书签 ： P576

# 目录

## [游戏概述](#目录)

Triple-A ： 暗黑破坏神3，星际争霸，上古卷轴，使命召唤，战地3，战争机器2，巫师，求生之路2，最终幻想，鬼泣，火炬之光，刺客信条，孤岛危机，暗黑血统，真三国无双，生化危机，波斯王子，极品飞车。。。

QQ农场、会说话的汤姆猫，愤怒的小鸟，神庙逃亡。。。

社交休闲类网页游戏 flash开发；

PC，3A 大作都是C++开发，windows平台下， C++，win API，图形库(DirectX,OpenGL)；

Android用Java，IOS用OC，Windows Phone用C#，脚本用Python与Lua；

开发方式：win API、MFC、win Forms(.NET)

封装的不适合游戏的效率（MFC）；

GDI写游戏效率底下，游戏引擎就是对图形库DirectX或OpenGL的再封装；GDI+封装了GDI，效率远不如GDI，功能换性能；

### [游戏简史](#目录)

**id software** 公司，pc游戏及引擎开发；半条命half-lift，反恐精英counter-strike就是用Quake制作，93年发行毁灭战士DOOM，里程碑；

#### 《DOOM启示录》阅读心得

约翰·卡马克(John Carmack) 和 约翰·罗梅洛(John Romero) 是《毁灭战士》和《雷神之锤》的创造者，3D游戏的里程碑；

约翰·罗梅洛 小时候沉迷于街机，即使父亲每次都要将他打成香肠嘴；

乔布斯和沃兹 给罗梅洛带来了苹果机，一切是这么的巧合；

罗梅洛把大部分的时间都花在游戏上；

卡马克是“天才班”的：黑客，偷盗，炸药，“精神失常；

后来也是 把大部分时间花在游戏上；

有能力的年轻人相互吸引，一个乐于编写游戏引擎，一个乐于做各种辅助工具和策划游戏；一个编程的天才，一个艺术的大师，一对完美的CP，就像“苹果双雄”一样；

在每个晚上，音乐伴随着他们工作到凌晨，就像是不打烊的派对；

时代在进步，新的技术，新的算法，精英不断加入，创新在继续，成功像2的幂次方一样，虽说白手起家一统江湖的故事很难再上演，但这其中的精神仍在延续；

DOOM like a bomb！

DND（Dungeons & Dragons）桌面角色扮演游戏（TRPG）的鼻祖；每个玩家控制扮演虚拟世界中的一个人物，这个人物有包括性格、外貌、装备、法术、特殊技能、力量等等的属性，他/她的种族可能是精灵、矮人、人类，而职业则可以是法师、刺客、商人、吟游诗人等等不一而足。然后几个玩家一起在特定场景（比如指环王的“中土世界”）下进行游戏冒险。另外还会有一个人担任仲裁者的角色（在“龙与地下城”的系统中即被称为DM——地下城主），同时也担任说书人；

### [学习路线](#目录)

1. C++编程
2. windows API，GDI
3. DirectX
4. 学习三维游戏引擎
5. 开源游戏源码

## MCI接口

MP3等音乐文件 是采用 压缩算法保存的音频文件，要播放就必须知道解码方法，转换成音频流格式播放出来；

MCI（媒体控制接口）操作系统内置的解码引擎，支持avi，wav，mpeg，mp3，wma等；

编码：

头文件<Mmsystem.h>

库 Winmm.lib

Command String 和 Command Message， 一个用mciSendString 一个用mciSendMessage，string就是先转msg；

有4种Command， System由MCI直接处理；Required由MCI driver处理；Basic是某些设备的；Extended是特殊设备的；

全部:<https://msdn.microsoft.com/en-us/library/windows/desktop/dd797783(v=vs.85).aspx>

常用的，MCI\_OPEN, MCI\_CLOSE, MCI\_STATUS, MCI\_PLAY, MCI\_PAUSE

使用

命令：对应一个或多个结构，如MCI\_OPEN，MCI\_OPEN\_PARMS，一些设备有额外的结构作为参数传入；

一个多媒体文件，对应一个 device ID， 打开设备是另外一个操作，你可以只通过id获取文件信息而不打开设备；

api ：mciSendCommand，mciGetDeviceID…

宏：MCI\_HMS\_HOUR 得到小时，H（hour）M（minutes）S（second）F（frame）T（tracks）

通知：MCI\_NOTIFY 通知目标窗口完成，dwCallBack的低字节是目标窗口，由窗口过程处理；

MCIERROR mciSendCommand(

 MCIDEVICEID IDDevice, //设备ID

 UINT        uMsg,//命令

 DWORD\_PTR   fdwCommand,//命令的Flags

 DWORD\_PTR   dwParam //命令的参数

);

0 成功，否则错误码; 详见MSDN；

想找如何实现你的需求，先大致判断该用什么命令，然后在MSDN 搜索 相应的命令 就会进入Content，如MCI\_STATUS；

## 五子棋

无禁手规则，黑棋必胜，所以要限制（太无聊了，不开发，只学习过程）

过程

一、需求说明书

1）五子棋的需求分析

1. 开发目的
2. 游戏用户
3. 项目背景（项目提出者。。。）
4. 游戏功能描述
5. 其它

2）可行性分析

1. 功能要求合理
2. 开发软件及硬件环境要求，运行环境
3. 界面要求
4. 完成期限
5. 经费，开发人员，开发技术。。。
6. 收益，投入 计算
7. 其它

二、项目计划安排

1，表格

序号 任务 起始时间 周期 人员安排 等

1 需求及可行性分析 2016-12-18 3 A

4，错误处理。。。

三、总体设计文档

1，系统功能架构

2，游戏流程图

3，接口设计（类图等）

5，界面（能整出个效果图 是最好的了，用PS等）

四、详细设计

1）网络通信

分析需求 ，网络传输的数据有：

1. 棋子坐标
2. 游戏控制信息，如“和棋”
3. 聊天等 ， 扩展业务

定义数据包 ：

|  |  |  |
| --- | --- | --- |
| 数据长度 2B | 消息类型 1B | 数据 nB |

各种数据 详细的结构定义，如棋子坐标：行，列，颜色；

和棋 包， 无数据；

同意和棋包 无数据；

拒绝和棋包 无数据；

。。。

2）交互界面

想怎么弄，就怎么弄；

3）核心算法

1，棋子类， 棋盘， 容器

备忘录模式， 绘制棋子， 内存dc

1. 鼠标左键弹起

3，游戏规则判断

搜索 数组中“-----”的数据 是否有连5， 竖向连5， 斜向连5， 都没有， 禁手判断；

## DirectX

### [SDK下载及介绍](#SDK下载及介绍)

SDK 和 运行时环境（dll，网上下）；

DirectX 9 功能强劲， 10,11 并没有大的变革；OpenGL 主攻专业图形 和 3D游戏，DirectX 9 主攻娱乐应用；

官网下载 DirectX SDK安装；

VS，VC++包含目录属性 添加 安装后的include文件夹，vs竟然有$(DXSDK\_DIR)include；库目录 ：$(DXSDK\_DIR)lib\$(Platform)；添加库文件 d3d9.lib 和 d3dx9.lib 也可以代码加；

