2018计算机图形学 二维纹理过滤实验说明

Shi Ruofei
Nanjing University

实验所涉及的具体理论、公示、模型等在罗浩然学长的《二维纹理过滤_2D_Texture_Filtering》中已有完整的阐述及推导过程，本说明除了一些争议点外不会重复对理论的讲解
关于本次实验所需使用的环境、外部依赖、实现要求等均以此说明文档为准
意思其实就是可以无视学长的pdf最后关于实验要求的几页啦 XD

框架代码 && 外部依赖说明

Build && Configure

本次实验的框架代码基于C++编写，OpenGL版本为4.1
所需要的外部依赖包括：

1. glad -- a GL loader
2. glfw 3.2.1 -- 基于OpenGL创建GUI并提供一些交互接口的库
3. glm 0.9.9.1 -- OpenGL Mathematics，一个数学库，包括很多向量、矩阵运算接口

如果你是Windows系统

你可以选择使用Visual Studio(MSVC x86 or x64)、Mingw(32bit)或Mingw-w64(64bit)作为编译器，你需要3.1版本以上的CMake来make框架代码中提供的CMakeList.txt
对于CLion + Mingw or Mingw-w64的用户可以直接build框架代码并运行
对于Visual Studio用户需要使用CMake生成对应VS版本的解决方案文件(.sln)，再用VS打开项目并正常build or run
在框架代码的/ext目录下有纯头文件组成的glad和glm相关内容和预编译后的glfw动态链接库，CMakeList内已有指令进行include和link
因此你可以在原生环境下直接build成功而不需要去下载安装上述依赖(方便我们统一环境)
我成功build的版本包括：VS2017，Mingw(GCC 6.3)，Mingw-w64(GCC 8.1)
应该可以向下兼容（?

如果你是MacOS系统

很不幸目前的CMakeList.txt对你来说失效了
你可以参考某博客进行上述外部依赖的配置，请注意尽量保证版本一致
glad在线服务中你需要选择OpenGL 4.1版本并在Profile中选择核心模式(Core)
glad与glm均只需要头文件/源文件就可以运行
配置成功后可以将框架代码的剩下部分移入

如果你是Linux系统

我认为CMakeList和编译第三方库之类的东西对你来说都不是问题，你可以自行修改/重新编写

框架代码介绍

OpenGL是一套非常繁杂的API，从零开始搭建一个可以看到三维世界的视窗所需的步骤过于冗杂，而且它的逻辑是基于上下文的(context)，如果你习惯了顺序编程/OOP，会觉得OpenGL很反人类
因此提供了框架代码屏蔽这些流程上的细节，大家可以专注于shader的编写(本次实验中你只需要改动两个glsl文件便可实现所有要求)
但你如果对OpenGL管线和渲染流程完全陌生，在实验中遇到问题可能会不知道为什么(我也不知道什么时候会出现问题，也可能是在之后的实验里)

/resource

包含纹理图片subject.jpg

/shader

包含glsl(The OpenGL Shading Language)文件，分别为顶点着色器vertex.glsl和片段着色器fragment.glsl，这两个着色器也是本次实验中你需要实现的部分

/src

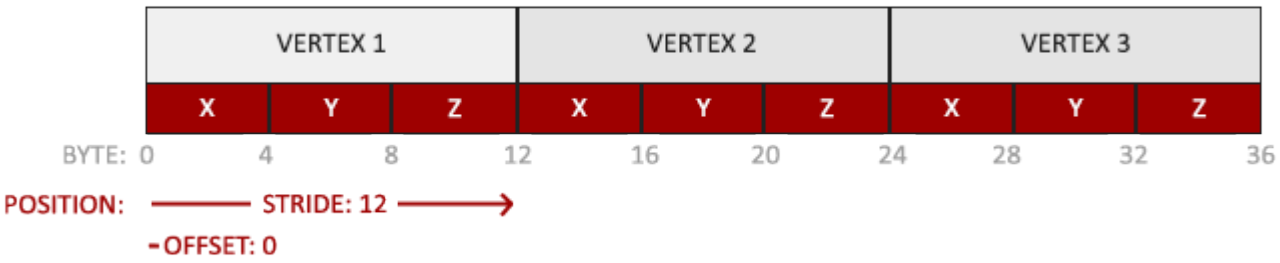
TextureFilteringScene.h/.cpp控制本次实验二维纹理过滤场景的初始化、渲染、交互等功能，有兴趣的同学可以看看其中的操作流程
main.cpp控制glfw窗体的创建or刷新等

