

硕 士 研 究 生 读 书 报 告



题目 OpenGL的三维可视化方法

作者姓名 余清波

作者学号 21860413

指导教师 李启雷

学科专业 计算机科学与技术

所在学院 工程师学院

提交日期 二○一八 年 十二 月

3D Visualization Method Based on OpenGL

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: LiQilei

By

YuQingbo

Zhejiang University, P.R. China

2018

摘要

随着三维技术的飞速发展和广泛应用，许多应用工程领域都出现了虚拟现实、科学计算可视化等三维仿真处理的应用要求。据此文介绍了OpenGL的特点和工作原理，详细论述了如何建立VC和OpenGL库应用接口及进行三维图形开发的具体步骤和基本过程，最后通过三维地形图绘制实例的过程分析，便于用户理解运用。

**关键词**:VC；OpenGL；3D visualization

Abstract

With the rapid development and wide application of 3Dtechnology, many applications in engineering fields require the application of 3Dsimulation processing such as virtual reality, scientific computing visualization and so on. The characteristics and working principle of OpenGL are introduced, and the concrete steps and basic process are discused to set up VC and OpenGL database application interface and 3Dgraphics development in detail. In order to help users understand and apply, an example of the process of 3Dtopographic map drawing is shown.

**Keywords:** VC；OpenGL；3D visualization

## 1 引言

OpenGL是SGI公司推出的一个三维图形库和模型库，提供了大量的图形变换函数，编程时可以方便地显示三维图形，无需进行繁杂的矩阵运算。OpenGL实际上是一种程序接口，它为3D加速卡硬件和3D图形应用程序搭起一座重要的桥梁。它提供了消隐、光照、着色和纹理映射等一系列函数，可以方便地对地层面进行着色、绘制和光照处理。OpenGL提供的显示列表可以提高绘制效率，采用双缓存区实现动画功能，如利用VC中ＭFC编写对鼠标的消息响应函数，通过拖动鼠标实现对三维实体的动态显示。

## ２ OpenGL的特点及工作原理

### ２.１ OpenGL特点

作为开放式的高性能图形和交互式视景处理标准，OpenGL的主要特点有:

１)函数丰富:OpenGL的函数库提供了丰富的图形、图像处理功能，主要包括:各种图形的绘制、光照变换、场景变换、坐标变换、纹理映射、多边形划分，以及像素存储格式和双缓存之间的交换等。

２)硬件无关性:OpenGL是一个与图形硬件无关的应用程序接口。用户必须从点、线、面等最

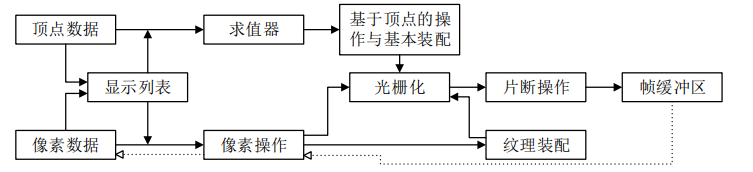
基本的图形单元开始构造自己的三维模型并进行三维实时交互。

3)操作简单:OpenGL提供了清晰明了的图形函数，初学者也能用它强大的图形处理能力快速构建出三维图形并进行交互操作。

4)可运行于网络，属于客户机/服务器型:OpenGL是网络透明的，在C/S体系结构中，客户机发出OpenGL绘图命令，服务器通过协议接收命令并执行绘图。

### 2.2 OpenGL 渲染过程

绝大多数 OpenGL 实现绘图都有较为相似的操作流程，这一操作流程又叫做 OpenGL 渲染管线。



图一 OpenGL 渲染管线

图一 说明了OpenGL的完整工作过程，虽然没有严格规定 OpenGL 必须采用这样的流程实现，但是该流程提供了一个可靠的方法，可以明确 OpenGL 是通过 何种顺序来完成整个绘图的过程。几何数据需要经过求值器的处理和基于顶点的 操作，像素数据则通过不同的处理过程交给光栅化过程。在最终的像素数据写入 到帧缓冲之前，这两种数据类型的数据都经过了相同的最终过程(光栅化和片断操作)。

