SDL Guide 中文译版

内容目录

SDL 即 Simple DirectMedia Layer，类似 DirectX，是完整的游戏、多媒体开发包，但不同的是它跨越几乎所有的平台，有各种语言的接口，多种语言的文档，而这一切都是广大志愿者完成的。

类似 DirectX，是完整的游戏、多媒体开发包，但不同的是它跨越几乎所有的平台，有各种语言的接口，多种语言的文档，而这一切都是广大志愿者完成的。目前扩展部分还没有正式的文档，以下为核心部分文档的向导部分。
SDL Guide
SDL 向导中文译版
序言
关于 SDL
SDL 为方便制作能跨跃 Linux、BSD、MacOS、Win32 和 BeOS 平台，使用本地高性能媒体接口，并且让您可以只需使用一份源代码级 API 而设计。SDL 是相当低层的 API，但使用它可以让你以极大的灵活性写出完全跨平台的程序。

关于 SDL 文档
SDLdoc 项目即要重新编写 SDL 文档并同步更新。项目组有使用 SDL 的志愿者组成。

最新版本可在 http://sdldoc.sourceforge.net 下载

第一章 基础

初始化

SDL 由八个子系统组成——音频、CDROM、事件处理、文件 I/O、游戏杆、线程、记时器和视频。使用前必须调用 SDL_Init 或 SDL_InitSubSystem 初始化。SDL_Init 必须早于其他所有 SDL 调用，它将自动初始化事件处理、文件 I/O 和线程子系统，并根据参数选择启动其他子系统。例如，初始化缺省和视频子系统：

SDL_Init(SDL_INIT_VIDEO);
初始化缺省、视频和记时器子系统：

SDL_Init(SDL_INIT_VIDEO | SDL_INIT_TIMER);
SDL_Init 对应 SDL_Quit（和 SDL_QuitSubSystem）。SDL_Quit 关闭所有子系统，必须在程序关闭前调用。

除此之外，我们还必须进行错误处理。很多 SDL 函数返回一个值指示成功与否。例如 SDL_Init 失败时返回-1。每当 SDL 出错时，错误信息被保存，并可用 SDL_GetError 取得。

例 1-1 初始化 SDL

```
#include "SDL.h"   /* All SDL App's need this */
```

```
#include <stdio.h>int main()
{
	printf("Initializing SDL.");		/* Initialize defaults, Video and Audio */
	if((SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO)==-1))
	{
		printf("Could not initialize SDL: %s.", SDL_GetError());
		exit(-1);
	}
	printf("SDL initialized.");
	printf("Quiting SDL.");		/* Shutdown all subsystems */
	SDL_Quit();
	printf("Quiting....");
	exit(0);
}
```

第二章　图像和视频

初始化 SDL Video 显示

视频是最常用的部分，也是 SDL 最完整的子系统。下面的初始化过程是每个 SDL 程序都要做的，即使可能有些不同。

例 2-1 初始化视频

```
SDL_Surface  *screen;

/*  Initialize  the  SDL  library  */
if(  SDL_Init(SDL_INIT_VIDEO)  <  0  )
{
      fprintf(stderr,
                  "Couldn""t  initialize  SDL:  %s\n",  SDL_GetError());
      exit(1);
}

/*  Clean  up  on  exit  */
atexit(SDL_Quit);

/*
  *  Initialize  the  display  in  a  640x480  8-bit  palettized  mode,
  *  requesting  a  software  surface
  */
screen  =  SDL_SetVideoMode(640,  480,  8,  SDL_SWSURFACE);
if  (  screen  ==  NULL  )  {
      fprintf(stderr, "Couldn""t set 640x480x8 video mode: %s\n",
                      SDL_GetError());
      exit(1);
}
```

初始化最佳视频模式

如果你希望某种色深（颜色数）但如果用户的显示器不支持也可以接受其他色深，使用加 SDL_ANYFORMAT 参数的 SDL_SetVideoMode。您还可以用 SDL_VideoModeOK 来找到与请求模式最接近的模式。

例 2-2 初始化最佳视频模式

```
    /* Have a preference for 8-bit, but accept any depth */
    screen = SDL_SetVideoMode(640, 480, 8, SDL_SWSURFACE|SDL_ANYFORMAT);
    if ( screen == NULL ) {
        fprintf(stderr, "Couldn""t set 640x480x8 video mode: %s\n",
                        SDL_GetError());
        exit(1);
    }
    printf("Set 640x480 at %d bits-per-pixel mode\n",
          screen->format->BitsPerPixel);
```

读取并显示 BMP 文件

当 SDL 已经初始化，视频模式已经选择，下面的函数将读取并显示指定的 BMP 文件。

例 2-3 读取并显示 BMP 文件

```
void display_bmp(char *file_name)
{
    SDL_Surface *image;

    /* Load the BMP file into a surface */
    image = SDL_LoadBMP(file_name);
    if (image == NULL) {
        fprintf(stderr, "Couldn""t load %s: %s\n", file_name, SDL_GetError());
        return;
    }

    /*
     * Palettized screen modes will have a default palette (a standard
     * 8*8*4 colour cube), but if the image is palettized as well we can
     * use that palette for a nicer colour matching
     */
    if (image->format->palette && screen->format->palette) {
    SDL_SetColors(screen, image->format->palette->colors, 0,
                  image->format->palette->ncolors);
    }

    /* Blit onto the screen surface */
    if(SDL_BlitSurface(image, NULL, screen, NULL) < 0)
        fprintf(stderr, "BlitSurface error: %s\n", SDL_GetError());

    SDL_UpdateRect(screen, 0, 0, image->w, image->h);

    /* Free the allocated BMP surface */
    SDL_FreeSurface(image);
}
```

直接在显示上绘图

下面两个函数实现在图像平面的像素读写。它们被细心设计成可以用于所有色深。记住在使用前要先锁定图像平面，之后要解锁。

在像素值和其红、绿、蓝值间转换，使用 SDL_GetRGB()和 SDL_MapRGB()。

例 2-4 getpixel()

```
/*
 * Return the pixel value at (x, y)
 * NOTE: The surface must be locked before calling this!
 */
Uint32 getpixel(SDL_Surface *surface, int x, int y)
{
    int bpp = surface->format->BytesPerPixel;
    /* Here p is the address to the pixel we want to retrieve */
    Uint8 *p = (Uint8 *)surface->pixels  y * surface->pitch  x * bpp;

    switch(bpp) {
    case 1:
        return *p;

    case 2:
        return *(Uint16 *)p;

    case 3:
        if(SDL_BYTEORDER == SDL_BIG_ENDIAN)
            return p[0] << 16 | p[1] << 8 | p[2];
            else
                  return  p[0]  |  p[1]  <<  8  |  p[2]  <<  16;

      case  4:
            return  *(Uint32  *)p;

      default:
            return  0;       /*  shouldn""t  happen,  but  avoids  warnings  */
      }
}
```