/tools

SpriteRenderer.h/.cpp代表一个三维空间内的矩形，矩形上贴了一张纹理，构成一个Sprite，你可以在SpriteRenderer.cpp的*init*和*drawSprite*函数里找到这个矩形的顶点是如何定义的
Texture2D.h/.cpp用于加载纹理至内存 Shader.h/.cpp用于编译、链接shader文件 ResourceManager.h/.cpp用于加载、存储资源文件(shader文件和纹理文件)

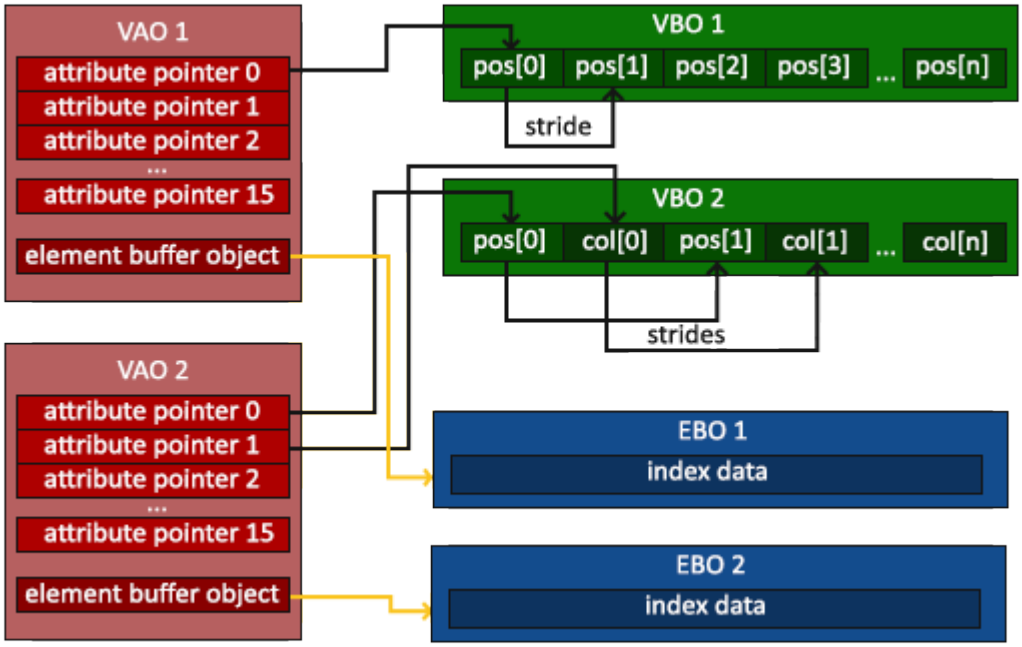
VAO && VBO && IBO

在课程实验中我们会用到的3维物体模型都是由三角形片组成的，三角形由其三个顶点表示
顶点(vertex)是什么?



除了位置vertex还可以包括颜色、纹理坐标、法线向量、切线向量.....