组件介绍

带头大哥：Direct3D；June 2010版本集成了 D3D 9，10，11；大版本下根据实际功能又分多个小版本如D3D 9.0、D3D 9.0a。。。；各组件功能独立，单独使用，单独更新；

Direct2D 新东西，不推荐用， D3D9接口ID3DXSprite可以开发2D；

DirectWrite ：Direct2D的文字渲染；

DXGI （Graphics Infrastructure）；

Direct3D：3D相关操作；

XAudio2 ：底层音频处理API，替换了DirectSound，XAudio只能用于Xbox；

XACT3：建立在XAudio2上的音频处理API，更容易；

XInput：Xbox 360输入，不支持键盘和鼠标；

XNA Math：数学库；

DirectCompute：D11加入，支持GPU的通用多线程计算；

DirectSetup：检查DirectX版本；

Windows Game Explorer：游戏资源管理器；

DirectInput：侦测键盘、鼠标、操作杆输入；

文件组成

Developer Runtime：dll；

Documentation：帮助文档，DirectX9\ directx\_sdk.chm、windows\_graphics.chm；

Extras：额外的dll；

Redist：历史版本中的一些重要dll；

Samples：看懂并运用，去世界上任何一家游戏公司做主程都够格！

Media：用到的资源；

SampleBrowser：样例浏览器；

C++：源码；

system：安装和卸载相关；

Utilities：数十款好用的工具；

Direct10 搭载了坑爹的操作系统Vista，不用；11 在国内还没普及，资料也少，支持的引擎也少； Direct11 和 9比，是技术架构的完全颠覆，也就是要从头学11，而且好多大作仍是用的9；综上，用 9；

![计算机生成了可选文字: Window'｝·砍·）！」护犷｛卜
Dir口《t3o八尸l
GDI/GDl+
硬件抽象从
(H凡）
设备明‘动程序1麦l](DDI)
」，!](data:image/png;base64,)

如果你的显卡支持Direct3D的话， HAL就存在， 它是绘制高性能高质量精美图形的；

HAL 充分利用硬件的加速功能，又实现了 设备无关性（直接调用 DirectX API ，硬件制造商 会在HAL 里提供相应接口）；

### [COM组件的一些概念](#COM组件的一些概念)

COM对象 -> C++类， COM接口，COM接口对象，COM对象接口都差不多意思；

COM接口对象 –> 包含 特定方法 的内存结构；

应用程序不能直接访问 COM接口对象，而是通过 接口对象的接口指针来执行方法 ；

C++与COM 配DirectX ， 99%的大型游戏开发方案；

引用计数 控制生命周期， 不用new，delete；

### [Direc3D 初始化4步曲](#Direc3D初始化4步曲)

创建Direct3D接口对象；

获取设备硬件信息；

填充D3DPRSENT\_PARAMETERS结构体；

创建Direct3D设备接口；

//创建Direct3D接口对象（函数 数组）

LPDIRECT3D9 pD3D = NULL;

if (NULL == (pD3D = Direct3DCreate9(D3D\_SDK\_VERSION)))

return E\_FAIL;

//获取系统硬件信息：显卡。。

HRESULT GetDeviceCaps(

UINT 显卡序号，D3DADAPTER\_DEFAULT；

D3DDEVTYPE 设备类型，D3DDEVTYPE\_HAL P312

[out] D3DCAPS9 \*pCaps 显卡所有性能参数；

);

D3DCAPS9 caps;

int vp = 0;

if (FAILED(pD3D->GetDeviceCaps(D3DADAPTER\_DEFAULT, D3DDEVTYPE\_HAL, &caps)))

{

return E\_FAIL;

}

if (caps.DevCaps & D3DDEVCAPS\_HWTRANSFORMANDLIGHT)

vp = D3DCREATE\_HARDWARE\_VERTEXPROCESSING; //硬件顶点

else

vp = D3DCREATE\_SOFTWARE\_VERTEXPROCESSING;

顶点是3D图形学的基本元素，Direc3D 有 硬件顶点（高富帅） 和 软件顶点运算；

//填充D3DPRSENT\_PARAMETERS 结构体

typedef struct \_D3DPRESENT\_PARAMETERS\_

{

UINT BackBufferWidth; //后台缓冲区宽度

UINT BackBufferHeight;

D3DFORMAT BackBufferFormat;//缓冲区保存的像素格式，D3DFORMAT,GetDisplayMode获取当前像素格式;

UINT BackBufferCount;

D3DMULTISAMPLE\_TYPE MultiSampleType;//多重采样类型，D3DMULTISAMPLE\_NONE

DWORD MultiSampleQuality;//多重采样格式，0

D3DSWAPEFFECT SwapEffect;//如何将后台缓冲区复制到前台，D3DSWAPEFFECT\_DISCARD

HWND hDeviceWindow;//绘制的窗口句柄，NULL当前激活的窗口

BOOL Windowed;//TRUE窗口模式，FALSE全屏模式

BOOL EnableAutoDepthStencil;//是否自动管理深度缓存

D3DFORMAT AutoDepthStencilFormat;//深度缓存的像素格式, D3DFORMAT

DWORD Flags;//附加属性，0

/\* FullScreen\_RefreshRateInHz must be zero for Windowed mode \*/

UINT FullScreen\_RefreshRateInHz;//全屏下屏幕刷新率D3DPRESENT\_REATE\_DEFAULT，窗口模式下0

UINT PresentationInterval;//指定后台缓冲区与前台缓冲区的交换频率，D3DPRESENT

} D3DPRESENT\_PARAMETERS;

D3DPRESENT\_PARAMETERS d3dpp;