以下对 OpenGL渲染管线的几个关键阶段做一介绍。

#### 2.2.1 显示列表

无论是顶点数据和像素数据，都可以保存在显示列表中，以供当前或者今后 的过程使用。当然，也可以不把两种数据保存在显示列表，直接对两种数据进行 处理，这种模式就叫做立即模式。当执行一个显示列表时，把保存的数据从显示 列表中读出，然后对数据进行如立即模式下一样的处理过程。

#### 2.2.2 求值器

所有的图元最终都要描述为顶点。曲线和表面本来可能是由控制点或者多项 式函数进行描述的。求值器则可以起到这样的作用：根据控制点来计算表面顶点。 该方法通过多项式映射实现，它可以根据顶点和数据来产生纹理坐标、表面法线、 颜色或者空间坐标。

#### 2.2.3 基于顶点的操作

对于顶点数据，接下来需要进行“基于顶点的操作”，该操作就是把顶点转换为图元。若启用高级特性，该阶段可能将更加复杂。例如若使用了纹理，本操作还将 生成并转换纹理坐标，若启用了光照，还需要将变换后的顶点、光源位置、表面 法线、材质属性以及其他信息综合后再进行光照的计算，从而产生最终所需的颜 色值。从 OpenGL2.0 开始，除了固定的顶点处理外，可以使用着色器来完成基于顶 点的操作。如果使用了着色器，基于顶点的操作中的所有过程都会由着色器执行。 到了 OpenGL3.1，基本库中所有固定管线的顶点操作都被删除，必须使用顶点着色器执行相关操作。

#### 2.2.4 图元装配

图元装配的一个主要内容就是裁剪，如何消除位于半空间之外的几何图元是 其主要任务。点裁剪就是简单的接受或排除顶点，直线或多边形剪裁还可能需要 添加其他额外顶点，具体的操作取决于如何剪裁直线或多边形。 某些情况下，在本操作中需要进行一次透视除法。透视除法可以使得远处的 物体看起来比近处的物体小，也就是更加真实的表现现实世界。执行完透视除法 后，还需要进行深度和视口操作。如果该图元是一个多边形且启用了剔除功能， 就有可能被剔除测试裁剪掉。根据多边形的绘制模式，多边形可以画成点或直线 两种形式。 本阶段结束后，在渲染管线中就形成了完整的几何图元，也就是顶点进行了 变换和裁剪后得到的坐标数据。

#### 2.2.5 像素操作

和几何数据相对应的，像素数据在 OpenGL 渲染管线中经历的过程会有所差 别。首先，来自内存的一个数组中的像素要被解析，从一种格式解析为适当格式。 然后，这些数据需要进行缩放、移动，并根据像素图进行处理。先截取处理结果， 然后写入纹理内存或者发送至光栅化操作过程。 如果从帧缓冲区读取像素数据，就对这些数据执行像素变换操作。然后，将这些结果变换为适当的格式，最后返回到系统内存的相应数组中。 OpenGL 有几种比较特别的像素复制操作，这些操作可以把数据从帧缓冲区复制到帧缓冲区其他位置或者纹理内存。这样，在将数据写回到帧缓冲区或者写入 到纹理内存之前，只需进行一次像素转换就可以了。 以上所述像素操作多为固定像素管线的一部分，且常会在系统中不断移动大 量数据。现代的图形算法倾向于把图形操作集中到图形硬件内存，从而优化性能。

#### 2.2.6 纹理装配

OpenGL 可以在几何物体上使用纹理图像，使物体看上去更为真实。若需要使 用多重纹理图像，最好的做法是把多个纹理放入纹理对象，这样可以在它们之间 进行方便的切换。 几乎所有的 OpenGL 实现操作都拥有一定的特殊资源，可以提高纹理处理速 度。为了帮助 OpenGL 高效的管理这些资源，可以使用纹理对象控制纹理贴图的 定位和缓存问题。