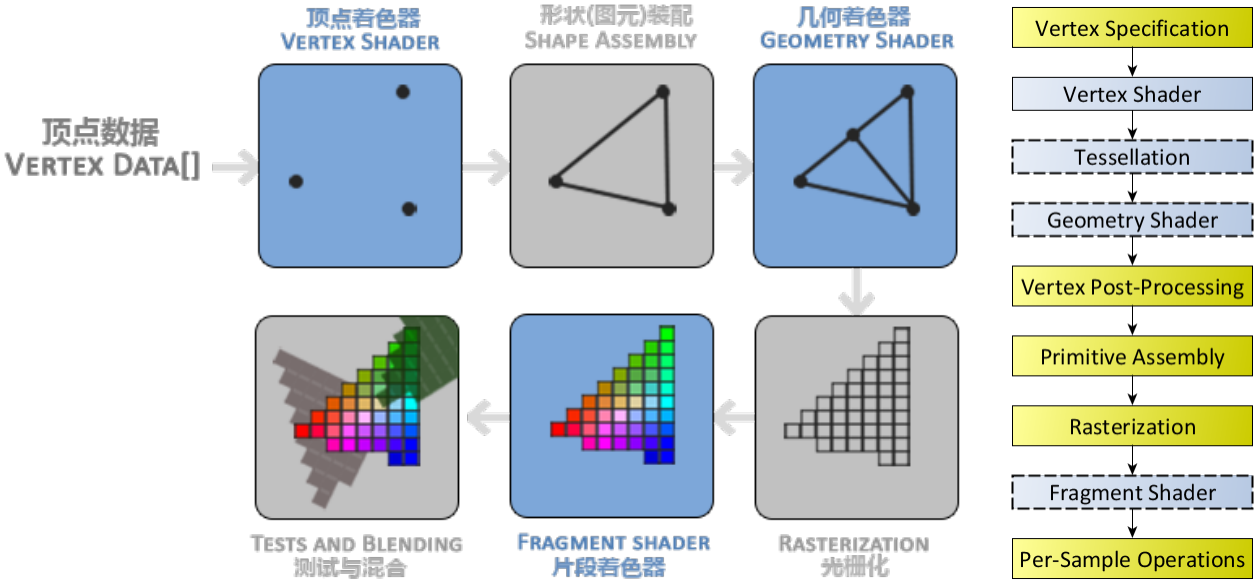
不使用重复的顶点以节约内存：IBO(Index Buffer Object or Element Buffer Object)




```
GLfloat vertices[] = {
        //  ---- 位置 ----     - 纹理坐标 -
        1.0f, 1.0f, 0.0f,     1.0f, 1.0f,   // 右上
        1.0f, -1.0f, 0.0f,    1.0f, 0.0f,   // 右下
        -1.0f, -1.0f, 0.0f,   0.0f, 0.0f,   // 左下
        -1.0f, 1.0f, 0.0f,    0.0f, 1.0f    // 左上
};

GLuint indices[] = {
    0, 2, 3,
    0, 1, 2
};

// some generate and bind codes

glEnableVertexAttribArray(0);
/*
* parameters:
* 0 -- layout position
* 3 -- number of attributes
* GL_FLOAT -- type of attributes
* GL_FALSE -- is normalized
* 5 * sizeof(GLfloat) -- attributes' data length
* nullptr(0) -- offset
*/
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(GLfloat), (GLvoid*)nullptr);
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(GLfloat), (GLvoid*)(3 *
```

```
sizeof(GLfloat)));
```

