计算机图形学

王培钰 16340220 16级电子政务

A. 什么是计算机图形学?

计算机图形学是研究如何在计算机中表示图形、以及利用计算机进行图形的计算、处理和显示的相关原理与算法的一门科学。

计算机图形学主要包括四部分：建模、渲染、动画和人机交互。

1. 建模：

三维模型的建模师计算机图形学的基础。表达一个几何物体可以用数学上的样条函数或隐式函数来表达，也可以是用光滑曲面上的采样点及其连接关系所表达的三角网格来表达。

2. 渲染：

渲染是计算机图形学的核心任务，个人理解是使得图形更逼近真实，加上颜色，光照等视觉要素。常见的渲染模型有局部光照模型、光线跟踪算法、辐射度等。

3. 动画

动画是采用连续播放静止图像的方法产生物体运动的效果。计算机动画技术借助编程或动画制作软件生成一系列的景物画面，是各类动态仿真应用的核心技术，极大的提高了虚拟现实系统的沉浸感。

4. 人机交互

人机交互是指人与计算机之间以一定的交互方式或交互界面来完成确定的人与计算机之间的信息交换过程。

B. 什么是openGL?

openGL是一个应用程序编程接口，包含了一系列可操作图形、图像的函数。实际OpenGL的开发者通常是显卡的生产商。

1. **核心模式与立即渲染模式**：

早起的OpenGL使用立即渲染模式(Immediate mode，即固定渲染管线)：OpenGL的大多数功能都被库隐藏起来，开发者很少能控制OpenGL如何进行计算的自由，渲染模式容易使用和理解，但是效率太低。

当使用OpenGL的核心模式时，OpenGL迫使我们使用现代的函数，现代函数具有更高的灵活性和效率性，让人更容易理解OpenGL是如何运作的，深入理解图形编程。

2. **扩展**

当一个显卡公司提出一个新特性或者渲染上的大优化，通常会以扩展的方式在驱动中实现。如果一个程序在支持这个扩展的显卡上运行，开发者可以使用这个扩展提供的一些更先进更有效的图形功能，而不需要等待一个新的OpenGL规范面世，只需要简单检查一下显卡是否支持此扩展。

3. **状态机**

   一系列的变量描述OpenGL此刻应当如何运行，OpenGL的状态通常被称为OpenGL的上下文，更改OpenGL状态：设置选项，操作缓冲。最后，使用当前OpenGL上下文来渲染。

使用OpenGL的时候，我们会遇到一些状态设置函数，改变上下文；状态使用函数，根据当前状态执行一些操作。

4. **对象**

在OpenGL中一个对象是指一些选项的集合，它代表OpenGL状态的一个子集。比如，我们可以用一个对象来代表绘图窗口的设置，之后我们就可以设置它的大小、支持的颜色位数等等。

C. 什么是OpenGL ES?

OpenGL ES (OpenGL for Embedded Systems) 是 OpenGL三维图形 API 的子集，针对手机、PDA和游戏主机等嵌入式设备而设计。该API由Khronos集团定义推广，Khronos是一个图形软硬件行业协会，该协会主要关注图形和多媒体方面的开放标准。

OpenGL ES 是从 OpenGL 裁剪的定制而来的，去除了glBegin/glEnd，四边形（GL_QUADS）、多边形(GL_POLYGONS)等复杂图元等许多非绝对必要的特性。

1. **顶点**

在OpenGL ES中，支持三种类型的绘制：点、直线以及三角形；由这三种图形组成其他所有图形，比如我们看到的圆滑的球体也是由一个个三角形组成的，三角形越多看上去越圆滑。在绘制图形时我们需要提供相应的顶点位置，然后指定绘制方式去绘制这些顶点，以此呈现出我们想要的图形。