ZeroMemory(&d3dpp, sizeof(d3dpp));

d3dpp.BackBufferWidth = WINDOW\_WIDTH;

d3dpp.BackBufferHeight = WINDOW\_HEIGHT;

d3dpp.BackBufferFormat = D3DFMT\_A8R8G8B8;

d3dpp.BackBufferCount = 1;

d3dpp.MultiSampleType = D3DMULTISAMPLE\_NONE;

d3dpp.MultiSampleQuality = 0;

d3dpp.SwapEffect = D3DSWAPEFFECT\_DISCARD;

d3dpp.hDeviceWindow = hwnd;

d3dpp.Windowed = TRUE;

d3dpp.EnableAutoDepthStencil = TRUE;

d3dpp.AutoDepthStencilFormat = D3DFMT\_D24FS8;

d3dpp.Flags = 0;

d3dpp.FullScreen\_RefreshRateInHz = 0;

d3dpp.PresentationInterval = D3DPRESENT\_INTERVAL\_IMMEDIATE;

//创建设备接口对象

HRESULT CreateDevice(

[in] UINT Adapter,//显卡序号 D3DADAPTER\_DEFAULT

[in] D3DDEVTYPE DeviceType,//设备类型

[in] HWND hFocusWindow,//程序从前台转后台的提示窗口，一般和d3dpp.hDeviceWindow一致

[in] DWORD BehaviorFlags,//D3DCREATE\_HARDWARE\_VERTEXPROCESSING 和 软件顶点运算

[in, out] D3DPRESENT\_PARAMETERS \*pPresentationParameters,//第三步结构

[out, retval] IDirect3DDevice9 \*\*ppReturnedDeviceInterface//设备接口指针

);

//创建设备接口对象

IDirect3DDevice9\* idd9;

if (FAILED(pD3D->CreateDevice(D3DADAPTER\_DEFAULT, D3DDEVTYPE\_HAL,

hwnd, vp, &d3dpp, &idd9)))

{

return E\_FAIL;

}

### [D3D动画显示技术](#D3D动画显示技术)

#### [交换链](#D3D动画显示技术)

由2个以上的表面组成；交换链利用指针 操作，实现高效流畅的动画绘制；

D3D通常在一系列后台缓冲区中生成动画帧，再通过交换链技术，逐个翻转到前台显示；

在d3dpp.BackBufferCount = 2指定，在CreateDevice() 中创建；

三维坐标 转 二维坐标（世界坐标到屏幕坐标），D3D中称为 顶点坐标变换（矩阵完成）；

两套渲染体系

固定功能流水线

遵循固定准则渲染，11已经移除；

可编程渲染流水线

着色器shader，顶点着色器vertex shader，像素着色器pixel shader，为shader编写一些代码，然后再GPU上编译运行；

固定功能渲染流水线

第一阶段：坐标变换 和 光照处理（Transforming & Lighting）

第二阶段：光栅化，完成变换和光照后的 顶点组织成点线面，纹理贴图等；

#### [顶点缓存](#顶点缓存)

3D模型都是通过多边形网格逼近的，顶点是基础元素，顶点缓存是其表现形式，可以在内存，显存中；

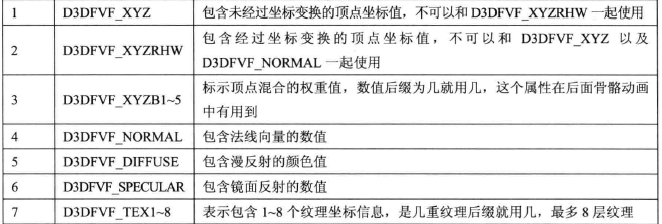
顶点缓存四部曲

设计顶点缓存

固定功能流水线——灵活顶点格式

struct CUSTOMVERTEX{ float x,y,z; DWORD color; } 随心所欲定义

顶点属性 都有宏对应



定义 配套宏，要按下面的顺序：

C:\Users\Administrator\AppData\Roaming\Tencent\Users\920746980\QQ\WinTemp\RichOle\8I%{F1_[FXO$P`241D1}ALS.png

2）创建顶点缓存 及索引缓存

IDirect3DVertexBuffer9 顶点缓存接口对象 LPDIRECT3DVERTEXBUFFER9

HRESULT CreateVertexBuffer(

[in] UINT Length,//顶点缓存大小，字节

[in] DWORD Usage,//附加属性 D3DUSAGE\_DYNAMIC，P343

[in] DWORD FVF,//顶点格式，宏组合

[in] D3DPOOL Pool,//缓存位置，D3DPOOL\_DEFAULT ，P343

[out, retval] IDirect3DVertexBuffer9 \*\*ppVertexBuffer,//顶点缓存接口对象 金钥匙

[in] HANDLE \*pSharedHandle//保留,0

);

HRESULT CreateIndexBuffer(

[in] UINT Length,//索引缓存大小，字节

[in] DWORD Usage,//附加属性 D3DUSAGE\_DYNAMIC P360

[in] D3DFORMAT Format,//索引大小，D3DFMT\_INDEX16，16位

[in] D3DPOOL Pool,//缓存位置，D3DPOOL\_DEFAULT，显存中 P361

[out, retval] IDirect3DIndexBuffer9 \*\*ppVertexBuffer,//所有缓存接口对象 金钥匙

[in] HANDLE \*pSharedHandle//保留,0

);

D3D的三种内存

AGP内存，显存，内存；早期显存16M，32M不够用，向内存借，这就是AGP；

D3DUSAGE\_DYNAMIC 创建动态缓存，放在AGP中，更新时因为要 移到显存中，所以会慢一点；

3）访问顶点缓存 及索引缓存

HRESULT Lock(

[in] UINT OffsetToLock,//字节，存储的开始位置到 开始锁定位置的偏移

[in] UINT SizeToLock,//加锁区域大小

[out] VOID \*\*ppbData,//被锁区域起始地址

[in] DWORD Flags//锁定方式，0，D3DLOCK\_DISCARD P362

);

赋值顶点

VOID\* pVertices;

if (FAILED(g\_pVertexBuffer->Lock(0, sizeof(Vertices), (void\*\*)&pVertices, 0)))

return E\_FAIL;

memcpy(pVertices, Vertices, sizeof(Vertices));

g\_pVertexBuffer->Unlock();

绘制

//将顶点缓存关联固定流水线

HRESULT SetStreamSource(

[in] UINT StreamNumber,//与顶点缓存连接的数据流，0

[in] IDirect3DVertexBuffer9 \*pStreamData,//顶点缓存接口

[in] UINT OffsetInBytes,//数据流中的偏移量，字节，0

[in] UINT Stride //顶点结构大小

);

HRESULT SetFVF(DWORD FVF //宏，灵活顶点格式

);

HRESULT SetIndices( IDirect3DIndexBuffer9\* ) //设置索引缓存

HRESULT DrawPrimitive( //有索引缓存，就用DrawIndexedPrimitive

[in] D3DPRIMITIVETYPE PrimitiveType,//绘制的图元类型，P349

[in] UINT StartVertex,//顶点缓存的起始索引

[in] UINT PrimitiveCount//图元数量

);

