《计算机游戏程序设计》
二维游戏基本编程

主要内容

- 二维游戏的概述
- SDL开源媒体库介绍
- 游戏地图的创建
- 精灵动画
- 二维游戏世界的模拟

二维游戏

- 早期的游戏都是二维的
  - 如Diablo（暗黑破坏神）
  - 只有两个轴（绘制好的背景）
- 很多游戏采用的是二维半

二维游戏

- 二维游戏对现在的编程仍然有意义（特别是现在的手机游戏、Web游戏等）：如内存、分辨率
- 本质上视频游戏是一个连续的循环，执行逻辑指令，并将图像输出到屏幕。这和电影的播放非常类似，但是这个电影是用户指定的

二维游戏的基本流程和架构

- 简单的调度模块是一个do-while循环，串行执行：交互→输出→再输出→再渲染→……
- 游戏地图的加载和编辑
- 游戏角色的加载
- 图像的半透明技术
- 精灵动画技术
- 碰撞检测技术

SDL

- Simple DirectMedia Layer
- DirectX的一个替代品
- 开源、跨平台
- 简单易用、调试方便
- 执行效率高
- 功能强大

SDL_Video、SDL_Image、OpenGL —— DirectDraw、Direct3D
SDL_Audio、SDL_Mixer —— DirectSound
SDL_Joystick、SDL_Base —— DirectInput
SDL_Net —— DirectPlay
SMPEG、SDL_Video、SDL_Audio、SDL_Sound、SDL_Filter —— DirectShow

初始化

- 调用SDL_Init()动态的加载和初始化SDL库， 该函数带有一组标记来表示哪部分需要激活：
  - SDL_INIT_AUDIO
  - SDL_INIT_VIDEO
  - SDL_INIT_CDROM
  - SDL_INIT_TIMER
- 使用完毕最后调用SDL_Quit()。

```
#include <stdlib.h>
#include "SDL.h "
main(int argc, char *argv[])
{
    if ( SDL_Init(SDL_INIT_AUDIO|SDL_INIT_VIDEO) < 0 )
    {
        fprintf(stderr, "无法初始化SDL: %s\n", SDL_GetError());
        exit(1);
    }
    atexit(SDL_Quit);
     ...
}
```


事件处理

- 等待：调用SDL_WaitEvent()等待事件

```
SDL_Event event;
SDL_WaitEvent(&event);
switch (event.type)
{
     case SDL_KEYDOWN: break;
    case SDL_QUIT:  exit(0);
}
```

- 轮询：调用SDL_PollEvent()函数

```
SDL_Event event;
while ( SDL_PollEvent(&event) )
{
   switch (event.type)
   {
      case SDL_MOUSEMOTION:  break;
      case SDL_MOUSEBUTTONDOWN: break;
      case SDL_QUIT: exit(0);
   }
}
```


选择视频模式

```
{
    SDL_Surface *screen;
    screen = SDL_SetVideoMode(640, 480, 16, SDL_SWSURFACE);
    if ( screen == NULL )
        {
            fprintf(stderr, “无法设置640x480的视频模式：s\n", SDL_GetError());
            exit(1);
        }
}
```

- SDL_SWSURFACE 软件模式
- SDL_HWSURFACE 硬件模式
- SDL_ASYNCBLIT 异步Blit
- SDL_ANYFORMAT 实际显示面的像素格式
- SDL_HWPALETTE 256色调色板
- SDL_DOUBLEBUF 双缓冲
- SDL_FULLSCREEN 全屏显示
- SDL_OPENGL 使用OpenGL
- SDL_OPENGLBLIT 使用OpenGL但是用SDL绘制
- SDL_RESIZABLE 可改变大小的窗口
- SDL_NOFRAME 不需要其他窗口装饰、全屏时默认

一段完整代码

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "SDL/SDL.h"
#include "windows.h"

SDL_Surface *screen = NULL;      //屏幕接口


int main( int argc, char* args[] )  //主程序
{
    char msg[500];                  // 得到系统消息
    int done;