例 2-5 putpixel()

```
/*
  *  Set  the  pixel  at  (x,  y)  to  the  given  value
  *  NOTE:  The  surface  must  be  locked  before  calling  this!
  */
void  putpixel(SDL_Surface  *surface,  int  x,  int  y,  Uint32  pixel)
{
      int  bpp  =  surface->format->BytesPerPixel;
      /*  Here  p  is  the  address  to  the  pixel  we  want  to  set  */
      Uint8  *p  =  (Uint8  *)surface->pixels    y  *  surface->pitch    x  *  bpp;

      switch(bpp)  {
      case  1:
            *p  =  pixel;
            break;

      case  2:
            *(Uint16  *)p  =  pixel;
            break;

      case  3:
            if(SDL_BYTEORDER  ==  SDL_BIG_ENDIAN)  {
                  p[0]  =  (pixel  >>  16)  &  0xff;
                  p[1]  =  (pixel  >>  8)  &  0xff;
                  p[2]  =  pixel  &  0xff;
            }  else  {
                  p[0]  =  pixel  &  0xff;
                  p[1]  =  (pixel  >>  8)  &  0xff;
                  p[2]  =  (pixel  >>  16)  &  0xff;
            }
```

```
        break;

    case 4:
        *(Uint32 *)p = pixel;
        break;
    }
}
```

例 2-6 使用上面的 putpixel()在屏幕中心画一个黄点

```
    /* Code to set a yellow pixel at the center of the screen */

    int x, y;
    Uint32 yellow;

    /* Map the color yellow to this display (R=0xff, G=0xFF, B=0x00)
       Note: If the display is palettized, you must set the palette first.
    */
    yellow = SDL_MapRGB(screen->format, 0xff, 0xff, 0x00);

    x = screen->w / 2;
    y = screen->h / 2;

    /* Lock the screen for direct access to the pixels */
    if ( SDL_MUSTLOCK(screen) ) {
        if ( SDL_LockSurface(screen) < 0 ) {
            fprintf(stderr, "Can""t lock screen: %s\n", SDL_GetError());
            return;
        }
    }

    putpixel(screen, x, y, yellow);

    if ( SDL_MUSTLOCK(screen) ) {
        SDL_UnlockSurface(screen);
    }
    /* Update just the part of the display that we""ve changed */
    SDL_UpdateRect(screen, x, y, 1, 1);

    return;
```

并用 SDL 和 OpenGL

SDL 可以在多种平台（Linux/X11, Win32, BeOS, MacOS Classic/Toolbox, MacOS X, FreeBSD/X11 and Solaris/X11）上创建和使用 OpenGL 上下文。这允许你在 OpenGL 程序中使用 SDL 的音频、事件、线程和记时器，而这些通常是 GLUT 的任务。

和普通的初始化类似，但有三点不同：必须传 SDL_OPENGL 参数给 SDL_SetVideoMode；必须使用 SDL_GL_SetAttribute 指定一些 GL 属性（深度缓冲区位宽，帧缓冲位宽等）；如果您想使用双缓冲，必须作为 GL 属性指定

例 2-7 初始化 SDL 加 OpenGL

```
/* Information about the current video settings. */
const SDL_VideoInfo* info = NULL;
/* Dimensions of our window. */
int width = 0;
int height = 0;
/* Color depth in bits of our window. */
int bpp = 0;
/* Flags we will pass into SDL_SetVideoMode. */
int flags = 0;

/* First, initialize SDL""s video subsystem. */
if( SDL_Init( SDL_INIT_VIDEO ) < 0 ) {
    /* Failed, exit. */
    fprintf( stderr, "Video initialization failed: %s\n",
         SDL_GetError( ) );
    quit_tutorial( 1 );
}

/* Let""s get some video information. */
info = SDL_GetVideoInfo( );

if( !info ) {
    /* This should probably never happen. */
    fprintf( stderr, "Video query failed: %s\n",
         SDL_GetError( ) );
    quit_tutorial( 1 );
}

/*
 * Set our width/height to 640/480 (you would
 * of course let the user decide this in a normal
 * app). We get the bpp we will request from
 * the display. On X11, VidMode can""t change
 * resolution, so this is probably being overly
 * safe. Under Win32, ChangeDisplaySettings
 * can change the bpp.
 */
width = 640;
height = 480;
bpp = info->vfmt->BitsPerPixel;

/*
 * Now, we want to setup our requested
 * window attributes for our OpenGL window.
 * We want *at least* 5 bits of red, green
 * and blue. We also want at least a 16-bit
 * depth buffer.
 *
 * The last thing we do is request a double
 * buffered window. ""1"" turns on double
 * buffering, ""0"" turns it off.
 *
 * Note that we do not use SDL_DOUBLEBUF in
 * the flags to SDL_SetVideoMode. That does
 * not affect the GL attribute state, only
 * the standard 2D blitting setup.
 */
SDL_GL_SetAttribute( SDL_GL_RED_SIZE, 5 );
SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 5 );
SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 5 );
SDL_GL_SetAttribute( SDL_GL_DEPTH_SIZE, 16 );
SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );

/*
 * We want to request that SDL provide us
 * with an OpenGL window, in a fullscreen
 * video mode.
 *
 * EXERCISE:
 * Make starting windowed an option, and
 * handle the resize events properly with
 * glViewport.
```

```
     */
    flags = SDL_OPENGL | SDL_FULLSCREEN;

    /*
     * Set the video mode
     */
    if( SDL_SetVideoMode( width, height, bpp, flags ) == 0 ) {
        /*
         * This could happen for a variety of reasons,
         * including DISPLAY not being set, the specified
         * resolution not being available, etc.
         */
        fprintf( stderr, "Video mode set failed: %s\n",
             SDL_GetError( ) );
        quit_tutorial( 1 );
    }
```