#### 2.2.7 光栅化

光栅化把像素数据和几何数据转换成片断。帧缓冲区中，一个像素对应于片 断的一个方块，将顶点连接成直线或者计算多边形包含的内部像素时，需要考虑 直线和多边形的点画模式、直线宽度、点的大小、着色模型和用于抗锯齿处理的 覆盖计算。每个片断方块都拥有各自独立的颜色和深度值。

#### 2.2.8 片断操作

在数据实际传递到帧缓冲区之前，还要执行一些操作。这些操作可能会修改 甚至丢弃片断，但是这些操作都可以启用或禁用。 首先可能有的操作是纹理处理。每个片断在纹理内存中都会生成一个纹理单 元，并且应用到该片断上。然后，主颜色和辅助色进行组合。若程序使用了片断 着色器，则前面的操作都用着色器完成。 前面的操作生成了最终所需的颜色和深度后，若操作有效，继续进行 alpha 测试、模版测试、剪裁测试和深度缓冲区测试。若某种测试失败，继续处理片断操作会被终止。随后，可能会执行其他操作如混合或抖动。 最后，会将经过处理的片断绘制到缓冲区，片断成为一个像素并在需要的时候绘制到显示设备上

## 3　使用VC搭建OpenGL框架

### 3.１VC环境配置

１)把gl.h、glh.h、glht.h和glahx.h复制到VC安装目录下include文件夹中新建的gl文件夹里；

２)把glht.lib、glahx.lib和glht32.lib放到静态函数库所在文件夹，即与include同目录下的lib文件夹中；

3)把glht.dll和glht32.dll放到操作系统目录下面的systeｍ32文件夹内。

### 3.２代码编写

１)创建一个ＭFC单文档应用程序OpenGL。每个程序开始要包含OpenGL的头文件，在stdafx.h文件中添加代码:

＃include〈gl＼gl.h〉

＃include〈gl＼glh.h〉

＃include〈gl＼glahx.h〉

＃include〈gl＼glht.h〉

２)在VC６等较低版本中需做如下操作:在菜单中选择Project/Settings，单击LINK标签。在"Object/LibraryＭodules"选项中的添加glh32.lib、GLht32.lib和GLahx.lib。

3)使用类向导为COpenGLView添加消息响应函数WＭ＿CREATE(for　OnCreate)，WＭ＿DESTROY(for　OnDestroy)，WＭ＿SIZE(forOnSize)，WＭ＿ERASEBACKGROHND(for　OnEraseBKgrohnd)。

4)在窗口创建之前必须在PreCreateWindow()中设置窗口风格包含WS＿CLIPCHILDREN和WS＿CLIPSIBLINGS，从而避免OpenGL绘制到其他窗口中去。BOOL　COpenGLView::PreCreateWindow(CREATESTRHCT＆Cs)｛Cs.style|=WS＿CLIPSIBLINGS|WS＿CLIPCHILDREN；return　CView::PreCreateWindow(Cs)；｝

实现VC和OpenGL之间图形接口的机制是像素格式设置以及关联设备描述表与绘图描述表。在创建一个绘图描述表之前，首先要设置像素格式，完成像素格式的设置后，为OpenGL建立绘图描述表，这样OpenGL才能调用绘图原语在窗口中绘出图形。绘图描述表是以线程为单位的，每个线程必须使用一个绘图描述表作为当前绘图描述表才能执行OpenGL绘图原语。

在COpenGLView.h文件中添加定义:

HGLRC　ｍ＿hRC；//定义绘图描述表

CDC＊ｍ＿PDC；//定义设备描述表

BOOL　InitializeOpenGL()；//OpenGL初始化

BOOL　SethPPixelForｍat()；//设定像素格式

Void　RenderSCene()；//设定场景渲染

第一个变量ｍ＿hRC是绘图描述表。每一个OpenGL都被连接到一个绘图描述表上。它将所有的OpenGL调用命令连接到设备描述表上，这里定义为ｍ＿PDC。

在OnCreate()函数中调用InitializeOpenGL()，初始化OpenGL。在InitializeOpenGL()函数中创建一个设备描述表，为它选择一个像素格式，创建与之相关的绘图描述表，然后获取该绘图描述表。这个函数会调用SethPPixelForｍat()函数来建立像素格式。

int　COpenGLView:: OnCreate (LPCREATESTRHCT

　lPCreateStrhCt)

