos de hacer algo importante,
38           //terminamos la aplicación
39           System.exit(0);
40      }
41
42      /**
43       * Simple delegación del proceso
44       * */
45      public void dibujar(Graphics g){
46           modelo.dibujar(g);
47      }
48
49      /**
50       * Implementar los eventos generados desde la vista.
51       * Como el de cerrar la ventana:
52       * */
53      public WindowAdapter AdaptadorVentana = new WindowAdapter() {
54           public void windowClosing(java.awt.event.WindowEvent evt) {
55                System.out.println();
56                System.exit(0);
57           }
58      };
59  }
60  //fin de la clase Controlador
```

Listing A.2: Clase del modelo

```
1  /* cA/Modelo.java
2   * Objeto principal de la lógica
3   * de la aplicación
```

```
4  */
5  import java.awt.*;
6
7  public class Modelo{
8
9      public Modelo(){
10         /**
11          * Echar a andar todos
12          * los mecanismos del modelo
13          * y toda su lógica.
14          * */
15     }
16
17     public hacerAlgo(){
18         /**
19          * Aquí debería estar el corazón
20          * de la ejecución del modelo.
21          * */
22     }
23
24     /**
25      * El modelo no tiene conciencia
26      * de la procedencia del contexto
27      * gráfico en el que se le
28      * está solicitando trabajar.
29      *
30      * Simplemente responde por delegación.
31      * */
32     public void dibujar(Graphics g){
33         /**
34          * Aquí hay que hacer lo propio.
35          *
36          * Aquí hay que dibujar lo que haya
37          * que dibujar de acuerdo al estado
38          * actual del modelo y de otros
39          * factores relevantes.
40          * */
41     }
42 }
```

Listing A.3: Clase vista

```
1  /* cA/Ventana.java
2   * Objeto vista,
3   * típicamente la Ventana de la aplicación
4  */
5  import java.awt.*;
6
7  public class Ventana extends javax.swing.JFrame implements Config3D{
8
```

```
9       /**
10       * Referencia al controlador para
11       * delegarle la respuesta a los
12       * eventos generados desde aquí.
13       * */
14      private Controlador control;
15
16      /**
17       * Objeto especial para lograr
18       * la delegación de las solicitudes
19       * de refrescamiento
20       * */
21      private PanelEspecial panelprin;
22
23
24      public Ventana(Controlador control) {
25          this.control = control;
26          inicializarComponentes();
27      }
28
29      /** Inicializar los componentes de la interfaz
30       * gráfica de usuario si es que hay.
31       * En el caso de Java con Swing siempre hay.
32       * */
33      private void inicializarComponentes() {
34
35          panelprin = new PanelEspecial(control);
36
37          setDefaultCloseOperation(javax.swing.WindowConstants.
                EXIT_ON_CLOSE);
38          setTitle("Título␣de␣la␣aplicación");
39
40          /**
41           * Agregar todos los "escuchadores",
42           * que deberían estar implementados
43           * en el controlador.
44           *
45           * Todas las respuestas a los eventos
46           * diparados desde este objeto gráfico
47           * y sus incluídos, deberían ser respondidos
48           * por el controlador con asistencia de
49           * los datos del modelo.
50           * */
51          addWindowListener(control.AdaptadorVentana);
52
53          panelprin.setBorder(new javax.swing.border.LineBorder(new java.
                awt.Color(0, 0, 0)));
54          panelprin.setMinimumSize(new java.awt.Dimension(400, 300));
55
56          //agregar el panel principal a la ventana
```

```
57          getContentPane().add(panelprin, java.awt.BorderLayout.CENTER);
58
59          //acomodar dinámicamente los objetos gráficos
60          pack();
61
62          //centrar esta ventana en la pantalla:
63          Dimension tamanjoForzado = new Dimension(400, 300);
64          java.awt.Dimension tamanjoPantalla = java.awt.Toolkit.
                getDefaultToolkit().getScreenSize();
65          setLocation((tamanjoPantalla.width-tamanjoForzado.width)/2,(
                tamanjoPantalla.height-tamanjoForzado.height)/2);
66      }
67  }
68  //fin de la clase vista
```

Listing A.4: Clase especial para delegar el refrescamiento