OpenGL 绘图

除了初始化，在 SDL 程序中使用 OpenGL 和其他情况基本相同，是同样函数和数据类型。但是如果您使用双缓冲，则必须用 SDL_GL_SwapBuffers()来交换前后缓冲，而不是 glxSwapBuffers()（GLX）或 SwapBuffers()（Windows）。

例 2-8 SDL 和 OpenGL

```
/*
  * SDL OpenGL Tutorial.
  * (c) Michael Vance, 2000
  * briareos@lokigames.com
  *
  * Distributed under terms of the LGPL.
  */

#include <SDL/SDL.h>
#include <GL/gl.h>
#include <GL/glu.h>

#include <stdio.h>
#include <stdlib.h>

static GLboolean should_rotate = GL_TRUE;

static void quit_tutorial( int code )
{
        /*
         * Quit SDL so we can release the fullscreen
         * mode and restore the previous video settings,
         * etc.
         */
        SDL_Quit( );

        /* Exit program. */
        exit( code );
}

static void handle_key_down( SDL_keysym* keysym )
{

        /*
         * We""re only interested if ""Esc"" has
         * been presssed.
         *
         * EXERCISE:
         * Handle the arrow keys and have that change the
         * viewing position/angle.
         */
        switch( keysym->sym ) {
        case SDLK_ESCAPE:
            quit_tutorial( 0 );
            break;
        case SDLK_SPACE:
            should_rotate = !should_rotate;
            break;
        default:
            break;
        }

}

static void process_events( void )
{
        /* Our SDL event placeholder. */
        SDL_Event event;

        /* Grab all the events off the queue. */
        while( SDL_PollEvent( &event ) ) {
```

```
        switch( event.type ) {
        case SDL_KEYDOWN:
            /* Handle key presses. */
            handle_key_down( &event.key.keysym );
            break;
        case SDL_QUIT:
            /* Handle quit requests (like Ctrl-c). */
            quit_tutorial( 0 );
            break;
        }

    }

}

static void draw_screen( void )
{
    /* Our angle of rotation. */
    static float angle = 0.0f;

    /*
     * EXERCISE:
     * Replace this awful mess with vertex
     * arrays and a call to glDrawElements.
     *
     * EXERCISE:
     * After completing the above, change
     * it to use compiled vertex arrays.
     *
     * EXERCISE:
     * Verify my windings are correct here ;).
     */
    static GLfloat v0[] = { -1.0f, -1.0f,  1.0f };
    static GLfloat v1[] = {  1.0f, -1.0f,  1.0f };
    static GLfloat v2[] = {  1.0f,  1.0f,  1.0f };
    static GLfloat v3[] = { -1.0f,  1.0f,  1.0f };
    static GLfloat v4[] = { -1.0f, -1.0f, -1.0f };
    static GLfloat v5[] = {  1.0f, -1.0f, -1.0f };
    static GLfloat v6[] = {  1.0f,  1.0f, -1.0f };
    static GLfloat v7[] = { -1.0f,  1.0f, -1.0f };
    static GLubyte red[]    = { 255,   0,   0, 255 };
    static GLubyte green[]  = {   0, 255,   0, 255 };
    static GLubyte blue[]   = {   0,   0, 255, 255 };
    static GLubyte white[]  = { 255, 255, 255, 255 };
    static GLubyte yellow[] = {   0, 255, 255, 255 };
    static GLubyte black[]  = {   0,   0,   0, 255 };
    static GLubyte orange[] = { 255, 255,   0, 255 };
    static GLubyte purple[] = { 255,   0, 255,   0 };

    /* Clear the color and depth buffers. */
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    /* We don""t want to modify the projection matrix. */
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity( );

    /* Move down the z-axis. */
    glTranslatef( 0.0, 0.0, -5.0 );

    /* Rotate. */
    glRotatef( angle, 0.0, 1.0, 0.0 );

    if( should_rotate ) {
        if(  angle > 360.0f ) {
            angle = 0.0f;
        }
    }

    /* Send our triangle data to the pipeline. */
    glBegin( GL_TRIANGLES );

    glColor4ubv( red );
    glVertex3fv( v0 );
    glColor4ubv( green );
    glVertex3fv( v1 );
    glColor4ubv( blue );
    glVertex3fv( v2 );

    glColor4ubv( red );
    glVertex3fv( v0 );
    glColor4ubv( blue );
```

```
glVertex3fv(  v2  );
glColor4ubv(  white  );
glVertex3fv(  v3  );

glColor4ubv(  green  );
glVertex3fv(  v1  );
glColor4ubv(  black  );
glVertex3fv(  v5  );
glColor4ubv(  orange  );
glVertex3fv(  v6  );

glColor4ubv(  green  );
glVertex3fv(  v1  );
glColor4ubv(  orange  );
glVertex3fv(  v6  );
glColor4ubv(  blue  );
glVertex3fv(  v2  );

glColor4ubv(  black  );
glVertex3fv(  v5  );
glColor4ubv(  yellow  );
glVertex3fv(  v4  );
glColor4ubv(  purple  );
glVertex3fv(  v7  );

glColor4ubv(  black  );
glVertex3fv(  v5  );
glColor4ubv(  purple  );
glVertex3fv(  v7  );
glColor4ubv(  orange  );
glVertex3fv(  v6  );

glColor4ubv(  yellow  );
glVertex3fv(  v4  );
glColor4ubv(  red  );
glVertex3fv(  v0  );
glColor4ubv(  white  );
glVertex3fv(  v3  );

glColor4ubv(  yellow  );
glVertex3fv(  v4  );
```

```
glColor4ubv( white );
glVertex3fv( v3 );
glColor4ubv( purple );
glVertex3fv( v7 );

glColor4ubv( white );
glVertex3fv( v3 );
glColor4ubv( blue );
glVertex3fv( v2 );
glColor4ubv( orange );
glVertex3fv( v6 );

glColor4ubv( white );
glVertex3fv( v3 );
glColor4ubv( orange );
glVertex3fv( v6 );
glColor4ubv( purple );
glVertex3fv( v7 );

glColor4ubv( green );
glVertex3fv( v1 );
glColor4ubv( red );
glVertex3fv( v0 );
glColor4ubv( yellow );
glVertex3fv( v4 );

glColor4ubv( green );
glVertex3fv( v1 );
glColor4ubv( yellow );
glVertex3fv( v4 );
glColor4ubv( black );
glVertex3fv( v5 );

glEnd( );

/*
 * EXERCISE:
 * Draw text telling the user that ""Spc""
 * pauses the rotation and ""Esc"" quits.
 * Do it using vetors and textured quads.
 */
```

```
        /*
         * Swap the buffers. This this tells the driver to
         * render the next frame from the contents of the
         * back-buffer, and to set all rendering operations
         * to occur on what was the front-buffer.
         *
         * Double buffering prevents nasty visual tearing
         * from the application drawing on areas of the
         * screen that are being updated at the same time.
         */
        SDL_GL_SwapBuffers( );
}

static void setup_opengl( int width, int height )
{
        float ratio = (float) width / (float) height;

        /* Our shading model--Gouraud (smooth). */
        glShadeModel( GL_SMOOTH );

        /* Culling. */
        glCullFace( GL_BACK );
        glFrontFace( GL_CCW );
        glEnable( GL_CULL_FACE );

        /* Set the clear color. */
        glClearColor( 0, 0, 0, 0 );

        /* Setup our viewport. */
        glViewport( 0, 0, width, height );

        /*
         * Change to the projection matrix and set
         * our viewing volume.
         */
        glMatrixMode( GL_PROJECTION );
        glLoadIdentity( );
        /*
         * EXERCISE:
         * Replace this with a call to glFrustum.
         */
        gluPerspective( 60.0, ratio, 1.0, 1024.0 );
}

int main( int argc, char* argv[] )
{
        /* Information about the current video settings. */
        const SDL_VideoInfo* info = NULL;
        /* Dimensions of our window. */
        int width = 0;
        int height = 0;
        /* Color depth in bits of our window. */
        int bpp = 0;
        /* Flags we will pass into SDL_SetVideoMode. */
        int flags = 0;

        /* First, initialize SDL""s video subsystem. */
        if( SDL_Init( SDL_INIT_VIDEO ) < 0 ) {
            /* Failed, exit. */
            fprintf( stderr, "Video initialization failed: %s\n",
                 SDL_GetError( ) );
            quit_tutorial( 1 );
        }

        /* Let""s get some video information. */
        info = SDL_GetVideoInfo( );

        if( !info ) {
            /* This should probably never happen. */
            fprintf( stderr, "Video query failed: %s\n",
                 SDL_GetError( ) );
            quit_tutorial( 1 );
        }

        /*
         * Set our width/height to 640/480 (you would
         * of course let the user decide this in a normal
         * app). We get the bpp we will request from
         * the display. On X11, VidMode can""t change
         * resolution, so this is probably being overly
         * safe. Under Win32, ChangeDisplaySettings
```

```
 * can change the bpp.
 */
width = 640;
height = 480;
bpp = info->vfmt->BitsPerPixel;

/*
 * Now, we want to setup our requested
 * window attributes for our OpenGL window.
 * We want *at least* 5 bits of red, green
 * and blue. We also want at least a 16-bit
 * depth buffer.
 *
 * The last thing we do is request a double
 * buffered window. ""1"" turns on double
 * buffering, ""0"" turns it off.
 *
 * Note that we do not use SDL_DOUBLEBUF in
 * the flags to SDL_SetVideoMode. That does
 * not affect the GL attribute state, only
 * the standard 2D blitting setup.
 */
SDL_GL_SetAttribute( SDL_GL_RED_SIZE, 5 );
SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 5 );
SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 5 );
SDL_GL_SetAttribute( SDL_GL_DEPTH_SIZE, 16 );
SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );

/*
 * We want to request that SDL provide us
 * with an OpenGL window, in a fullscreen
 * video mode.
 *
 * EXERCISE:
 * Make starting windowed an option, and
 * handle the resize events properly with
 * glViewport.
 */
flags = SDL_OPENGL | SDL_FULLSCREEN;

/*
```

```
         * Set the video mode
         */
        if( SDL_SetVideoMode( width, height, bpp, flags ) == 0 ) {
            /*
             * This could happen for a variety of reasons,
             * including DISPLAY not being set, the specified
             * resolution not being available, etc.
             */
            fprintf( stderr, "Video mode set failed: %s\n",
                 SDL_GetError( ) );
            quit_tutorial( 1 );
        }

        /*
         * At this point, we should have a properly setup
         * double-buffered window for use with OpenGL.
         */
        setup_opengl( width, height );

        /*
         * Now we want to begin our normal app process--
         * an event loop with a lot of redrawing.
         */
        while( 1 ) {
            /* Process incoming events. */
            process_events( );
            /* Draw the screen. */
            draw_screen( );
        }

        /*
         * EXERCISE:
         * Record timings using SDL_GetTicks() and
         * and print out frames per second at program
         * end.
         */

        /* Never reached. */
        return 0;
}
```