2. **坐标系**

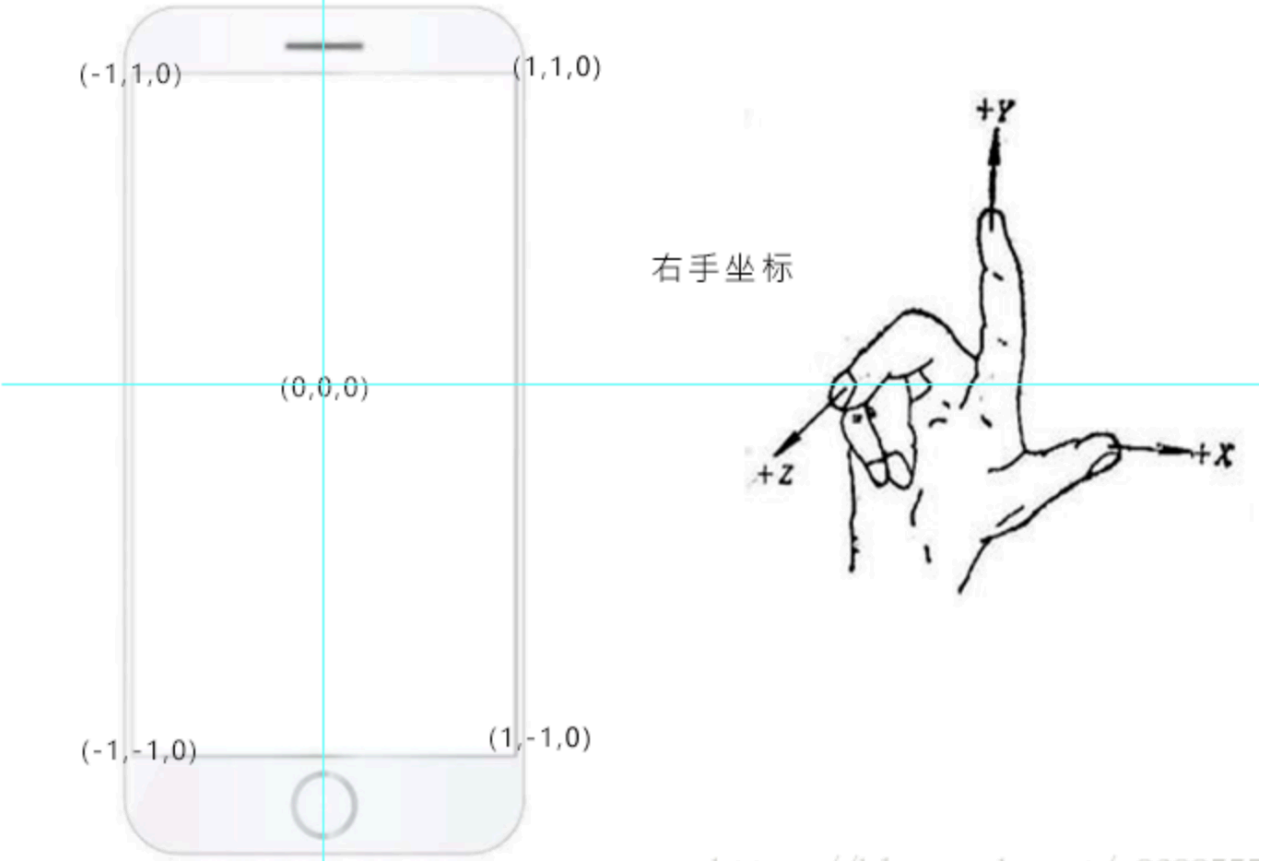
opengl是一个3d的世界，由 x,y,z 坐标组成顶点坐标。坐标表示如下：

标准化设备坐标(Normalized Device Coordinates, NDC)

