关于 sdl 的翻译(主要是介绍部分)

1.介绍

SDL 是一个由 8 个子系统组成——音频、CDROM、事件驱动、文件 I/O、操作杆驱动、线程、时钟和视频。在你使用这些子系统之前，你必须要初始化他们，使用 SDL_Init（或者 SDL_InitSubSystem。SDL_Init 必须在其他 SDL 功能被调用之前调用。所以初始化默认的子系统和视频子系统你需要调用 SDL_Init(SDL_INIT_VIDEO)，而初始化默认的子系统、视频系统和时间系统你需要调用 SDL_Init (SDL_INIT_VIDEO | SDL_INIT_TIMER)）

SDL_Init 的退出使用 SDL_Quit（和 SDL_QuitSubSystem）SDL_Quit 关掉所有子系统，包括默认的，它应该被调用在 SDL 应用退出之前。

SDL_Init 和 SDL_Quit 坚固的嵌入你的程序工具箱，你能写你的第一个和最基础的 SDL 应用。但是，我们必须准备处理错误。许多 SDL 功能返回一个值指出这个功能是成功或者失败，例如，SDL_Init 返回 -1 如果不能初始化一个子系统。SDL 提供一个有用的设备允许你精确的定位错误的类型，每次一个 SDL 内错误发生一个错误信息被存储，使用 SDL_GetError 可以找到。经常使用这个，你能知道错误的很多。

例子 1——1 初始化 SDL

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

2.图形和视频

2.1SDL 的视频介绍

视频也许是最常用的 SDL 的应用地方，所以它有最全套的子系统。这里是一些例子来显示这个基础。

2.1.1 初始化视频显示

例子 2——1 初始化视频显示

```
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);
    }
```

2.1.2 初始化最好的视频模式

如果你有一个偏好一个可靠的像素深度但是将要接受其他的，那么如下使用 SDL_SetVideoMode 和 SDL_ANYFORMAT。你也能使用 SDL_VideoModeOK() 来找到当地视频模式来尽量匹配你要求的模式。

例子 2——2 初始化最好的视频模式

```
 /* 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);
```

2.1.3 读取和显示一个 bmp 文件

下面的功能读取和显示一个作为参数给出的 bmp 文件，前提 SDL 被初始化视频模式被设置。

例子 2——3 读取和现实一个 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);
}
```

2.1.4 直接绘图来显示

以下的两个功能能够用于得到和设置单独的一个平面的像素。他们仔细的被写和任何当前 SDL 的深度工作。基础要锁定这个表面再调用他们，在调用其他 SDL 功能之前需要解锁。

在像素值和他们的红、绿、蓝部件之间转换，使用 SDL_GetRGB() 和 SDL_MapRGB()。
例子 2——4getpixel()

```
/*
 * 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 */
    }
}
```

下面的代码使用 putpixel()功能来设置一个黄色像素在屏幕中间
例子 2——6 使用 putpixel()

```
 /* 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;
```

2.2 在 sdl 中使用 opengl

SDL 有能力创建和使用 opengl 环境在一系列的平台上（Linux/X11，Win32，BeOs，MacOs，Classic/Toolbox，MacOs X，FreeBSD/X11 和 Solaris/X11）。允许你使用 SDL 的音频，事件处理，线程和定时器在你的 opengl 应用（一个功能经常被 GLUT 执行）。

2.2.1 初始化

初始化 SDL 来使用 opengl 是非常不同于正常的初始化 SDL。有三个不同;你必须传递 SDL_OPENGL 到 SDL_SetVideoMode，你必须详说明一系列 GL 属性（深度缓冲尺寸，帧缓冲尺寸）通过使用 SDL_GL_SetAttribute 和最终的，如果你希望来使用“双缓冲”你必须详说明它作为一个 GL 属性，不通过传送 SDL_DOUBLEBUF 标记到 SDL_SetVideoMode

例子 2——7 初始化 SDL 的 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 );
}
```