第三章上 游戏杆输入处理

初始化

使用游戏杆的第一步是初始化游戏杆子系统。即在 SDL_Init 时使用参数 SDL_INIT_JOYSTICK。

例 3-1 初始化 SDL 并带游戏杆支持

```
if ( ! SDL_Init( SDL_INIT_VIDEO | SDL_INIT_JOYSTICK ) )
{
      fprintf(stderr, "Couldn""t initialize SDL: %s\n", SDL_GetError());
      exit(1);
}
```

此例启动 SDL 并带视频和游戏杆支持。

查询

至此，我们可以假定初始化以完成。但我们还需指导有没有、有几个游戏杆。即使您指导有一个游戏杆可用，也最好经常检查，因为这样可以帮助检测游戏杆被拔掉的情况。

检查游戏杆的函数是 SDL_NumJoysticks()。此函数简单的返回系统中游戏杆的数量。下一步是决定用户想用哪一个。当然，如果只有一个就不用决定了。SDL_JoystickName 取得系统赋给游戏杆的名字。游戏杆用序号指定，第一个为 0，最后一个为 SDL_NumJoysticks - 1。

例 3-2 打印所有游戏杆的名字

```
printf("%i joysticks were found.\n\n", SDL_NumJoysticks() );
printf("The names of the joysticks are:\n");

for( i=0; i < SDL_NumJoysticks(); i   )

      printf("    %s\n", SDL_JoystickName(i
}
```

启动游戏杆取得事件

SDL 使用事件架构，游戏杆可以触发四种事件。

SDL_JoyAxisEvent 轴改变

SDL_JoyBallEvent 轨迹球坐标改变

SDL_JoyHatEvent hat（sorry，在下不解何为 hat）方向改变

SDL_JoyButtonEvent 按钮按下或释放

所有启动的游戏杆都会触发事件。为了收到事件，首先要用 SDL_ENABLE 调用 SDL_JoystickEventState，以允许事件。其次要启动指定的游戏杆，用 SDL_JoystickOpen()。