SetRenderState(D3DRS\_SHADEMODE, D3DSHADE\_GOURAUD);//设置渲染状态

D3DSHADE\_GOURAUD 高洛德着色模式

#### [索引缓存](#索引缓存)

HRESULT DrawIndexedPrimitive(

[in] D3DPRIMITIVETYPE Type, //图元类型 ，P349

[in] INT BaseVertexIndex,//做索引目录的起始顶点位置

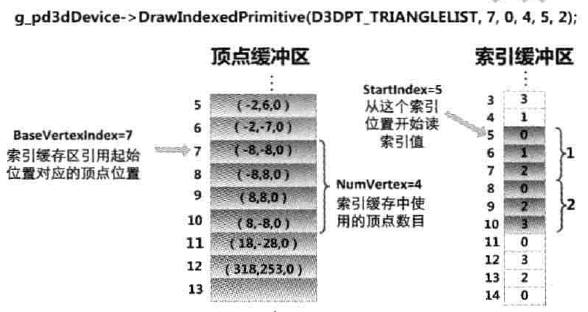
[in] UINT MinIndex,//最小索引,0

[in] UINT NumVertices,//顶点个数

[in] UINT StartIndex,//开始绘制图元的索引

[in] UINT PrimitiveCount //绘制的图元个数

);



缓冲区第7个，取4个顶点 拿来索引（它们的索引就是0，1，2，3）， 从索引缓冲区 第5个位置 开始读索引（事先建立好的）；

WORD Indices[] = { 0,1,2, 0,2,3, 0,3,4, 0,4,5, 0,5,6, 0,6,7, 0,7,8, 0,8,9, 0,9,10, 0,10,11 ,0,11,12, 0,12,13 ,0,13,14 ,0,14,15 ,0,15,16, 0, 16,1 };

### [Direct3D文本绘制](#Direct3D文本绘制)

ID3DXFont接口负责创建字体和文本绘制；

HRESULT D3DXCreateFont(

\_\_in LPDIRECT3DDEVICE9 pDevice, //D3D设备指针

\_\_in INT Height,//字高

\_\_in UINT Width,

\_\_in UINT Weight,//权重

\_\_in UINT MipLevels,//过滤属性

\_\_in BOOL Italic,//斜体

\_\_in DWORD CharSet,//字符集，DEFAULT\_CHARSET

\_\_in DWORD OutputPrecision,//精度，OUT\_DEFAULT\_PRECIS

\_\_in DWORD Quality,//输出质量，DEFAULT\_QUALITY

\_\_in DWORD PitchAndFamily,//字体索引号，0

\_\_in LPCTSTR pFacename,//字体名

\_\_out LPD3DXFONT \*ppFont //字体指针

);

INT DrawText(

[in] LPD3DXSPRITE pSprite, //字符串所属的ID3DXSprite对象接口，0表示在当前窗口绘制

[in] LPCTSTR pString, //字符串

[in] INT Count, //长度，-1会绘制到结束

[in] LPRECT pRect,//绘制的矩形区域

[in] DWORD Format,// P325

[in] D3DCOLOR Color

);

Direct3D11 移除了上面的东西， 要使用第三方库、资源和软件才能输出中文；

### [Direct3D渲染五部曲](#Direct3D渲染五部曲)

清屏

开始场景

绘制

结束场景

翻转显示

HRESULT Clear(

[in] DWORD Count,//pRects的数量

[in] const D3DRECT \*pRects,//清空的矩形数组

[in] DWORD Flags,//要清空的缓冲区，D3DCLEAR\_STENCIL/TARGET/ZBUFFER，分别是模板，颜色，深度缓冲区

[in] D3DCOLOR Color,//清空颜色缓冲区后每个像素的颜色，D3DCOLOR\_XRGB(R,G,B)

[in] float Z,//清空深度缓冲区后每个像素的深度值

[in] DWORD Stencil//清空模板缓冲区后每个像素的模板值

);

g\_pd3dDevice->BeginScene(); //设备接口对象

绘制过程 千变万化；

绘制的结果在幕后；

g\_pd3dDevice->EndScene();

//翻转

HRESULT Present(

[in] const RECT \*pSourceRect,//源矩形区域，NULL

[in] const RECT \*pDestRect,//目标，NULL

[in] HWND hDestWindowOverride,//绘制窗口句柄，0就取 D3DPRESENT\_PARAMETERS的窗口

[in] const RGNDATA \*pDirtyRegion//最小更新区域，NULL

);

### [四大变换](#四大变换)

#### 概念

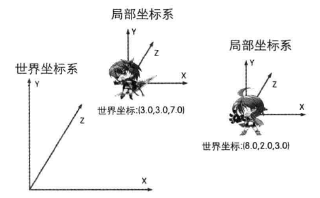
世界变换 –> 为了在指定位置绘图

取景变换 -> 为了以不同的视角观察图形

投影变换 -> 为了远近变化

视口变换 -> 为了控制输出图形

#### [世界变换](#世界变换)



平移、旋转、缩放；

HRESULT SetTransform(

[in] D3DTRANSFORMSTATETYPE State,//变换类型，D3DTS\_WORLD世界矩阵，P374

[in] const D3DMATRIX \*pMatrix //认定该矩阵就是我们的世界变换矩阵了（或是其它）

);

D3DXMATRIX \* D3DXMatrixTranslation( //创建一个平移矩阵

\_\_inout D3DXMATRIX \*pOut,

\_\_in FLOAT x,//偏移量

\_\_in FLOAT y,

\_\_in FLOAT z

);

D3DXMATRIX \* D3DXMatrixMultiply(//矩阵相乘（平移）operator\* 也是可以的

\_\_inout D3DXMATRIX \*pOut,

\_\_in const D3DXMATRIX \*pM1,

\_\_in const D3DXMATRIX \*pM2

);

D3DXMATRIX m;

D3DXMatrixTranslation(&m, 5, -3, 0);

m2 \*= m; //m2平移了m

D3DXMATRIX \* D3DXMatrixRotationX(//旋转，Y,Z都有

\_\_inout D3DXMATRIX \*pOut,

\_\_in FLOAT Angle //弧度

);

m2 = m \* m2; //右乘

D3DXMATRIX \* D3DXMatrixScaling(//缩放

\_\_inout D3DXMATRIX \*pOut,

\_\_in FLOAT sx,//缩放比例

\_\_in FLOAT sy,

\_\_in FLOAT sz

);