从原点到屏幕边缘默认长度为1,向右为X正轴方向，

向左为X负轴方向，向上为Y轴正轴方向，向下为Y轴负轴方向，

屏幕面垂直向上为Z轴正轴方向，垂直向下为Z轴负轴方向



3. **着色器**

OpenGL编程语言是GLSL，就是我们说的着色器语言。语法上比较像c/c++。但是在这一基础上加上了许多适合图形处理的一些东西,比如一些针对向量和矩阵操作等。我们要图形渲染，就一定需要**顶点着色器**和一个简单的**片元着色器**。

- 顶点着色器：

主要作用是接受顶点数据，也是说通常是用来接受定义的顶点坐标，或者是一些关于矩阵变换，纹理生成和坐标变换等...。总的来说就是处理顶点数据。

- 片段着色器：

最主要是确定渲染的颜色。就像画家调色板。很多滤镜、美颜、图片处理。或者说抖音的一些特效，都可以用片元着色器编写。

D. 什么是Web GL?

WebGL是一种3D绘图协议，这种绘图技术标准允许把JavaScript和OpenGL ES 2.0结合在一起，通过增加OpenGL ES 2.0的一个JavaScript绑定，WebGL可以为HTML5 Canvas提供硬件3D加速渲染，这样Web开发人员就可以借助系统显卡来在浏览器里更流畅地展示3D场景和模型了，还能创建复杂的导航和数据视觉化。显然，WebGL技术标准免去了开发网页专用渲染插件的麻烦，可被用于创建具有复杂3D结构的网站页面，甚至可以用来设计3D网页游戏等等。

WebGL 的出现使得在浏览器上面实时显示 3D 图像成为，WebGL 本质上是基于光栅化的 API ,而不是基于 3D 的 API。

WebGL 只关注两个方面，即投影矩阵的坐标和投影矩阵的颜色。使用 WebGL 程序的任务就是实现具有投影矩阵坐标和颜色的 WebGL 对象即可。可以使用“着色器”来完成上述任务。顶点着色器可以提供投影矩阵的坐标，片段着色器可以提供投影矩阵的颜色。无论要实现的图形尺寸有多大，其投影矩阵的坐标的范围始终是从 -1 到 1 。

E. 什么是Vulkan?

Vulkan是Khronos组织制定的“下一代”开放的图形显示API。是与DirectX12能够匹敌的GPU API标准。它是基于AMD的Mantle API演化而来，眼下Vulkan 1.0标准已经完毕并正式公布。上一代的OpenGL|ES并不会被遗弃。还会继续发展，非常有可能OpenGL|ES变为Vulkan的简化API。

Vulkan的优势

与OpenGL|ES相比Vulkan的优势：

l 更简单的显示驱动层

Vulkan提供了能直接控制和訪问底层GPU的显示驱动抽象层。 显示驱动仅仅是对硬件薄薄的封装，这样能够显著提升操作GPU硬件的效率和性能。之前OpenGL的驱动层对开发人员隐藏的非常多细节，如今都暴露出来。Vulkan甚至不包括执行期的错误检查层。驱动层干的事情少了，隐藏的bug也就少了。

l 支持多线程

Vulkan不再使用OpenGL的状态机设计，内部也不保存全局状态变量。显示资源全然由应用层负责管理。包括内存管理、线程管理、多线程绘制命令产生、渲染队列提交等。

应用程序能够充分利用CPU的多核多线程的计算资源，降低CPU等待，降低延迟。 带来的问题是。线程间的同步问题也由应用程序负责，从而对开发人员的要求也更高。

l 预编译Shaders

驱动层不提供前端shader编译器。仅仅支持标准可移植中间表示二进制代码（SPIR-V）。

即提高了执行Shaders的效率又添加了将来着色语言的灵活性。

所以眼下的GLSL/HLSL能够直接通过工具转换为SPIR-V。在Vulkan中使用。这样就能够使用离线的shader编译。

另外。SPIR-V还支持OpenCL!