例 3-3 启动第一个游戏杆

```
SDL_Joystick  *joystick;

SDL_JoystickEventState(SDL_ENABLE);
joystick  =  SDL_JoystickOpen(0);
```

游戏杆对象的指针只在查询和关闭游戏杆时有用。

现在为了响应事件，我们需要一个消息循环。所有的 SDL 程序至少要接受系统退出消息。设想我们的消息循环象这样：

```
SDL_Event  event;
/*  Other  initializtion  code  goes  here  */

/*  Start  main  game  loop  here  */

while(SDL_PollEvent(&event))
{
      switch(event.type)
      {
            case  SDL_KEYDOWN:
            /*  handle  keyboard  stuff  here  */
              break

              case  SDL_QUIT
            /*  Set  whatever  flags  are  necessary  to  *
            /*  end  the  main  game  loop  here  *
            break
  /*  End  loop  here  */
```

要响应游戏杆事件，只需添加一个 case。轴检测得有些技巧，因为大部分事件是垃圾。摇杆动一点就会有事件。所以必须设定一个阈值，并且忽略未达到阈值的事件。一般 10%是个比较好的阈值。

例 3-4 游戏杆轴事件

```
case  SDL_JOYAXISMOTION:  /*  Handle  Joystick  Motion  */
if  (  (  event.jaxis.value  <  -3200  )  ||  (event.jaxis.value  >  3200  )  )
{
    /* code goes here */
}
break;
```

另一个技巧是上下和左右是两组不同的运动。最重要的轴是轴 0（左右）和轴 1（上下）。按如下方法作不同处理：

例 3-5

```
case SDL_JOYAXISMOTION:  /* Handle Joystick Motion */
if ( ( event.jaxis.value < -3200 ) || (event.jaxis.value > 3200 ) )
{
    if( event.jaxis.axis == 0)
    {
        /* Left-right movement code goes here */
    }

    if( event.jaxis.axis == 1)
    {
        /* Up-Down movement code goes here */
    }
}
break;
```

理想情况下，应该用 event.jaxis.value 来调整一些值。例如你在用游戏杆控制飞船，将摇杆推一点则慢速前进，推很多则快速前进。这样设计会使用户的体验更好。如果你的游戏杆有更多的轴，可用于其他控制，用法完全一样，event.jaxis.axis 会有不同的值。

例 3-6 游戏杆按钮事件

```
case SDL_JOYBUTTONDOWN:  /* Handle Joystick Button Presses */
if ( event.jbutton.button == 0 )
{
    /* code goes here */
}
break;
```

按钮检测很简单，因为只有按下和放开两个状态。按下时 SDL_JOYBUTTONDOWN 触发，放开时 SDL_JOYBUTTONUP 触发。event.jbutton.button 指示是哪个按钮。

最后，程序结束时用 SDL_JoystickClose()关闭游戏杆。例如关闭前面启动的 0 号游戏杆：
SDL_JoystickClose(joystick);
高级游戏杆函数

轨迹球消息包含 X 和 Y 方向的改变量。

例 3-7 轨迹球事件

```
case  SDL_JOYBALLMOTION:   /*  Handle  Joyball  Motion  */
if(  event.jball.ball  ==  0  )
{
   /*  ball  handling  */
}
break;
```

此例检测第一个轨迹球。坐标改变量在 event.jball.xrel 和 event.jball.yrel 中。

最后是 hat 事件。hat 只报告方向。我们通过位掩码检测:

SDL_HAT_CENTERED
SDL_HAT_UP
SDL_HAT_RIGHT
SDL_HAT_DOWN
SDL_HAT_LEFT

预定义的组合:
SDL_HAT_RIGHTUP
SDL_HAT_RIGHTDOWN
SDL_HAT_LEFTUP
SDL_HAT_LEFTDOWN

例 3-8 游戏杆 hat 事件

```
case  SDL_JOYHATMOTION:   /*  Handle  Hat  Motion  */
if  (  event.jhat.hat  |  SDL_HAT_UP  )
{
      /*  Do  up  stuff  here  */
}

if  (  event.jhat.hat  |  SDL_HAT_LEFT  )
{
      /*  Do  left  stuff  here  */
}

if  (  event.jhat.hat  |  SDL_HAT_RIGHTDOWN  )
{
      /*  Do  right  and  down  together  stuff  here  */
}
```

```
        break;
```

除了游戏杆的数量，还可查询：
SDL_JoystickNumAxes 轴数量
SDL_JoystickNumButtons 按钮数量
SDL_JoystickNumBalls 轨迹球数量
SDL_JoystickNumHats hat 数量

只需将启动游戏杆时得到的指针传给这些函数即可。

例 3-9 查询游戏杆特性

```
        int  number_of_buttons;
        SDL_Joystick  *joystick;

        joystick  =  SDL_JoystickOpen(0);
        number_of_buttons  =  SDL_JoystickNumButtons(joystick);
```

第三章下 键盘输入

键盘相关数据结构
SDLKey 枚举类型，每一个符号代表一个键，如 SDLK_a、SDLK_SPACE，在 SDL_keysym.h 中定义。
SDLMod 枚举类型，类似 SDLKey，但用于修饰键，如 Control、Alt、Shift，可以组合。
SDL_keysym

```
typedef  struct{
    Uint8  scancode;
    SDLKey  sym;
    SDLMod  mod;
    Uint16  unicode;
}  SDL_keysym;
```

用于按键和放开消息。scancode 是硬件相关键码。除非你有特殊用途，否则忽略。sym 指示键，mod 为修饰键集合，例如 KMOD_NUM | KMOD_CAPS | KMOD_LSHIFT 为 Numlock、Capslock 和左 Shift 组合。最后 unicode 为对应字符。注意，仅当是按键消息时 unicode 才有效，放开消息时无效。Unicode 变换必须用 SDL_EnableUNICODE()打开。
SDL_KeyboardEvent

```
typedef  struct{
    Uint8  type;
    Uint8  state;
    SDL_keysym  keysym;
}  SDL_KeyboardEvent;
```

描述一个键盘消息。type 指示按键（SDL_KEYDOWN）或放开（SDL_KEYUP）。state 是冗余的，和含义 type 相同只是符号不同（SDL_PRESSED，SDL_RELEASED）。