    /* 初始化SDL */
    if (SDL_Init (SDL_INIT_VIDEO) < 0)
    {
      sprintf (msg, "不能初始化 SDL: %sn", SDL_GetError ());
      MessageBox (0, msg, "Error", MB_ICONHAND);
      exit (1);
    }
    atexit (SDL_Quit);

    /* 设置 640x480 16-bits 图象模式 */
    screen = SDL_SetVideoMode (640, 480, 16, SDL_SWSURFACE |
SDL_DOUBLEBUF);

    if (screen == NULL)
    {
      sprintf (msg, "不能设置成 640x480x16 图象模式: %sn",
        SDL_GetError ());
      MessageBox (0, msg, "Error", MB_ICONHAND);
      exit (2);
    }

    /*设置SDL窗口标题 */
    SDL_WM_SetCaption ("hello,the world", NULL);

    done = 0;

    //游戏循环 done=1时退出，done=0时继续
    while (!done)
    {
      SDL_Event event;

      /* Check for events */
      while (SDL_PollEvent (&event))
      {
        switch (event.type)
        {
        case SDL_KEYDOWN:
            break;
        case SDL_QUIT:
            done = 1;
            break;
        default:
            break;
        }
      }

    }
    SDL_Quit();
    return 0;
}
```


图像显示

- SDL_Surface
- SDL_BlitSurface()
- SDL核心库只支持BMP格式

```
void ShowBMP(char *file, SDL_Surface *screen, int x, int y)
{
    SDL_Surface *image;
    SDL_Rect dest;
    /* 将BMP文件加载到一个surface*/
    image = SDL_LoadBMP(file);
    if ( image == NULL )
    {
        fprintf(stderr, "无法加载 %s: %s\n", file, SDL_GetError());
        return;
    }
     /* Blit到屏幕surface*/
    dest.x = x; dest.y = y;
    dest.w = image->w; dest.h = image->h;
    SDL_BlitSurface(image, NULL, screen, &dest);
    /* 刷新屏幕的变化部分 */
    SDL_UpdateRects(screen, 1, &dest);
}
```


SDL学习资料

- P. Ernest. *Focus on SDL*, Premier Press, 2003
- http://www.cppblog.com/lf426/category/6107.html?Show=All
- http://www.lampchina.net/article/htmls/201008/Mjk2MTc3.html
- http://www.cnblogs.com/Henrya2/category/181879.html
- http://hi.baidu.com/dvalyan/blog/item/d1f557fb6a7e169c59ee900d.html
- http://www.libsdl.org/intro.cn/toc.html

图像表示

- 文件格式
  - BMP, TGA, TIFF, GIF, JPEG 等
- 定义(Bitmap)
  - 位图图像是一块由彩色点集组成的矩形区域
- DIB(Device Independent bitmap)
  - 设备无关位图
- DDB(Device-Dependent Bitmaps)
  - 设备有关位图：老的Windows系统

BMP结构

- BMP文件头
  - 比较简单的头信息
- 位图信息
  - 关于数据尺寸的详细信息
- 调色板(optional)：相当于一个查找表
  - RGB四元组
- 位图数据
  - RGB象素值
  - 索引值(如果有调色板)

BMP头

```
typedef struct {
   WORD   bfType;              //"BM",表示该文件为位图
   DWORD bfSize;               //文件大小(以byte计)
   WORD   bfReserved1;         //保留位
   WORD   bfReserved2;
   DWORD bfOffbits;
                  //实际位图起始位置相对于文件头的偏移量
} BITMAPFILEHEADER
```


位图信息