2.2.2 绘制

相比初始化，在 SDL 中使用 opengl 与在其他 api 中使用 opengl 是相同的，e.g.GLUT。你还是使用所有同样功能的调用和数据结构。但是如果你在使用一个双缓冲显示，你必须使用 SDL_GL_GetAttribute 来看你是否真正的获得了它。

一个全面的例子显示如下。

例子 2——8SDL 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;
}
```

3 输入处理

3.1 控制杆处理

3.1.1 初始化

在一个 SDL 程序中使用一个控制杆的第一步是初始化 SDL 的控制杆子系统。传入 SDL_INIT_JOYSTICK 标志到 SDL_Init 函数可以完成这一步。这个控制杆标记通常与其他标志关联（像 video 标志）因为这个控制杆经常用于控制某些东西。

例子 3——1 初始化 SDL 支持控制杆

```
if ( ! SDL_Init( SDL_INIT_VIDEO | SDL_INIT_JOYSTICK ) )
    {
        fprintf(stderr, "Couldn't initialize SDL: %s\n", SDL_GetError());
        exit(1);
    }
```

这个将尝试启动 SDL 的视频和控制杆子系统。

3.1.2 询问

如果我们通过询问测试，我们能安全的假设 SDL 库已经被初始化且控制杆子系统开始工作。我们现在能调用一些视频和（或）音频功能在我们需要控制杆之前。最后我们必须确认有一个真实的控制杆在工作。即使你知道一个控制杆将在在系统上，它也希望检查因为它能帮助识别控制杆被拔去的时候。这个用于检查控制杆的功能是 SDL_NumJoysticks。

这个功能简单的返回控制杆启动数目。如果它至少一个。接下来一步是决定哪个控制杆用户想使用。如果这个控制杆活动个数只有一个然后它会安全的假设这个控制杆是用户想使用的。SDL 有一个功能来获取控制杆由系统分配的名字的功能 SDL_JoystickName。这个控制杆被划分为从 0 开始作为第一个控制杆和最后一个控制杆，这个数目被 SDL_NumJoysticks - 1 返回。示范一个所有活动的控制杆清单被打印到 stdout。

例子 3——2 询问活动控制杆个数

```
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));
    }
```

3.1.3 打开一个控制杆接受控制杆事件

SDL 的事件驱动可以打开一个控制杆工作。控制杆能引发 4 种类型的事件

SDL_JoyAxisEvent　事件出发当一个坐标改变

SDL_JoyBallEvent　事件出现当一个控制杆跟踪球位置改变

SDL_JoyHatEvent　事件出现当一个帽子的位置改变

SDL_JoyButtonEvent 事件出现当一个按钮按下或者释放

事件从所有打开的控制杆接受。为了接受控制杆事件首先做的事情是调用 SDL_JoystickEventState 加上 SDL_ENABLE 功能。例如我们我们只对系统中第一个控制杆事件感兴趣，不管它是啥。从它接受事件我们要做这个：

例 3——3 打开一个控制杆

```
SDL_Joystick *joystick;

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

如果我们想接受其他控制杆事件我们需要打开他们而调用 SDL_JoystickOpen 就像我们打开控制杆 0 一样，除非我们要存储 SDL_Joystick 结构体（他们返回一个不同类型的指针）我们只需要控制杆指针当我们询问这个控制杆或者当我们关闭这个控制杆。

这之上所有代码我们只是用于初始化控制杆为了实时读取值。所有我们现在需要的是一个事件循环，所有 SDL 程序应该有任何方法来接受系统退出事件。我们现在必须添加代码来检查事件循环中至少一些以下注意到的事件。让我们假设我们事件循环像这样：

```
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 */
```

处理控制杆事件我们只不过为他们添加 case，首先我们将添加坐标处理代码，坐标检查能获取棘手的 kinda 因为许多接受的控制杆事件是垃圾。控制杆坐标有一个微小改变趋势只是在它们设计的轨迹路径之间。你必须设置一个门槛来补偿改变和忽略事件如果它们没有超出门槛。10%是一个通常的很好的门槛值。这个听起来比它本身更复杂。这里是一个坐标事件处理