l 跨平台

支持桌面、移动设备、游戏主机、嵌入式……仅仅要须要显示的地方，貌似都能支持。

这也是Vulkan与DirectX12相比的优势。

F. 什么是DirectX?

DirectX，(Direct eXtension，简称DX) 是由微软公司创建的多媒体编程接口。由C++实现，遵循COM。被广泛使用于Microsoft Windows、Microsoft XBOX、Microsoft XBOX 360和Microsoft XBOX ONE电子游戏开发，并且只能支持这些平台。最新版本为DirectX 13，创建在最新的Windows10。

组成：

DirectX是由很多API组成的，按照性质分类，可以分为四大部分，显示部分、声音部分、输入部分和网络部分。

显示部分

显示部分担任图形处理的关键，分为DirectDraw（DDraw）和Direct3D（D3D），前者主要负责2D图像加速。它包括很多方面：我们播放mpg、DVD电影、看图、玩小游戏等等都是用的DDraw，你可以把它理解成所有划线的部分都是用的DDraw。后者则主要负责3D效果的显示，比如CS中的场景和人物、FIFA中的人物等等，都是使用了DirectX的Direct3D。

声音部分

声音部分中最主要的API是DirectSound，除了播放声音和处理混音之外，还加强了3d音效，并提供了录音功能。我们前面所举的声卡兼容的例子，就是利用了DirectSound来解决的。

输入部分

输入部分DirectInput可以支持很多的游戏输入设备，它能够让这些设备充分发挥最佳状态和全部功能。除了键盘和鼠标之外还可以连接手柄、摇杆、模拟器等。

网络部分

网络部分DirectPlay主要就是为了具有网络功能游戏而开发的，提供了多种连接方式，TCP/IP，IPX，Modem，串口等等，让玩家可以用各种连网方式来进行对战，此外也提供网络对话功能及保密措施。

G. gl.h, glu.h, glew.h

<GL/gl.h>：OpenGL所使用的函数和常量声明。

<GL/glu.h>：GLU（OpenGL实用库）所使用的函数和常量声明。GLU库属于OpenGL标准的一部分。它包含有43个函数，函数名的前缀为glu。Glu 为了减轻繁重的编程工作，封装了OpenGL函数，Glu函数通过调用核心库的函数，为开发者提供相对简单的用法，实现一些较为复杂的操作。

<GL/glew.h>：GLEW是一个跨平台的C++扩展库，基于OpenGL图形接口。使用OpenGL的朋友都知道，window目前只支持OpenGL1.1的涵数，但 OpenGL现在都发展到2.0以上了，要使用这些OpenGL的高级特性，就必须下载最新的扩展，另外，不同的显卡公司，也会发布一些只有自家显卡才支 持的扩展函数，你要想用这数涵数，不得不去寻找最新的glext.h,有了GLEW扩展库，你就再也不用为找不到函数的接口而烦恼，因为GLEW能自动识 别你的平台所支持的全部OpenGL高级扩展涵数。也就是说，只

要包含一个glew.h头文件，你就能使用gl,glu,glext,wgl,glx的全 部函数。GLEW支持目前流行的各种操作系统（including Windows, Linux, Mac OS X, FreeBSD, Irix, and Solaris）。

H. freeglut和glew的作用和区别

在程序中使用OpenGL之前，你需要对他进行初始化，但是由于OpenGL是跨平台的，所以也没有一个标准的方式。

进行初始化。OpenGL初始化分为两个阶段：

- 第一个阶段，你需要创建一个OpenGL上下文环境，这个上下文环境存储了所有与OpenGL相关的状态（OpenGL是一个状态机），上下文位于操作 系统中某个进程中，一个进程可以创建多个上下文，每一个上下文都可以描绘一个不同的可视界面，就像应用程序中的窗口；简单来理解就是为了创建 一个窗口。
- 第二个阶段，你需要定位所有需要在OpenGL中使用的函数。