先缩放，再旋转，最后平移；

D3DXMATRIX \* D3DXMatrixIdentity( //单位化矩阵

\_\_inout D3DXMATRIX \*pOut

);

#### [取景变换](#取景变换)

变换虚拟摄像机的位置和观察点；

D3DXMATRIX \* D3DXMatrixLookAtLH(

\_\_inout D3DXMATRIX \*pOut,//取景变换矩阵

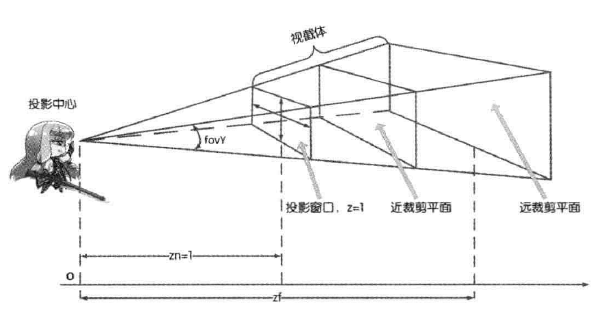
\_\_in const D3DXVECTOR3 \*pEye,//虚拟摄像机在世界坐标中的位置

\_\_in const D3DXVECTOR3 \*pAt,//观察点位置

\_\_in const D3DXVECTOR3 \*pUp//方向，（0,1,0）

);

#### [投影变换](#投影变换)



投影中心：虚拟摄像机

投影窗口：z=1的二维平面

视截体：投影中心+投影窗口， 后面可见的三维空间

投影变换就是 将视截体内的物体映射到投影窗口中；

D3DXMATRIX\* WINAPI D3DXMatrixPerspectiveFovLH(

D3DXMATRIX \*pOut, //投影变换矩阵

FLOAT fovy, //弧度，虚拟摄像机在y轴上的成像角度，视域角度

FLOAT Aspect, //屏幕宽/高

FLOAT zn, FLOAT zf//近/远裁剪面到摄像机的距离

);

#### [视口变换](#视口变换)

将投影窗口中的图形 转换到 屏幕窗口上；

视口可以一个矩形区域；

typedef struct D3DVIEWPORT9 {

DWORD X;//视口相对窗口的x坐标

DWORD Y;

DWORD Width;//视口宽度

DWORD Height;

float MinZ;//视口在深度缓存中的最小深度值

float MaxZ;

} D3DVIEWPORT9, \*LPD3DVIEWPORT9;

g\_pd3dDevice->SetViewport(&vp);

### [光照与材质](#光照与材质)

增加真实感的；D3D中，我们只要告诉其光照类型，物体材质参数 以及 光源方向，D3D就会根据其内置算法，计算每个顶点的颜色值；

更高阶：运用各种功能的着色器，自己编写光照效果；

#### 四大光照类型

环境光Ambient Light

没有刻意光源， pd3dDevice->SetRenderState( D3DRS\_AMBIENT, 颜色);

漫反射光 Diffuse Light

光到达表面，沿各个方向反射；

镜面反射光 Specular Light

光照到光滑表面上，如镜子等上面；计算量大，D3D默认关闭；

自发光 Emissive Light

自己发出的光；

typedef struct \_D3DLIGHT9 {

D3DLIGHTTYPE Type; //光源类型,D3DLIGHT\_POINT P393

D3DCOLORVALUE Diffuse; //漫反射颜色值

D3DCOLORVALUE Specular; //镜面反射

D3DCOLORVALUE Ambient; //环境光

D3DVECTOR Position; //光源位置

D3DVECTOR Direction; //光源方向

float Range; //光照范围

float Falloff; //Falloff,Theta,Phi 聚光灯用

float Attenuation0; /\* 衰减系数 Constant attenuation \*/

float Attenuation1; /\* Linear attenuation \*/

float Attenuation2; /\* Quadratic attenuation \*/

float Theta; /\* Inner angle of spotlight cone \*/

float Phi; /\* Outer angle of spotlight cone \*/

} D3DLIGHT9;

 D为光源到顶点的距离，A0，A1，A2对应系数；

HRESULT SetLight(

[in] DWORD Index, //0~7,选择第1~8个光源

[in] const D3DLight9 \*pLight

);

HRESULT LightEnable(

[in] DWORD LightIndex,//0~7,选择第1~8个光源

[in] BOOL bEnable //true 启用光源

);

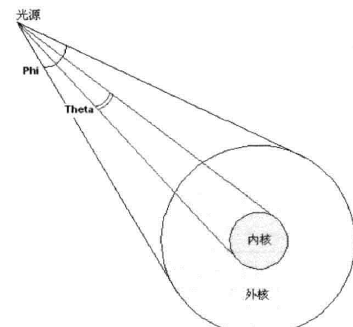
#### 三大光源

点光源Point Light，方向光 Directional Light， 聚光灯Spot Light

点光源 具有颜色和位置，没有方向；

方向光 具有颜色和方向，不受衰减和范围影响（不用设置不相关参数）；

聚光灯 演唱会的聚光灯，锥形的，光线强度由内向外衰减；



Phi 和 Theta，上图；

Falloff 控制光强从 内锥到外锥 的衰减；

开销最大；

D3DLIGHT9 light;

::ZeroMemory(&light, sizeof(light));

//…

#### 材质

材质决定了物体能反射什么颜色以及能反射多少；

typedef struct \_D3DMATERIAL9 {

D3DCOLORVALUE Diffuse; //漫反射的反射率

D3DCOLORVALUE Ambient; //环境光

D3DCOLORVALUE Specular; //镜面反射

D3DCOLORVALUE Emissive; //自发光的颜色值

float Power; //镜面反射指数

} D3DMATERIAL9;

物体顶点颜色 = 各种反射率综合决定



SetMaterial( const D3DMATERIAL9 \* );

GetMaterial();

D3D默认材质是漫反射光，不反射环境光，镜面反射光，没有自发光；

#### 顶点法线

为了在光照计算时，能够知道光线到达表面时的入射角；

一般和面法线相同， 特殊情况下不同，如近似球体或圆的；

复习高中 立体几何 去吧！

#### 快速画几何体

P402

绘制四部曲 P404

### [DirectInput](#DirectInput)

#### 概念

基于消息的性能不行， DirectInput接口直接与硬件打交道；

由IDirectInput8，IDirectInputDevice8，IDirectInputEffect 这3个接口组成，47个方法； P422

IDirectInput8 初始化系统，创建输入设备接口；

IDirectInputDevice8 表示各种输入设备；

IDirectInputEffect控制设备的力反馈效果；

<DInput.h> DInput8.lib dxguid.lib

HRESULT DirectInput8Create(

HINSTANCE hinst,//

DWORD dwVersion,//DirectInput版本,DIRECTINPUT\_VERSION

REFIID riidltf,//接口标识, IID\_IDirectInput8

LPVOID \* ppvOut,//IDirectInput8接口对象指针，LPDIRECTINPUT8 金钥匙

LPUNKNOWN punkOuter//NULL

)