读取键盘消息

使用消息循环用 SDL_PollEvent()从消息队列里读取，并用 switch-case 检测 SDL_KEYUP 和 SDL_KEYDOWN。下面是个基本的例子。

例 3-10 读键盘消息

```
SDL_Event  event;
.
.
/*  Poll  for  events.  SDL_PollEvent()  returns  0  when  there  are  no   */
/*  more  events  on  the  event  queue,  our  while  loop  will  exit  when  */
/*  that  occurs.                                              */
while(  SDL_PollEvent(  &event  )  ){
   /*  We  are  only  worried  about  SDL_KEYDOWN  and  SDL_KEYUP  events  */
   switch(  event.type  ){
      case  SDL_KEYDOWN:
         printf(  "Key  press  detected\n"  );
         break;

      case  SDL_KEYUP:
         printf(  "Key  release  detected\n"  );
         break;

      default:
         break;
   }
}
```

更详细的内容

我们已经知道要用 SDL_Init 和 SDL_SetVideoMode 初始化，但我们还得用另外两个函数取得必要信息。要调用 SDL_EnableUNICODE(1)以允许 unicode 变换，还要用 SDL_GetKeyName 将 SDLKey 转换成可打印字符。注意：小于 0x80 的 unicode 字符直接映射到 ASCII 码。下例用到这一点。

例 3-11 解释按键消息

```
#include  "SDL.h"

/*  Function  Prototypes  */
void  PrintKeyInfo(  SDL_KeyboardEvent  *key  );
void  PrintModifiers(  SDLMod  mod  );
```

```
/* main */
int main( int argc, char *argv[] ){

     SDL_Event event;
     int quit = 0;

     /* Initialise SDL */
     if( SDL_Init( SDL_INIT_VIDEO ) ){
          fprintf( stderr, "Could not initialise SDL: %s\n", SDL_GetError() );
          exit( -1 );
     }

     /* Set a video mode */
     if( !SDL_SetVideoMode( 320, 200, 0, 0 ) ){
          fprintf( stderr, "Could not set video mode: %s\n", SDL_GetError() );
          SDL_Quit();
          exit( -1 );
     }

     /* Enable Unicode translation */
     SDL_EnableUNICODE( 1 );

     /* Loop until an SDL_QUIT event is found */
     while( !quit ){

          /* Poll for events */
          while( SDL_PollEvent( &event ) ){

               switch( event.type ){
                    /* Keyboard event */
                    /* Pass the event data onto PrintKeyInfo() */
                    case SDL_KEYDOWN:
                    case SDL_KEYUP:
                         PrintKeyInfo( &event.key );
                         break;

                    /* SDL_QUIT event (window close) */
                    case SDL_QUIT:
                         quit = 1;
                         break;
```

```
                default:
                    break;
            }

        }

    }

    /* Clean up */
    SDL_Quit();
    exit( 0 );
}

/* Print all information about a key event */
void PrintKeyInfo( SDL_KeyboardEvent *key ){
    /* Is it a release or a press? */
    if( key->type == SDL_KEYUP )
        printf( "Release:- " );
    else
        printf( "Press:- " );

    /* Print the hardware scancode first */
    printf( "Scancode: 0x%02X", key->keysym.scancode );
    /* Print the name of the key */
    printf( ", Name: %s", SDL_GetKeyName( key->keysym.sym ) );
    /* We want to print the unicode info, but we need to make */
    /* sure its a press event first (remember, release events */
    /* don't have unicode info                                */
    if( key->type == SDL_KEYDOWN ){
        /* If the Unicode value is less than 0x80 then the    */
        /* unicode value can be used to get a printable       */
        /* representation of the key, using (char)unicode.    */
        printf(", Unicode: " );
        if( key->keysym.unicode < 0x80 && key->keysym.unicode > 0 ){
            printf( "%c (0x%04X)", (char)key->keysym.unicode,
                    key->keysym.unicode );
        }
        else{
            printf( "? (0x%04X)", key->keysym.unicode );
        }
    }
```

```
        printf( "\n" );
        /* Print modifier info */
        PrintModifiers( key->keysym.mod );
    }

    /* Print modifier info */
    void PrintModifiers( SDLMod mod ){
        printf( "Modifers: " );

        /* If there are none then say so and return */
        if( mod == KMOD_NONE ){
            printf( "None\n" );
            return;
        }

        /* Check for the presence of each SDLMod value */
        /* This looks messy, but there really isn't    */
        /* a clearer way.                              */
        if( mod & KMOD_NUM ) printf( "NUMLOCK " );
        if( mod & KMOD_CAPS ) printf( "CAPSLOCK " );
        if( mod & KMOD_LCTRL ) printf( "LCTRL " );
        if( mod & KMOD_RCTRL ) printf( "RCTRL " );
        if( mod & KMOD_RSHIFT ) printf( "RSHIFT " );
        if( mod & KMOD_LSHIFT ) printf( "LSHIFT " );
        if( mod & KMOD_RALT ) printf( "RALT " );
        if( mod & KMOD_LALT ) printf( "LALT " );
        if( mod & KMOD_CTRL ) printf( "CTRL " );
        if( mod & KMOD_SHIFT ) printf( "SHIFT " );
        if( mod & KMOD_ALT ) printf( "ALT " );
        printf( "\n" );
    }
```

游戏式键盘输入

键盘消息仅在键的状态在按下和放开间变化时才触发。

设想你用光标键控制飞船运动以改变眼前看到的太空景象，当你按左键并希望镜头向左转时。看看下面的代码，并注意它为什么是错的。

```
    /* Alien screen coordinates */
    int alien_x=0, alien_y=0;
    .
    .
```

```
/* Initialise SDL and video modes and all that */
.
/* Main game loop */
/* Check for events */
while( SDL_PollEvent( &event ) ){
    switch( event.type ){
        /* Look for a keypress */
        case SDL_KEYDOWN:
            /* Check the SDLKey values and move change the coords */
            switch( event.key.keysym.sym ){
                case SDLK_LEFT:
                    alien_x -= 1;
                    break;
                case SDLK_RIGHT:
                    alien_x += 1;
                    break;
                case SDLK_UP:
                    alien_y -= 1;
                    break;
                case SDLK_DOWN:
                    alien_y += 1;
                    break;
                default:
                    break;
            }
        }
    }
}
```