例子 3——4 控制杆坐标事件

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

其他坐标事件特技例如上——下、左——右动作是两个不同不同的轴设置。最重要的坐标是坐标 0（左——右）和坐标 1（上——下）。在代码中处理它们我们要做以下：

例子 3——5 更多控制杆坐标事件

```
    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;
```

理想的代码应该使用 event.jaxis.value 来缩放某些东西。例如让我们假设你正使用控制杆来控制太空船的动作。如果用户正使用一个类似的控制杆且他们推动这个杆移动一点，飞船会移动一点。设计你的代码更好的用于这个情况因为它是为使用户类似的控制更好和同样的用户数字控制经验。

如果你的控制杆有任何另外的坐标然后人啊们也许被用于其他杆或者阀门控制且这些坐标返回值为 event.jaxis.axis 值。

按钮处理是简单对比坐标检查。

例子 3——6 控制杆按钮事件

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

按钮检查比坐标检查简单因为一个按钮只有按下或者没有按下。这个SDL_JOYBUTTONDOWN事件被触发当一个按钮被按下且这个SDL_JOYBUTTONUP事件被触发当一个按钮被释放。我们必须知道啥子按钮被按下，读取event.jbutton.button可以完成这个事件。

最后当我们完成使用我们的控制杆我们应该关闭他们调用SDL_JoystickClose。关闭我们开启控制杆0我们应该在程序的末尾做这件事：

```
SDL_JoystickClose(joystick);
```

3.1.4 优化的控制杆功能

关心好控制那么你能使用在世界上的每种控制杆，但是有一小部分设备SDL不能支持。Joyball是我们下一个清单目标，它们与我们用的轴有很多的不同。Joyball存储相对改变不像轴存储绝对改变。一个轨迹球事件即包含了x轴的改变又包含了y的改变。下面是例子：

例子3——7控制杆球事件

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

下面的检查第一个joyball在控制杆上。这个位置的改变将被存储到event.jball.xrel和event.jball.yrel。最后我们有这个hat事件。Hats报告只有它们直接压入的。我们检查hat's位置使用bitmasks：

```
SDL_HAT_CENTERED
SDL_HAT_UP
SDL_HAT_RIGHT
SDL_HAT_DOWN
SDL_HAT_LEFT
```

也有一些预定义联合体如下：

```
SDL_HAT_RIGHTUP
SDL_HAT_RIGHTDOWN
SDL_HAT_LEFTUP
SDL_HAT_LEFTDOWN
```

Hat的例子如下：

例子3——8控制杆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;
```

另外询问控制杆数量和他们的名字有另外的功能来询问连接的控制杆能力：

SDL_JoystickNumAxes 返回控制杆的轴数

SDL_JoystickNumButtons 返回控制杆的按钮数量

SDL_JoystickNumBalls 返回控制杆球的个数

SDL_JoystickNumHats 返回控制杆 hats 的个数

使用这些功能我们必须传入控制杆结构体在我们打开控制杆获得的。例如：

例子 3——9 询问控制杆特性

```
    int number_of_buttons;
    SDL_Joystick *joystick;

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

这个代码将会得到第一个控制杆的按钮个数。

3.2 处理键盘

3.2.1 键盘相关结构体

你如果熟悉数据类型在键盘接受时将会更加容易明白这个指导，所以我将首先解释他们。

3.2.2SDLKey

SDLKey 是一个枚举类型定义在 SDL/include/SDL_keysym.h，具体见表（这个表自己看看英文文档去吧 ，实在太多）。每一个 SDLKey 记号描述一个键，SDLK_a 与键盘上的 a 按键是一致的，SDLK_SPACE 与键盘上的空格键是一致的，等等。