系统每有一个已安装的设备就有一个系统分配的全局唯一GUID；

鼠标 和 键盘的GUID：GUID\_SysMouse， GUID\_Keyboard，对于其它设备，用EnumDevices

HRESULT EnumDevices(

DWORD dwDevType,//要枚举的设备类型，DI8DEVCLASS\_DEVICE,P425

LPDIENUMDEVICESCALLBACK lpCallback,//系统每找到一个，回调

LPVOID pvRef,//[out] GUID

DWORD dwFlags//枚举设备的方式,DIEDFL\_ATTACHEDONLY, P426

)

HRESULT CreateDevice(

REFGUID rguid,//GUID

LPDIRECTINPUTDEVICE \* lplpDirectInputDevice,//设备对象\*,金钥匙

LPUNKNOWN pUnkOuter//NULL

)

HRESULT SetDataFormat( //设置数据格式，创建设备后 一定要做的事

LPCDIDATAFORMAT lpdf//设备数据，c\_dfDIkeyboard, P427

)

#### 协作级别

共享和协调应用程序对设备的访问；DirectInput 采用协作级别Cooperative Level处理机制；

前台，后台 和 共享，独占模式；

HRESULT SetCooperativeLevel(

HWND hwnd,//进程的顶级窗口

DWORD dwLevel//模式, DISCL\_BACKGROUND, P428

)

其它

HRESULT SetProperty(//设置特殊属性

REFGUID rguidProp,

LPCDIPROPHEADER pdiph//

)

HRESULT Acquire(); //获取设备的控制权

HRESULT Poll();//轮询设备，什么时候可用

HRESULT GetDeviceState(//获取设备数据

DWORD cbData,//缓冲区大小

LPVOID lpvData//和SetDataFormat设置的结构有关, P430

)

g\_pMouseDevice->Unacquire(); ->Release(); //释放

使用步骤

创建设备接口

设置数据格式和协作级别

获取设备控制权

获取按键状态并响应

释放控制权和接口对象

#### DirectInput 键码





if( g\_pKeyStateBuffer[DIK\_A] & 0x80) //…

### [纹理映射](#纹理映射)

2D图像，贴到几何体上；

#### 4步曲

纹理坐标定义

顶点访问

纹理创建

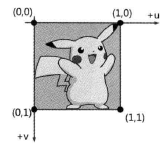
启用纹理

.bmp， dds，dib，png，tga等；

边长为2的N次幂正方形，128\*128,256\*256,512\*512等；

纹理贴图 -> 二维数组（纹理元素）， 纹理坐标 -> 纹理元素；

纹理坐标：



纹理坐标 在纹理层，灵活顶点格式可以有8层，是相对坐标；

纹理是COM对象， IDirectt3DTexture9 ，要对物体表面进行纹理映射，先要创建纹理对象，指定一些属性，然后将 纹理文件加载到 纹理对象中；

HRESULT D3DXCreateTextureFromFile(

\_\_in LPDIRECT3DDEVICE9 pDevice,

\_\_in LPCTSTR pSrcFile,//纹理文件

\_\_out LPDIRECT3DTEXTURE9 \*ppTexture//接口

);

D3DXCreateTextureFromFileEx 加强版

HRESULT SetTexture(//启用

[in] DWORD Sampler,//0~7,那一层

[in] IDirect3DBaseTexture9 \*pTexture//纹理接口

);

g\_pd3dDevice->SetRenderState(D3DRS\_LIGHTING, FALSE) 关闭需要通过光反射才能看到物体的自然界现象；

#### 纹理过滤

二维平面的颜色来自纹理， 一般纹理会比平面大或小， 如何着色就是纹理过滤；

四种方式：

最近点采样过滤

线性纹理过滤

各项异性过滤

多级渐进过滤

各有优缺点，线性较粗糙，计算量小；多级最好，和各向异性配合最佳，但计算量大；

HRESULT SetSamplerState( //设置纹理的采样属性

[in] DWORD Sampler,//0~7层

[in] D3DSAMPLERSTATETYPE Type,//纹理类型，D3DSAMP\_ADDRESSU, P473

[in] DWORD Value//和Type相关

);

最近点采样

纹理过滤元素（颜色）通常是float，最近采样会取最近的整数地址的元素颜色；

SetSamplerState(0, D3DSAMP\_MAGFILTER/D3DSAMP\_MINFILTER 放大/缩小过滤器, D3DTEXF\_POINT);

纹理大小和图元大小差不多，影响不大，否则会闪烁或失真；

线性

广泛采用的，性能影响不大；

取得纹理元素浮点地址最近的上下左右4个元素加权平均的颜色值；

SetSamplerState(0, D3DSAMP\_MAGFILTER/D3DSAMP\_MINFILTER 放大/缩小过滤器, D3DTEXF\_LINEAR);

显卡有针对性的优化；

各项异性

弯曲的表面，投影会被拉长或收缩；

SetSamplerState(0, D3DSAMP\_MAXANISOTROPY最大各向异性程度值, D3DTEXF\_ANISOTROP);

多级渐进

需要配合上面三种之一；

由一组分辨率逐渐降低的纹理序列组成，大小渐小，D3D会选择最接近的大小映射；

HRESULT D3DXCreateTextureFromFileEx(

\_\_in LPDIRECT3DDEVICE9 pDevice,//D3D设备接口

\_\_in LPCTSTR pSrcFile,//纹理文件

\_\_in UINT Width,//纹理宽度

\_\_in UINT Height,

\_\_in UINT MipLevels,//渐进级数

\_\_in DWORD Usage,//0

\_\_in D3DFORMAT Format,//纹理格式

\_\_in D3DPOOL Pool,//保存纹理的方式，P343

\_\_in DWORD Filter,//纹理过滤方式

\_\_in DWORD MipFilter,//生成纹理序列的过滤方式

\_\_in D3DCOLOR ColorKey,//替换Alpha值的颜色

\_\_inout D3DXIMAGE\_INFO \*pSrcInfo,//NULL

\_\_out PALETTEENTRY \*pPalette,//调色板,NULL

\_\_out LPDIRECT3DTEXTURE9 \*ppTexture//纹理接口

)

SetSamplerState(0, D3DSAMP\_MIPFILTER, 相邻纹理级之间的过滤方式 );