问题在于你必须按 100 次左以便得到 100 次键盘消息，才能向左转 100 像素。正确的方法是收到事件时设定标志，根据标志来移动。

例 3-12 正确的运动控制

```
/* Alien screen coordinates */
int alien_x=0, alien_y=0;
int alien_xvel=0, alien_yvel=0;
.
.
/* Initialise SDL and video modes and all that */
.
```

```
/* Main game loop */
/* Check for events */
while( SDL_PollEvent( &event ) ){
    switch( event.type ){
        /* Look for a keypress */
        case SDL_KEYDOWN:
            /* Check the SDLKey values and move change the coords */
            switch( event.key.keysym.sym ){
                case SDLK_LEFT:
                    alien_xvel = -1;
                    break;
                case SDLK_RIGHT:
                    alien_xvel =  1;
                    break;
                case SDLK_UP:
                    alien_yvel = -1;
                    break;
                case SDLK_DOWN:
                    alien_yvel =  1;
                    break;
                default:
                    break;
            }
            break;
        /* We must also use the SDL_KEYUP events to zero the x */
        /* and y velocity variables. But we must also be      */
        /* careful not to zero the velocities when we shouldn't*/
        case SDL_KEYUP:
            switch( event.key.keysym.sym ){
                case SDLK_LEFT:
                    /* We check to make sure the alien is moving */
                    /* to the left. If it is then we zero the    */
                    /* velocity. If the alien is moving to the   */
                    /* right then the right key is still press   */
                    /* so we don't tocuh the velocity            */
                    if( alien_xvel < 0 )
                        alien_xvel = 0;
                    break;
                case SDLK_RIGHT:
                    if( alien_xvel > 0 )
```

```
                        alien_xvel = 0;
                    break;
                case SDLK_UP:
                    if( alien_yvel < 0 )
                        alien_yvel = 0;
                    break;
                case SDLK_DOWN:
                    if( alien_yvel > 0 )
                        alien_yvel = 0;
                    break;
                default:
                    break;
            }
            break;

        default:
            break;
    }
}

/* Update the alien position */
alien_x += alien_xvel;
alien_y += alien_yvel;
```

如您所见，我们用了两个变量 alien_xvel 和 alien_yvel 来表示飞船的运动，并在响应键盘消息时更新它们。

第四章 样例

注：重复的例子没有列出

最快的图像平面块传送

将图像画到屏幕上有三种方式：1.创建一个图像平面并用 SDL_BlitSurface 传送到屏幕；2.在系统内存创建视频平面并调用 SDL_UpdateRect；3.在显存创建视频平面并调用 SDL_LockSurface。最好的方法是混合方式：

```
#include <stdio.h>
#include <stdlib.h>
#include "SDL.h"
#include "SDL_timer.h"
```

```
void ComplainAndExit(void)
{
        fprintf(stderr, "Problem: %s\n", SDL_GetError());
        exit(1);
}

int main(int argc, char *argv[])
{
        SDL_PixelFormat fmt;
        SDL_Surface *screen, *locked;
        SDL_Surface *imagebmp, *image;
        SDL_Rect dstrect;
        int i;
        Uint8 *buffer;

        /* Initialize SDL */
        if ( SDL_Init(SDL_INIT_VIDEO) < 0 ) {
              ComplainAndExit();
        }
        atexit(SDL_Quit);

        /* Load a BMP image into a surface */
        imagebmp = SDL_LoadBMP("image.bmp");
        if ( imagebmp == NULL ) {
              ComplainAndExit();
        }

        /* Set the video mode (640x480 at native depth) */
        screen = SDL_SetVideoMode(640, 480, 0, SDL_HWSURFACE|SDL_FULLSCREEN);
        if ( screen == NULL ) {
              ComplainAndExit();
        }

        /* Set the video colormap */
        if ( imagebmp->format->palette != NULL ) {
              SDL_SetColors(screen,
                            imagebmp->format->palette->colors, 0,
                            imagebmp->format->palette->ncolors);
        }

        /* Convert the image to the video format (maps colors) */
        image = SDL_DisplayFormat(imagebmp);
        SDL_FreeSurface(imagebmp);
        if ( image == NULL ) {
                ComplainAndExit();
        }

        /* Draw bands of color on the raw surface */
        if ( SDL_MUSTLOCK(screen) ) {
                if ( SDL_LockSurface(screen) < 0 )
                        ComplainAndExit();
        }
        buffer=(Uint8 *)screen->pixels;
        for ( i=0; ih; ++i ) {
                memset(buffer,(i*255)/screen->h,
                              screen->w*screen->format->BytesPerPixel);
                              buffer += screen->pitch;
        }
        if ( SDL_MUSTLOCK(screen) ) {
                SDL_UnlockSurface(screen);
        }

        /* Blit the image to the center of the screen */
        dstrect.x = (screen->w-image->w)/2;
        dstrect.y = (screen->h-image->h)/2;
        dstrect.w = image->w;
        dstrect.h = image->h;
        if ( SDL_BlitSurface(image, NULL, screen, &dstrect) < 0 ) {
                SDL_FreeSurface(image);
                ComplainAndExit();
        }
        SDL_FreeSurface(image);

        /* Update the screen */
        SDL_UpdateRects(screen, 1, &dstrect);

        SDL_Delay(5000);        /* Wait 5 seconds */
        exit(0);
}
```

过滤和处理事件

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include "SDL.h"

/* This function may run in a separate event thread */
int FilterEvents(const SDL_Event *event) {
        static int boycott = 1;

        /* This quit event signals the closing of the window */
        if ( (event->type == SDL_QUIT) && boycott ) {
              printf("Quit event filtered out -- try again.\n");
              boycott = 0;
              return(0);
        }
        if ( event->type == SDL_MOUSEMOTION ) {
              printf("Mouse moved to (%d,%d)\n",
                          event->motion.x, event->motion.y);
              return(0);    /* Drop it, we've handled it */
        }
        return(1);
}

int main(int argc, char *argv[])
{
        SDL_Event event;

        /* Initialize the SDL library (starts the event loop) */
        if ( SDL_Init(SDL_INIT_VIDEO) < 0 ) {
              fprintf(stderr,
                          "Couldn't initialize SDL: %s\n", SDL_GetError());
              exit(1);
        }

        /* Clean up on exit, exit on window close and interrupt */
        atexit(SDL_Quit);

        /* Ignore key events */
        SDL_EventState(SDL_KEYDOWN, SDL_IGNORE);
        SDL_EventState(SDL_KEYUP, SDL_IGNORE);

        /* Filter quit and mouse motion events */
        SDL_SetEventFilter(FilterEvents);
```