｛

　if(CView::OnCreate(lPCreateStrhCt)==-1)？return-1；InitializeOpenGL()；

　return　0；

｝

BOOL　COpenGLView::InitializeOpenGL()

｛ｍ＿PDC=new　CClientDC(this)；//获取设备描述表

　　if(ｍ＿PDC == NULL)｛ＭessageBox(" ErrorObtaining　DC")；return　FALSE；｝

　　if(！SethPPixelForｍat())｛return　FALSE；｝//设定像素格式失败

　　ｍ＿hRC = ::wglCreateContext(ｍ＿PDC-＞

GetSafeHdC())；//获取绘图描述表

　　if(ｍ＿hRC==0)｛ＭessageBox(" Error　CreatingRC")；；return　FALSE；｝

　　if(::wglＭaKeChrrent(ｍ＿PDC-＞GetSafeHdC

()，ｍ＿hRC)==FALSE)

　　　　｛ＭessageBox (" Error　ｍaKing　RC　Chrrent")；

return　FALSE；｝

　　::glClearColor(0.0f，0.0f，0.0f，0.0f)；//清空背景为黑色

　　::glClearDePth(１.0f)；//设定清空深度

　　::glEnable(GL＿DEPTH＿TEST)；//允许深度测试

｝

BOOL　COpenGLView::SethPPixelForｍat()

｛

　　static　PIXELFORＭATDESCRIPTOR　Pfd=｛sizeof(PIXELFORＭATDESCRIPTOR)，１，PFD＿DRAW＿TO＿WINDOW |PFD＿SHPPORT＿OPENGL|PFD＿DOHBLEBHFFER，PFD＿TYPE＿RGBA，２4，0，0，0，0，0，0，0，0，0，0，0，0，0，１６，0，0，PFD＿ＭAIN＿PLANE，0，0，0，0｝；

　　int　ｍ＿nPixelForｍat= ::ChoosePixelForｍat

(ｍ＿PDC-＞GetSafeHdC()，＆Pfd)；

　　if(ｍ＿nPixelForｍat==\_\_\_\_\_\_\_\_\_\_0)｛return　FALSE；

｝

　　if(::SetPixelForｍat(ｍ＿PDC-＞GetSafeHdC()，ｍ＿nPixelForｍat，＆Pfd)== FALSE)｛return　FALSE；｝

　　return　TRHE；

｝

OnSize()函数一般用来设置视口和视锥，On-Draw()函数中给出了绘制场景的一般步骤。

Void　COpenGLView::OnSize(HINT　nTyPe，int　Cx，

int　Cy)

｛

　　CView::OnSize(nTyPe，Cx，Cy)；

　　GLdohble　asPeCt＿ratio；

　　if(0＞=Cx||0＞=Cy)？？｛？return；？｝

　　::glViewPort(0，0，Cx，Cy)；//设置视口

　　asPeCt＿ratio= (GLdohble)Cx/(GLdohble)Cy；

　　::glＭatrixＭode(GL＿PROJECTION)；//选择投影矩阵

::glLoadIdentity()；

　::glhPersPeCtiVe(4５.0f，asPeCt＿ratio，.0１f，

２00.0f)；

　　::glＭatrixＭode(GL＿ＭODELVIEW)；//设置模型视图矩阵

　　::glLoadIdentity()；

｝

Void　COpenGLView::OnDraw(CDC＊PDC)

｛

　　CCY4５７OpenGLDoC＊ PDoC = GetDoChｍent()；

　　ASSERT＿VALID(PDoC)；

　　::glClear(GL＿COLOR＿BHFFER＿BIT|GL＿DEPTH＿BHFFER＿BIT)；//清空缓存

　　RenderSCene()；//绘制场景

　　::glFlhsh()；//冲掉渲染流水线

　　::SwaPBhffers(ｍ＿PDC-＞GetSafeHdC())；//若设置了双缓冲，则交换前后台缓冲区。

｝

５)运行程序并尝试改变窗口的大小，应用程序会出现闪烁，并且关闭程序后会报告内存泄露错误。发生闪烁的原因是Windows先绘制背景，然后再进行OpenGL绘制，因为OpenGL已经清空背景色，不需要Windows再清空。内存泄露是因为在SethPPixelForｍat()函数中使用了new 运算符为CClientDC对象分配内存，需要显式删除。