3.2.3SDLMod

SDLMod 是一个枚举类型，与 SDLKey 类似，但是它枚举键盘的控制键(Control,Alt,Shift)。这个完整的控制器记号见表（还是看英文文档）SDLMod 值能联合起来描述一些控制。

3.2.4SDL_keysym

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

这个 SDL_keysym 结构体描述一个按键按下或者一个按键释放。其中 scancode 参数是硬件描述，它应该被忽略除非你知道你在做什么。这个 sym 参数是被按下或者释放的 SDLKey 的键盘值。这个 mod 参数描述键盘控制器在按下或者释放发生状态。所以一个 KMOD_NUM|KMOD_CAPS|KMOD_LSHIFT 的意思是 Numlock,Capslock 和左边 shift 按键全部按下（或者在锁住按键时启动）。最后，这个 unicode 参数存储键盘的 16 位的 unicode 码。

注意：应该注意和明白这个参数只有当 SDL_keysym 正在描述一个按键按下有效时候，不是按键释放的时候。Unicode 值只有在按键按下才有意义因为这个 unicode 值描述一个国际符号且只有按键按下才产生符号。要查询 unicode 的信息要去它的网站去找。

注意：unicode 翻译必须使用 SDL_EnableUNICODE 起作用。

3.2.5SDL_KeyboardEvent

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

这个 SDL_KeyboardEvent 描述一个键盘事件（显然的）。这个 SDL_Event 联合体的 key 参数成员是一个 SDL_KeyboardEvent 结构体。这个 type 参数描述一个事件是键盘释放（SDL_KEYUP）或者是一个键盘按下（SDL_KEYDOWN）事件。这个 state 参数是非常的多余，它报告与 type 同样的信息但是用不同的值（SDL_RLEASED 和 SDL_PRESSED）。这个 keysym 包含键盘按下或者释放信息。

3.2.6 读取键盘事件

从事件队列中读取键盘事件是非常简单的（这个事件队列和它的使用在表中有描述）。我们读取事件使用 SDL_PollEvent 在一个 while() 循环且检查 SDL_KEYUP 和 SDL_KEYDOWN 事件使用一个 switch 结构，像这样

例子 3——10 读取键盘事件

```
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;
  }
}
.
.
```

这个是一个非常基础的例子。没有键盘按下或者释放的信息被说明。我们将搜索其他最全面的例子如下——报告一个键盘事件所有可能的信息。

3.2.7 一个更加详细的查看

在我们能读取 SDL 事件之前，SDL_Init 必须被开启且视频模式必须被设置使用 SDL_SetVideoMode。但是有两个其他功能我们必须使用来获得所有响应信息，使用 SDL_GetKeyName。

注意：unicode 编码值 ＜0x80 会直接翻译为一个 ASCII 符号编码值。这个在下面的例子中使用。

例子 3——11 说明键盘事件信息

```
#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" );
    }
```

3.2.8 游戏模式输入

我发现人们在游戏中和其他使用键盘事件和其他交互应用总是一直不明白一个 fundemental 点。

键盘事件只有在一个按键从没有按下到按下的状态发生时候，且取代多功能。

想象你有一个图像想从外面移动过来使用光标按键——当你按下左箭头按键你想让它滑动到左边，当你按下这个向下按键你想向下滑动。测试下面的代码，它是许多人会犯的错误。

```
/* 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;
                }
            }
        }
    }
    .
    .
```

首先你也许认为这是一个完美的响应代码，但是这不是。像我说的键盘事件只有当一个键状态改变时发生，所以用户将必须按下且释放这个左箭头按键 100 次来移动像右边 100 像素。

避免这个问题我们必须不使用这个事件来改变外部的位置，我们使用事件来设置用于一个区分选择代码移动到外部标记。就像这样：

例子 3——12 适当的游戏移动

```
/* 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;
```

可以被看到的，我们使用两个额外的差异，alien_xvel 和 alien_vvel，描述了船的移动，这些差异我们下载当我们发现键盘按下和释放。