```
/* The mouse isn't much use unless we have a display for reference */
if ( SDL_SetVideoMode(640, 480, 8, 0) == NULL ) {
    fprintf(stderr, "Couldn't set 640x480x8 video mode: %s\n",
                    SDL_GetError());
    exit(1);
}

/* Loop waiting for ESC+Mouse_Button */
while ( SDL_WaitEvent(&event) >= 0 ) {
    switch (event.type) {
        case SDL_ACTIVEEVENT: {
            if ( event.active.state & SDL_APPACTIVE ) {
                if ( event.active.gain ) {
                    printf("App activated\n");
                } else {
                    printf("App iconified\n");
                }
            }
        }
        break;

        case SDL_MOUSEBUTTONDOWN: {
            Uint8 *keys;

            keys = SDL_GetKeyState(NULL);
            if ( keys[SDLK_ESCAPE] == SDL_PRESSED ) {
                printf("Bye bye...\n");
                exit(0);
            }
            printf("Mouse button pressed\n");
        }
        break;

        case SDL_QUIT: {
            printf("Quit requested, quitting.\n");
            exit(0);
        }
        break;
    }
}
/* This should never happen */
```

```
        printf("SDL_WaitEvent error: %s\n", SDL_GetError());
        exit(1);
}
```

打开音频设备

```
SDL_AudioSpec wanted;
extern void fill_audio(void *udata, Uint8 *stream, int len);

/* Set the audio format */
wanted.freq = 22050;
wanted.format = AUDIO_S16;
wanted.channels = 2;    /* 1 = mono, 2 = stereo */
wanted.samples = 1024;  /* Good low-latency value for callback */
wanted.callback = fill_audio;
wanted.userdata = NULL;

/* Open the audio device, forcing the desired format */
if ( SDL_OpenAudio(&wanted, NULL) < 0 ) {
    fprintf(stderr, "Couldn't open audio: %s\n", SDL_GetError());
    return(-1);
}
return(0);
```

播放音频

```
static Uint8 *audio_chunk;
static Uint32 audio_len;
static Uint8 *audio_pos;

/* The audio function callback takes the following parameters:
   stream:  A pointer to the audio buffer to be filled
   len:     The length (in bytes) of the audio buffer
*/
void fill_audio(void *udata, Uint8 *stream, int len)
{
    /* Only play if we have data left */
    if ( audio_len == 0 )
        return;

    /* Mix as much data as possible */
    len = ( len > audio_len ? audio_len : len );
    SDL_MixAudio(stream, audio_pos, len, SDL_MIX_MAXVOLUME)
```

```
        audio_pos += len;
        audio_len -= len;
}

/* Load the audio data ... */

;;;;;

audio_pos = audio_chunk;

/* Let the callback function play the audio chunk */
SDL_PauseAudio(0);

/* Do some processing */

;;;;;

/* Wait for sound to complete */
while ( audio_len > 0 ) {
        SDL_Delay(100);         /* Sleep 1/10 second */
}
SDL_CloseAudio();
```

列出所有 CDROM

```
#include "SDL.h"

/* Initialize SDL first */
if ( SDL_Init(SDL_INIT_CDROM) < 0 ) {
        fprintf(stderr, "Couldn't initialize SDL: %s\n",SDL_GetError());
        exit(1);
}
atexit(SDL_Quit);

/* Find out how many CD-ROM drives are connected to the system */
printf("Drives available: %d\n", SDL_CDNumDrives());
for ( i=0; i<SDL_CDNumDrives(); ++i ) {
        printf("Drive %d:  \"%s\"\n", i, SDL_CDName(i));
}
```

打开缺省 CDROM 驱动器

```
SDL_CD *cdrom;
CDstatus status;
char *status_str;

cdrom = SDL_CDOpen(0);
if ( cdrom == NULL ) {
    fprintf(stderr, "Couldn't open default CD-ROM drive: %s\n",
                    SDL_GetError());
    exit(2);
}

status = SDL_CDStatus(cdrom);
switch (status) {
    case CD_TRAYEMPTY:
        status_str = "tray empty";
        break;
    case CD_STOPPED:
        status_str = "stopped";
        break;
    case CD_PLAYING:
        status_str = "playing";
        break;
    case CD_PAUSED:
        status_str = "paused";
        break;
    case CD_ERROR:
        status_str = "error state";
        break;
}
printf("Drive status: %s\n", status_str);
if ( status >= CD_PLAYING ) {
    int m, s, f;
    FRAMES_TO_MSF(cdrom->cur_frame, &m, &s, &f);
    printf("Currently playing track %d, %d:%2.2d\n",
    cdrom->track[cdrom->cur_track].id, m, s);
}
```

列出 CD 上所有音轨

```
SDL_CD *cdrom;      /* Assuming this has already been set.. */
```

```
    int i;
    int m, s, f;

    SDL_CDStatus(cdrom);
    printf("Drive tracks: %d\n", cdrom->numtracks);
    for ( i=0; i<cdrom->numtracks; ++i ) {
        FRAMES_TO_MSF(cdrom->track[i].length, &m, &s, &f);
        if ( f > 0 )
            ++s;
        printf("\tTrack (index %d) %d: %d:%2.2d\n", i,
        cdrom->track[i].id, m, s);
    }
```

播放 CD

```
    SDL_CD *cdrom;      /* Assuming this has already been set.. */

    // Play entire CD:
    if ( CD_INDRIVE(SDL_CDStatus(cdrom)) )
        SDL_CDPlayTracks(cdrom, 0, 0, 0, 0);

        // Play last track:
        if ( CD_INDRIVE(SDL_CDStatus(cdrom)) ) {
            SDL_CDPlayTracks(cdrom, cdrom->numtracks-1, 0, 0, 0);
        }

        // Play first and second track and 10 seconds of third track:
        if ( CD_INDRIVE(SDL_CDStatus(cdrom)) )
            SDL_CDPlayTracks(cdrom, 0, 0, 2, 10);
```

基于时间的游戏主循环

```
#define TICK_INTERVAL    30

Uint32 TimeLeft(void)
{
    static Uint32 next_time = 0;
    Uint32 now;

    now = SDL_GetTicks();
    if ( next_time <= now ) {
        next_time = now+TICK_INTERVAL;
        return(0);
        }
        return(next_time-now);
}


/*  main  game  loop

        while  (  game_running  )  {
              UpdateGameState();
              SDL_Delay(TimeLeft());
        }
```