```
typedef struct {
   DWORD biSize;          //= sizeof(BITMAPINFOHEADER)
   DWORD biWidth;         // 位图宽度(以象素计)
   DWORD niHeight;
                              //正数，左下角为起始点，从下向上
                              //负数，左上角为起点，从上向下
   WORD    biplanes;      //color plane数，恒等于1
   WORD    biBitCount;    // 1, 4, 8, 24, 32
   DWORD biCompression; //通常，以下域可以忽略，置为0即可
   DWORD biSizeImage;
   DWORD biXPelsPerMeter;
   DWORD biYPelsPerMeter;
   DWORD biClrUsed;     //位图使用的颜色
   DWORD biClrImportant; //重要颜色数
} BITMAPINFOHEADER
```


SDL图像库

- SDL_image
- 调用：SDL_Surface* IMG_Load(char* filename)
- 支持BMP, PNM, XPM, LBM, PCX, GIF, JPEG, TGA和PNG
- 下载：http://www.libsdl.org/projects/SDL_image/release/SDL_image-1.2.8.tar.gz

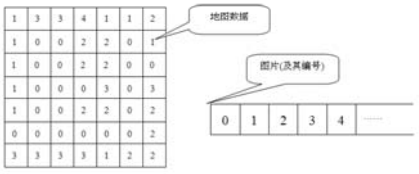
地图的创建与表示

- 为实现一个基本的二维游戏框架，首先要实现游戏地图的各种加载和编辑操作，为角色提供游戏环境。
- 4种通用地图实现的方法：固定地图、滚屏地图、多层次地图、菱形地图



固定地图

- 使用固定的背景作为地图
- 将屏幕切割成棋盘状的一系列小块
- 在内存中保持一个二维数组，保存每个小块对应的编号
- 绘制时根据数组提供的信息，在每个小块画上相应图块



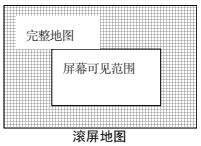
固定地图拼接算法

```
for (yi = 0; yi<y轴小地图个数; yi++){
    for(xi = 0; xi<x轴小地图个数; xi++){
        int screenx = xi * tile_wide;
                    //乘以宽度得到最后屏幕上位置
        int screeny = yi * tile_high;
        int tileid = mapping_matrix[yi][xi];
                //在这个数组中存放着对应位置的小地图编
                //号，如1表示水，2表示石头，3表示砖等
        blit(tillid,screenx,screeny);
                //自编函数，把相应地图贴到正确位置
    }
}
```

- 该程序段中blit函数的实现，可以调用SDL系统的一个API函数SDL_BlitSurface()直接实现

滚屏地图

- 是固定地图的进一步扩展，可以显示远大于固定地图的图像
- 根据玩家所在位置，确定显示的地图部分



滚屏地图

滚屏地图算法–声明及描述

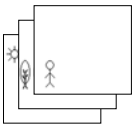
- 变量设置：
  playerx, playery 为人物相对于完整地图左上角的坐标；
  screen_wide, screen_high 为屏幕的宽和高；
  xtile为屏幕上*x*轴上可显示的小地图个数；
  ytile为屏幕上*y*轴上可显示的小地图个数；
  tileplayerx = playerx / tile_wide 为人物所在格*x*轴下标；
  tileplayery = playery / tile_high 为人物所在格*y*轴下标；
- 应该绘制的地图范围是：
  *x*轴： 由 tileplayerx – xtile/2 至 tileplayerx + xtile/2；
  *y*轴： 由 tileplayery – ytile/2 至 tileplayery + ytile/2；
- 当人物在屏幕正中央时，地图到屏幕的位置变化公式为：
  screenx = xi * tile_wide – playerx + 0.5 * screen_wide
  screeny = yi * tile_high – playery + 0.5 * screen_high

滚屏地图算法–实现