```
1   /* cA/PanelEspecial.java
2    * Un componente que puede ponerse en un contenedor...
3    * y mostrar las figuras
4   */
5   import java.awt.*;
6   import javax.swing.*;
7   import java.awt.event.*;
8   public class PanelEspecial extends JPanel {
9
10      Controlador control;
11
12      public PanelEspecial(Controlador control){
13          this.control = control;
14      }
15
16      /**
17       * Método llamado cuando es necesario
18       * redibujar la pantalla.
19       * La tarea se delega de la vista al
20       * controlador y de este al modelo.
21       * */
22      public void paintComponent(Graphics g){
23          control.dibujar(g);
24      }
25  }
26  //fin de la clase PanelEspecial
```

Luego de este breve ejemplo, conviene mencionar cómo compilar el código fuente y cómo ensamblarlo en un sólo archivo `.jar` en lugar de ejecutarlo desde los archivos `.class`.

Bueno, la compilación se realiza así:

```
$ javac Controlador.java
```

Se sobreentiende que el archivo de clase indicado es el que debe contener la función `main`.

Esto provocará la compilación en cascada de todos las clases necesarias para que el `main` de `Controlador.java` se ejecute sin problemas.

Para ejecutar el programa se puede hacer:

```
$ java Controlador
```

Pero en lugar de dejar todos los archivos `.class`, podría realizarse la generación de un archivo jar luego de la compilación, así:

```
$ jar -cfe aplicacion.jar Controlador *.class
```

La opción "`c`" indica que se desea crear un archivo `.jar` (el comando sirve para manipular archivos `.jar`, no sólo para crearlos).

La opción "f" indica que el siguiente parámetro debe ser el nombre del archivo a crear (o del archivo a operar).

La opción "e" indica que el siguiente parámetro (en este caso, después del nombre del archivo a operar), es el nombre de la clase con el punto de entrada (donde está el main que queremos que se ejecute primero).

Finalmente listamos todos los archivos que queremos incluir en el archivo creado (en este caso, todos los archivos `.class` generados en el paso de compilación.

Para mayor información sobre el comando `jar`, ver
http://java.sun.com/docs/books/tutorial/deployment/jar/ o ejecutar:
`$ man jar`

B Referencias y manuales

B.1. SDL – Simple DirectMedia Layer

B.1.1. Sitios de recursos

http://www.libsdl.org

http://www.libsdl.es

http://www.javielinux.com

http://www.agserrano.com/publi.html

http://www.losersjuegos.com.ar/

B.1.2. Artículos

¿Por qué SDL? (en español):

http://www.losersjuegos.com.ar/referencia/articulos/why_sdl/why_sdl.php

http://es.wikipedia.org/wiki/Gráficos_3D_por_computadora

Artículo sobre juegos libres

http://www.marevalo.net/creacion/unmundofeliz/1999_12_06_juegos_libres.html

http://es.wikipedia.org/wiki/Desarrollo_de_videojuegos

http://en.wikipedia.org/wiki/Game_programming

http://www.losersjuegos.com.ar/referencia/articulos/articulos.php

Game Programming Wiki

http://wiki.gamedev.net/

B.2. Python y pygame

http://inventwithpython.com/

http://en.wikibooks.org/wiki/Python_Programming

http://openbookproject.net//thinkCSpy/

http://pyspanishdoc.sourceforge.net/

http://www.pygame.org/

http://vpython.wikidot.com/

http://www.vpython.org/

http://www.diveintopython.org/

B.3. Java Me

B.3.1. Sitios de recursos

http://java.sun.com/javame/

http://www.agserrano.com/publi.html

http://programacion.com/java/tutorial/ags_j2me/

Bibliografía

[Foley et al.] Foley, James D.; van Dam, Andries; Feiner, Steven K.; Hughes, John F.; Phillips, Richard L. **Introducción a la graficación por computador**. Addison-Wesley Iberoamericana, 1996.

[Burden y Faires 2002] Burden, Richard L.; Faires, J. Douglas. **Análisis Numérico**. Séptima edición, Thomson Learning, 2002.

[Henríquez 1999] Henríquez, Mauro Hernán. **Cálculo integral en una variable real**. UCA editores, 1999.

[Henríquez 2001] Henríquez, Mauro Hernán. **Cálculo diferencial en una variable real**. UCA editores, 2001.

[RAE] RAE, **Diccionario de la real academia de la lengua española**. Vigésima segunda edición 2001.

[Wikipedia-Bézier Curve] Comunidad de Wikipedia en Inglés. **Wikipedia - Bézier Curve**. Edición del 26 de Febrero de 2010, 13:20 UTC. Revisado el 4 de Marzo de 2010 a las 8:42am, hora local de El Salvador. Enlace permanente: Bézier curve - oldid=346486604.