D3DTEXF\_LINEAR 最匹配的两极纹理线性混合

; D3DTEXF\_NONE, 使用最高一级的纹理

D3DTEXF\_POINT 使用与图元大小匹配的纹理

设置纹理过滤最大级数：SetSamplerState(0, D3DSAMP\_MAXMIPLEVEL，8)；

设置多级纹理映射级数的偏移值：SetSamplerState(0, D3DSAMP\_MIPLODBIAS，8);

#### 四大纹理寻址方式

图元 由 顶点组成， 顶点 去指定 纹理坐标， (u,v) -> [0.0, 1.0]， 但这个范围太小了，4种寻址方式，让我们处理超过[0.0, 1.0] 范围之外的纹理映射情况：

重复寻址模式(wrap texture address mode)

镜像纹理寻址模式 ( mirror texture address mode)

夹取纹理寻址模式 ( clamp texture address mode)

边框颜色纹理寻址( border color texture address mode)

重复

默认；允许在每个整数连接点处重复上一个整数的纹理；



纹理比平面小， 自动贴上；

手动设置模式：

SetSamplerState(0, D3DSAMP\_ADDRESSU，D3DTEXTUREADDRESS); //对u轴启用

typedef enum \_D3DTEXTUREADDRESS {

D3DTADDRESS\_WRAP = 1,//重复寻址

D3DTADDRESS\_MIRROR = 2,//镜像

D3DTADDRESS\_CLAMP = 3,//夹取

D3DTADDRESS\_BORDER = 4,//边框颜色

D3DTADDRESS\_MIRRORONCE = 5,

D3DTADDRESS\_FORCE\_DWORD = 0x7fffffff, /\* force 32-bit size enum \*/

} D3DTEXTUREADDRESS;

镜像

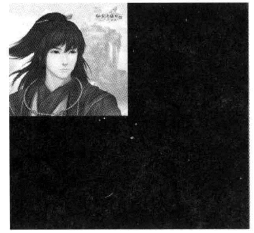


夹取



纹理坐标夹取在[0.0, 1.0]， 之外的将沿边缘向u，v轴延伸；

边框颜色



用边框颜色填充；

### [人物模型](#人物模型)

网格模型 是一种将顶点数据、纹理、材质等信息存储在一个外部文件的 3D模型；直接写代码指定顶点是不现实的，因此D3D提供了 **网格模型技术**：从特定文件读取和绘制3D图形；

网格模型普遍方式：用3D建模软件（3DS Max和Maya）生成模型，从3D模型文件中加载；

.3ds .max 是3DS Max的； .mb 是Maya的； .obj 是两者通用的；

3DS Max更适合游戏、建筑学、室内设计等，而Maya是专门为影视特效而生的，角色动画、运动学模拟等；Maya会更高级一点；

X文件

D3D中，用.X文件存储网格数据（纹理文件名，动画及用户定义的数据等）；

#### 网格模型接口

ID3DXMESH

HRESULT D3DXCreateMesh( //一般被封装在其它函数里

\_\_in DWORD NumFaces,//多边形数目

\_\_in DWORD NumVertices,//顶点数目

\_\_in DWORD Options,//附加选项，D3DXMESH\_SYSTEMMEM，P499

\_\_in const LPD3DVERTEXELEMENT9 \*pDeclaration,//顶点包含哪些数据

\_\_in LPDIRECT3DDEVICE9 pD3DDevice,//金钥匙

\_\_out LPD3DXMESH \*ppMesh //金钥匙

);

三步曲：1）从X文件加载网格模型

HRESULT D3DXLoadMeshFromX(

\_\_in LPCTSTR pFilename,//文件名

\_\_in DWORD Options,//附加选项，D3DXMESH\_SYSTEMMEM，P499

\_\_in LPDIRECT3DDEVICE9 pD3DDevice,//金钥匙

\_\_out LPD3DXBUFFER \*ppAdjacency,//网格的邻接信息

\_\_out LPD3DXBUFFER \*ppMaterials,//网格子集的材质

\_\_out LPD3DXBUFFER \*ppEffectInstances,//网格模型的特殊效果

\_\_out DWORD \*pNumMaterials,//材质子集数目

\_\_out LPD3DXMESH \*ppMesh //金钥匙

);

LPD3DXBUFFER 是泛型数据结构

(D3DXMATERIAL\*)ID3DXBuffer::GetBufferPointer();

ID3DXBuffer::GetBufferSize();

加载材质和纹理

ppMaterials 材质和纹理信息

typedef struct D3DXMATERIAL {

D3DMATERIAL9 MatD3D;//材质

LPSTR pTextureFilename;//纹理贴图文件

} D3DXMATERIAL, \*LPD3DXMATERIAL;

绘制

#### 网格模型优化

HRESULT ID3DXMesh::OptimizeInplace( //在原网格上优化

[in] DWORD Flags, //执行什么优化，P637

[in] const DWORD \*pAdjacencyIn,//邻接数组

[out] DWORD \*pAdjacencyOut,//优化后的邻接数组

[out] DWORD \*pFaceRemap, //NULL

[out] LPD3DXBUFFER \*ppVertexRemap//NULL

);

Optimize（

//… 多了一个 [out] LPD3DXMESH \*ppOptMesh //优化后的网格

）

#### 克隆网格

HRESULT CloneMeshFVF(

[in] DWORD Options,

[in] DWORD FVF,

[in] LPDIRECT3DDEVICE9 pDevice,

[out, retval] LPD3DXMESH \*ppCloneMesh

);

### [临时笔记](#临时笔记)

获取显卡型号

D3DADAPTER\_IDENTIFIER9 Adapter; //定义一个D3DADAPTER\_IDENTIFIER9结构体，用于存储显卡信息

pD3D->GetAdapterIdentifier(0, 0, &Adapter);

int len = MultiByteToWideChar(CP\_ACP, 0, Adapter.Description, -1, NULL, 0);

MultiByteToWideChar(CP\_ACP, 0, Adapter.Description, -1, g\_strAdapterName, len);

### [Alpha混合](#Alpha混合)

存储图片透明度信息的；8位的灰度通道，用256级灰度记录图像透明度信息，黑全透明，白不透明；

D3D中像素有4个分量：Alpha、R、G、B； 0~255； 0完全透明；

融合，Alpha融合公式：



RGB 像素颜色值，K 融合因子[0-1]，OP 融合方式 P513；

设置融合因子

SetRenderState( D3DRS\_SRCBLEND / D3DRS\_DESTBLEND, D3DBLEND类型，P514);