```
int beginx = tileplayerx - xtile/2
int endx = tileplayerx + xtile/2
int beginy = tileplayery - ytile/2
int endy = tileplayery + ytile/2
tileplayerx = playerx / tile_wide
tileplayery = playery / tile_high
for (yi = beginy; yi<endy; yi++){
   for(xi = beginx; xi<endx; xi++)
   {
      int screenx = xi * tile_wide – playerx + 0.5 * screen_wide;
                                                  //屏幕上位置
      int screeny = yi * tile_high – playery + 0.5 * screen_high;
      int tileid = mapping_matrix[yi][xi];        //地图数据数组
      blit(tillid,screenx,screeny);
      //自编函数，将相应地图贴到正确位置
   }
}
```


多层次地图

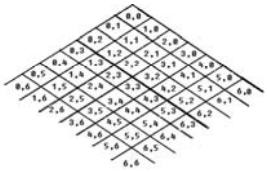
- 以下列情况，可以考虑使用多层次地图。
  - 需要小地图能重叠或者有层次关系；
  - 在背景上有多个物体运动；
  - 需要模拟物体远近不同的透视关系；



- 多层次地图的实现思想并不复杂,在滚屏地图的基础上设置多个层次的地图即可。不妨设从底往上分别为0层，1层，…把地图数据数组改为三维数组。
- 可以使每个图层以不同的速度运动,模拟景物远近不同的层次感。这种技术，又称视差卷轴（Parallax Scrollers）。

菱形地图（Isometric Map）

- 菱形地图是在二维画面上表现三维场景的常用技术



- 拼接所使用的小地图是菱形，计算比较复杂

菱形地图的算法

```
int MapDraw(HDC hdc){
    int i,j;
    int lim=MAXSCREENX/TILEWIDE;  //所需绘制地图的范围
    for (i=-lim; i<lim; i++){
            for (j=-lim; j<lim; j++){
                    int sx=MAXSCREENX/2-(TILEWIDE/2)+(i*TILEWIDE/2)-
    (j*TILEWIDE/2);
                    int sy=MAXSCREENY/2-
    (TILEHIGH/2)+(i*TILEHIGH/2)+(j*TILEHIGH/2);
                    if((sx<MAXSCREENX) && (sy<MAXSCREENY) && (sx+TILEWIDE>0)
                    && (sy+TILEHIGH>0) && (playerx+i<100 && playerx+i>=0)
                    && (playery+j<100 && playery+j>=0) ){          //边界判断
                    TransparentBlt(hdcMem,sx,sy,TILEWIDE,TILEHIGH,
                    hdcTiles[Data[playerx+i][playery+j]],
                    0,0,TILEWIDE,TILEHIGH,RGB(0,255,0)); //贴图
                    }
            }
    }
    BitBlt(hdc,0,0,MAXSCREENX,MAXSCREENY,hdcMem,0,0,SRCCOPY);//画到窗口
return 0;
}
```


图像的半透明操作

- 每种颜色都由红绿蓝3种基本色彩（三原色）组合而成；
- 三原色中每一种颜色的亮度用一个8位的二进制数来表示
- 半透明图色彩 = 源图像色彩×（100% - 透明度）+背景图像色彩×透明度
- Windows API函数：AlphaBlend

精灵动画简介

- 基于精灵的人物表现
- 鬼怪ghosts,精灵 sprites,骑士 knights
- 精灵：前景是图像，背景是透明的
- 精灵动画：将上一帧中精灵出现的地方用背景填充，并在新的指定地点绘制精灵

图像镂空（抠色）

- 将掩码图和背景图案进行按位AND，使得原始图像的对应位置变空。
- 将原始图像和上一步处理结果按位OR。
- 这样，原始图像贴到背景上并遮盖背景，其余部分（掩码图中白色部分）没有贴到背景上。

原始图像

掩码图

SDL ColorKeying