freeglut和**glew**就是用来解决这两个问题的。

freeglut的作用

因为OpenGL是跨平台的，这给开发者带来了方便，但是在创建上下文时这也是一个麻烦的地方，因为不同平台的窗口系统和API都是不一样的，freeglut就是一个openGL的工具库，封装了各个平台初始化窗口的过程，开发者只需要调用这个库中的初始化函数即可创建一个上下文。

glew的作用

因为OpenGL只是一个标准/规范，具体的实现是由驱动开发商针对特定显卡实现的。由于OpenGL驱动版本众多，它大多数函数的位置都无法在编译时确定下来，需要在运行时查询。任务就落在了开发者身上，开发者需要在运行时获取函数地址并将其保存在一个函数指针中供以后使用。

I. VR技术介绍

- 《VR凉凉？逛完展你就知道了》

VR是一项综合集成的虚拟现实技术，涉及CG，人机交互技术，传感技术，人工智能等领域，它利用计算机生成逼真的三维视、听、嗅觉等感觉，使人作为参与者通过适当装置，自然地对虚拟世界进行体验和交互作用。“虚拟”采用的就是三维的计算机图形学技术。要利用CG的处理产生一种可视化界面技术，可以有效的建立虚拟环境。主要集中在两方面，一是虚拟环境可以精确表示物体状态模型，二是环境的可视化渲染。

文章主要在商展区域介绍VR技术，谷歌、微软、英特尔、英伟达、AMD、Unity、惠普、Star VR等大公司都展示了自己最全最新的VR产品。该会议中公司带来的新产品所用技术重点在于几个方面：GPU等硬件和软件算法进一步提升计算机图形图像能力、3D打印和运动追踪更为成熟、VR硬件软件公司更倾向于商业方案的整合和优化。展览精彩的几个模块主要有：**惠普扩展VR软硬件商业合作**、**Star VR推集成眼球追踪的新头显**、**英伟达发布全新GPU架构**、**Vicon推全新运动追踪系统**、**斯坦福大学研发可校正视力的“AR眼镜”**等。

J. openGL环境配置

- 操作系统：mac OSX
- 配置glew和glfw
- IDE：XCode

测试代码：

```
#include <iostream>
#include <GL/glew.h>
#include <GLFW/glfw3.h>

void Render(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    {
        glColor3f(1.0,0.0,0.0);
        glVertex2f(0, .5);
        glColor3f(0.0,1.0,0.0);
        glVertex2f(-.5,-.5);
        glColor3f(0.0, 0.0, 1.0);
        glVertex2f(.5, -.5);
    }
    glEnd();
}

int main(int argc, const char * argv[]) {
    GLFWwindow* win;
    if(!glfwInit()){
        return -1;
    }
    win = glfwCreateWindow(640, 480, "OpenGL Base Project", NULL, NULL);
    if(!win)
    {
        glfwTerminate();
        exit(EXIT_FAILURE);
    }
    if(!glewInit())
    {
        return -1;
    }
    glfwMakeContextCurrent(win);
    while(!glfwWindowShouldClose(win)){
        Render();
        glfwSwapBuffers(win);
        glfwPollEvents();
    }
```

```
    glfwTerminate();
    exit(EXIT_SUCCESS);
    return 0;
}
```

配置成功后的运行结果：

```
#include <iostream>
#include <GL/glew.h>
#include <GLFW/glfw3.h>

void Render(void)
{
    glClearColor(0.
    glClear(GL_COLO
    glBegin(GL_TRIA
    {
        glColor3f(1
        glVertex2f(
        glColor3f(0
        glVertex2f(
        glColor3f(0
        glVertex2f(
    }
    glEnd();
}

int main(int argc,
    GLFWwindow* win
    if(!glfwInit())
        return -1;
    }
    win = glfwCreat
    if(!win)
    {
        glfwTermina
        exit(EXIT_F
    }
    if(!glewInit())
    {
        return -1;
    }
```