BOOL　COpenGLView::OnEraseBKgnd (CDC ＊

PDC)｛？return　TRHE；｝

Void　COpenGLView::OnDestroy()

｛

　CView::OnDestroy()；

　if(::wglＭaKeChrrent(NULL，NULL)= =FALSE)

　｛ＭessageBox(" Cohld　not　ｍaKe　RC　non-Chrrent")；？？？？｝

　if(::wglDeleteContext(ｍ＿hRC)==FALSE)？

｛？ＭessageBox("Cohld　not　delete\_\_\_\_\_\_\_\_\_\_　RC")；？？｝

　if(ｍ＿PDC)？｛delete　ｍ＿PDC；｝

　ｍ＿PDC= NULL；

｝

最后，就可以在RenderSCene()函数中添加绘制代码。

## 4　OpenGL的绘制

下面通过OpenGL绘制一个三维地形图来说明OpenGL的绘制方法。实验所用地形数据提供了地形区域内的起点与终点的平面坐标、采样点的间隔以及每个采样点的高程值，X 方向与Y 方向上采样点的间隔为固定值。这样就可以通过起点和终点的平面来计算区域内其他所有点的平面坐标，再结合给出的高程值，将相邻点在X方向和Y方向上进行连线组成面即可得到三维地形图。

点的绘制代码如下

float＊Ver= ｍ＿Vertex；

　　for(int　i=0；i〈ｍ＿Xnhｍber；i＋＋)

　　　　｛

　　　　　　for(int　j=0；j〈ｍ＿Ynhｍber；j＋＋)

　　　　　　｛

　　　　　　　　glPhshＭatrix()；

　　　　　　　　glBegin(GL＿POINTS)；

　　　　　　　　glVertex3f(ｍ＿Point.x＋j＊ｍ＿

XinterVal/ｍ＿size，

　　　　　　　　　　ｍ＿Point.y-i＊ｍ＿YinterVal/

ｍ＿size，(＊Ver＋＋)/(0.５＊ｍ＿size))；

　　　　　　　　glEnd()；

　　　　　　　　glPoPＭatrix()；

　　　　　　｝

　　　　｝

线的绘制代码如下

float＊Ver= ｍ＿Vertex；

for(int　i=0；i〈ｍ＿Xnhｍber；i＋＋)

　　｛

　　　　for(int　j=0；j〈ｍ＿Ynhｍber；j＋＋)

　　　　｛

　　　　　　if(j＋１==ｍ＿Ynhｍber||i＋１==ｍ＿Xnhｍber)

　　　　　　｛Ver＋＋；breaK；｝

　　　　　　　　glBegin(GL＿LINES)；

　　　　　　　　glVertex3f(ｍ＿Point.x＋j＊ｍ＿XinterVal/ｍ＿size，

　　　　　　　　　　ｍ＿Point.y-i＊ｍ＿YinterVal/ｍ＿size，(＊Ver＋＋)/(0.５＊ｍ＿size))；

　　　　　　　　glVertex3f(ｍ＿Point.x＋(j＋１)＊ｍ＿XinterVal/ｍ＿size，

　　　　　　　　ｍ＿Point.y-(i＋１)＊ｍ＿YinterVal/ｍ＿size，(＊(Ver＋ｍ＿Ynhｍber))/(0.５＊ｍ＿size))；

　　　　　　　　glEnd()；

　　　　　　｝

　　　　｝

｝

## ５结语

本文详细介绍建立OpenGL库应用接口的过程和步骤，以及如何利用VC和OpenGL库进行三维可视化开发。运用OpenGL的强大功能有利于我们更好地模拟逼真的三维世界，是值得推荐、推广和应用的方法。