Alpha来自三处，纹理 > 材质 > 顶点 ，取一处即可，如果使用了纹理，像素的Alpha值就是纹理混合之后的；

指定Alpha值的来源；

HRESULT SetTextureStageState(

[in] DWORD Stage, //0~7层

[in] D3DTEXTURESTAGESTATETYPE Type,//枚举，纹理渲染状态

[in] DWORD Value//和Type有关

);

使用融合

启用

默认是关闭Alpha融合的，SetRenderState(D3DRS\_ALPHABLENDENABLE, true);

设置融合因子

设置融合运算方式

SetRenderState(D3DRS\_BLENDOP, 融合方式 P513);

### [深度缓冲区](#深度缓冲区)

Z缓存，存储屏幕上像素点深度信息的一块内存；屏幕分辨率1920 \* 1024，缓存区就有这么大；

深度测试使用

d3d默认开启；

1）创建缓冲区，2）开启 3）设置深度测试函数 4）更新缓冲区

D3DPRESENT\_PARAMETERS 第10，11个参数；

开启 ： SetRenderState( D3DRS\_ZENABLE, true/false);

深度函数：SetRenderState( D3DRS\_ZFUNC, D3DCMPFUNC枚举）P525

更新：SetRenderState（D3DRS\_ZWRITERENABLE， true用当前像素的深度值更新缓冲区里的／false按兵不动）

### [模版技术](#模版技术)

模版缓存（stencil buffer），专门用于制作特效的离屏缓存，分辨率和深度缓存相同，它能让我们动态地、有针对性地决定是否将某个像素写到后台缓存中；

模版测试

我们根据每个像素的模板缓存的值，进行一些检查，最后得出这个像素是否需要绘制，这个检查的过程叫做模板测试；

创建深度缓冲区的同时创建模板缓冲区，而将深度缓冲区的一部分作为模板缓冲区使用；

相关参数

SetRenderState ( 模板相关的参数介绍 P534 ) 非常多；

模版参考值，默认0；

模版掩码，决定对模版参考值和模板缓冲区值的哪位进行比较，默认0xffffffff；

模板缓冲区值，；

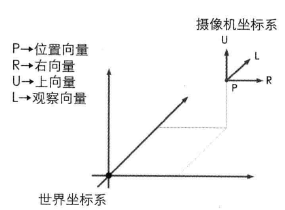
ref ，模版参考值；mask，模版掩码；value，模板缓冲区的值；op，模板比较函数；

### [摄像机](#摄像机)

顾名思义，你可以把一个摄像机 放到任意位置；

4个分量：右分量rightvector，上分量upvector，观察分量lookvector，位置分量positionvector，确定摄像机相对于世界坐标系的位置和朝向，并计算出取景变换矩阵；

right，up，look 两两垂直的单位向量，也就是标准正交，正交矩阵的逆矩阵和转置矩阵相等；



向量计算API

D3DXVECTOR3 \* D3DXVec3Normalize(

\_\_inout D3DXVECTOR3 \*pOut,

\_\_in const D3DXVECTOR3 \*pV //要规范化的向量

);

D3DXVECTOR3 \* D3DXVec3Cross( //向量叉乘

\_\_inout D3DXVECTOR3 \*pOut,

\_\_in const D3DXVECTOR3 \*pV1,

\_\_in const D3DXVECTOR3 \*pV2

);

FLOAT D3DXVec3Dot( //向量点乘

\_\_in const D3DXVECTOR3 \*pV1,

\_\_in const D3DXVECTOR3 \*pV2

);

D3DXMATRIX \* D3DXMatrixRotationAxis(//绕任意轴旋转一定角度的矩阵

\_\_inout D3DXMATRIX \*pOut,

\_\_in const D3DXVECTOR3 \*pV,//要旋转的轴

\_\_in FLOAT Angle

);

D3DXVECTOR3 \* D3DXVec3TransformCoord(//根据矩阵变换向量并规范化

\_\_inout D3DXVECTOR3 \*pOut,

\_\_in const D3DXVECTOR3 \*pV,//要变换的向量

\_\_in const D3DXMATRIX \*pM

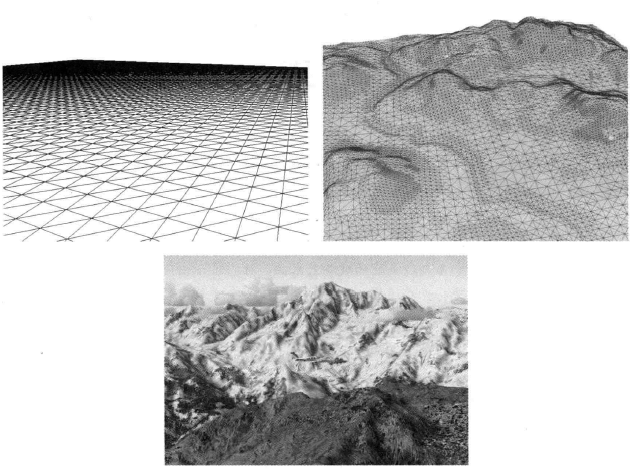
);

FLOAT D3DXVec3Length(//计算三维向量长度

\_\_in const D3DXVECTOR3 \*pV

);

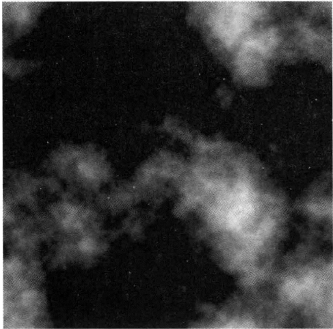
### [三维地图](#三维地图)



网格， 拉伸， 镀膜；

高度图

一个连续数组，顶点的高度； 其实有不同的实现方案，比如用灰度图grayscale map，地形的某一点海拔越高，灰度亮度越大；



每个高度元素 0~255，有比例变换；

常用的灰度图格式是 .RAW 后缀的；

制作高度图：

用算法写程序 2）图像编辑软件、三维建模等

Photoshop制作高度图： P579

代码：略；

### [三维天空](#三维天空)

### [粒子系统](#粒子系统)

火焰、爆炸、烟、水流、火花、落叶、云、雾、雪、尘、流星、光。。。。

Samples\C++\Direct3D11\NBodyGravityCS11

基本原理

粒子通常是带纹理的四边形；

广告版技术：因为主要是渲染一个多边形，首先根据观察方向生成一个旋转矩阵，然后随着观察方向的不断变化， 改变这个旋转矩阵，让多边形始终面向观众；

点精灵（Point Sprite）特殊的点元描述粒子，可以进行纹理映射并改变大小；

一般是预定义所有粒子，不用new，delete；

## 魔兽争霸3冰封王座