```
{
    Uint32 colorkey = SDL_MapRGB(pSurface->format, r, g, b);
    SDL_SetColorKey(pSurface, flag, colorkey );
}
```

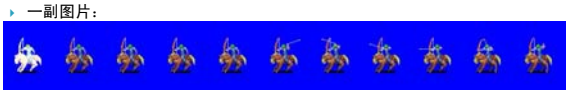
- 其中SDL_MapRGB()返回需要抠掉的颜色
- SDL_SetColorKey()将指定的颜色设为键值

精灵动画实现

- 英文为sprite animation
  - 一幅背景图
  - 一组模板图（mask）
  - 人物的连续显示方式
    - 双缓冲机制
    - 不要在窗口中直接贴图，避免闪烁
      - 建立一个内存DC，然后把所有的贴图动作都在这个DC上进行，最后把结果显示到操作窗口中。
- 对动画序列中的每一帧
  - Load 背景图
  - 确定sprite绘画的位置
  - 将某一掩码图与背景图作AND运算
  - 将对应的人物图与背景图作OR运算
  - 更新sprite绘画的位置

精灵动画举例

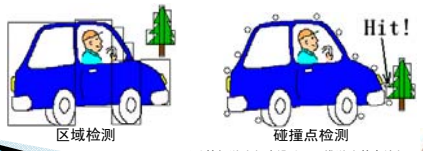
- 一副图片：



```
精灵帧数=0；
游戏循环
{
    是否有按键
    {
      有就处理
    }
    显示背景
    如果精灵帧数大于等于10，精灵帧数等于0
    显示精灵帧数所示的图片部分
    精灵帧数加1
    更新屏幕
}
```


碰撞检测

- 对运动物体的碰撞判断是许多游戏程序中不可或缺的要素
- 常见的碰撞检测方法
  - 区域检测
  - 碰撞点检测
  - 颜色检测：较为精确，相对耗时



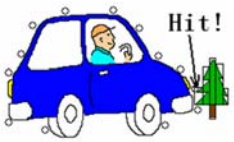
区域检测　　碰撞点检测

区域检测

- 采用某种规则形状逼近物体
- 物体之间的碰撞检测转化为规则形状之间的检测

碰撞点监测

- 本质是区域检测的一种
- 一般在两个运动物体中的一个物体上设置碰撞点，在另一个物体上设置检测区域，运行时逐个判断碰撞点是否在检测区域中。



颜色检测

- 为树林做一张掩码图，将树林用黑色填充。要产生汽车驶入树林后面的效果，先在背景上贴上汽车的图像，然后在上面用镂空图技术画上树林。然后，判断汽车图像在树林图像上的相对位置，将汽车图像上的点和掩码图上相应位置的点做按位AND操作，检查结果中是否有黑色点（RGB值为0）存在。任何颜色的RGB值与黑色图形进行按位AND运算，将得到黑色。如果存在黑色点，表明有碰撞。

图像操作

- 全局操作
  - 对图像上的所有象素作同样的操作

  如：傅立叶变换、直方图统计、块拷贝、灰度转换、缩放等
- 局部操作
  - 操作只与象素及其周围邻居的值有关

  如：滤波、边缘检测等

直方图

- 表示图像中象素颜色值的分布
- 直方图上每一点
  - 横坐标：颜色（亮度）值
  - 纵坐标：图像中具有该颜色（亮度）值的象素的数目
- 亮度增强、增加对比度

几何运算

- 平移
  - 将图像沿坐标轴移动若干偏移量
- 缩放
  - 整数倍放大
  - 整数倍缩小
  - 一般情况：缩放系数非整数的情况
- 旋转
  - 旋转矩阵R及其逆矩阵R^{-1}

图像滤波算子

- 本质上实现的时候都是将某个象素的新的值用邻域象素值的加权平均计算而得。

例如：边界增强算子

垂直方向算子

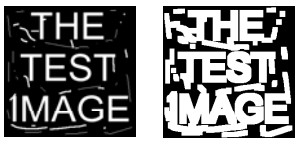
-1	0	1
-2	0	2
-1	0	1

水平方向算子

1	2	1
0	0	0
-1	-2	-1

形态算子

- 膨胀



- 腐蚀

Morphing

- 图像处理中最有意思的效果
  - 某个物体伸展到另外一个物体
  - 通常利用网格辅助
  - 也涉及很多计算机视觉的知识

风格化图像

- 用多种合成滤波方法产生油画风格

- 利用类比的方法产生更多风格各异的图像