OpenGL Pipeline



顶点着色器

将顶点的3D位置转化为另一个3D位置(将顶点由模型空间转化至剪裁空间)
输出信息至后续管线，在片段着色器中可以再次使用——本次实验内是对应的纹理坐标

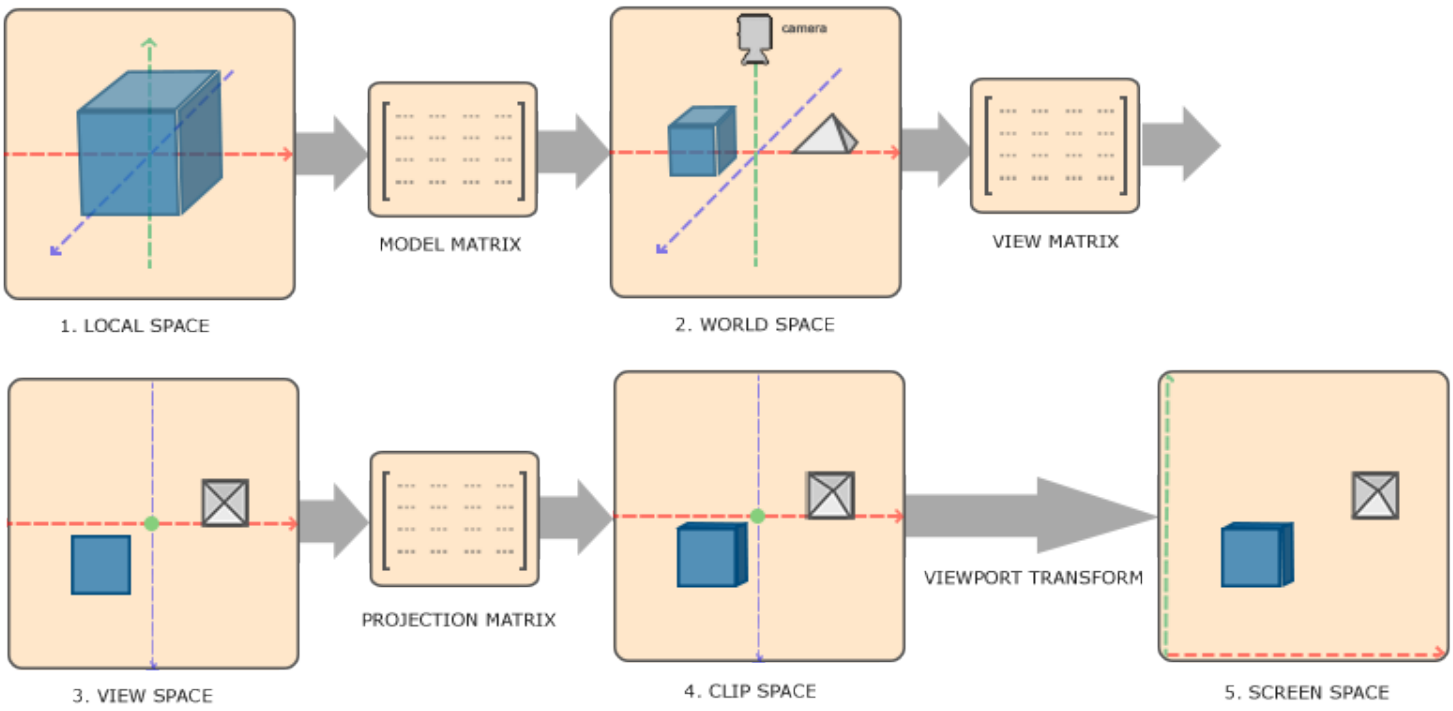
片段着色器

*OpenGL*中的一个片段是*OpenGL*渲染一个像素所需的所有数据
片段内的数据如何得到/计算——由顶点数据进行插值
输出：这个像素的颜色rgba

使用OpenGL渲染物体至少需要一个顶点着色器和一个片段着色器

以上渲染管线的描述是一个简略化的版本，主要目的是希望大家了解一下顶点/片段着色器是在干什么，具体的细节和更多流程有兴趣的同学可以看看OpenGL Insight内的图表

变换与坐标系统



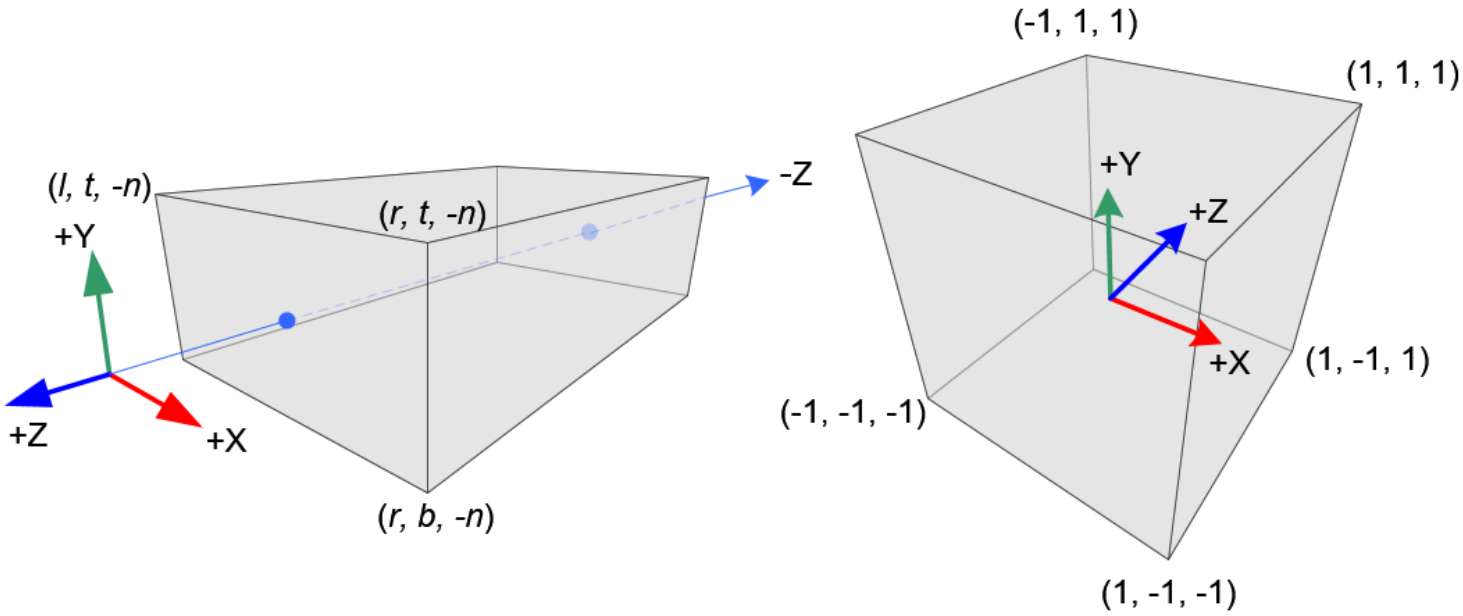
模型空间

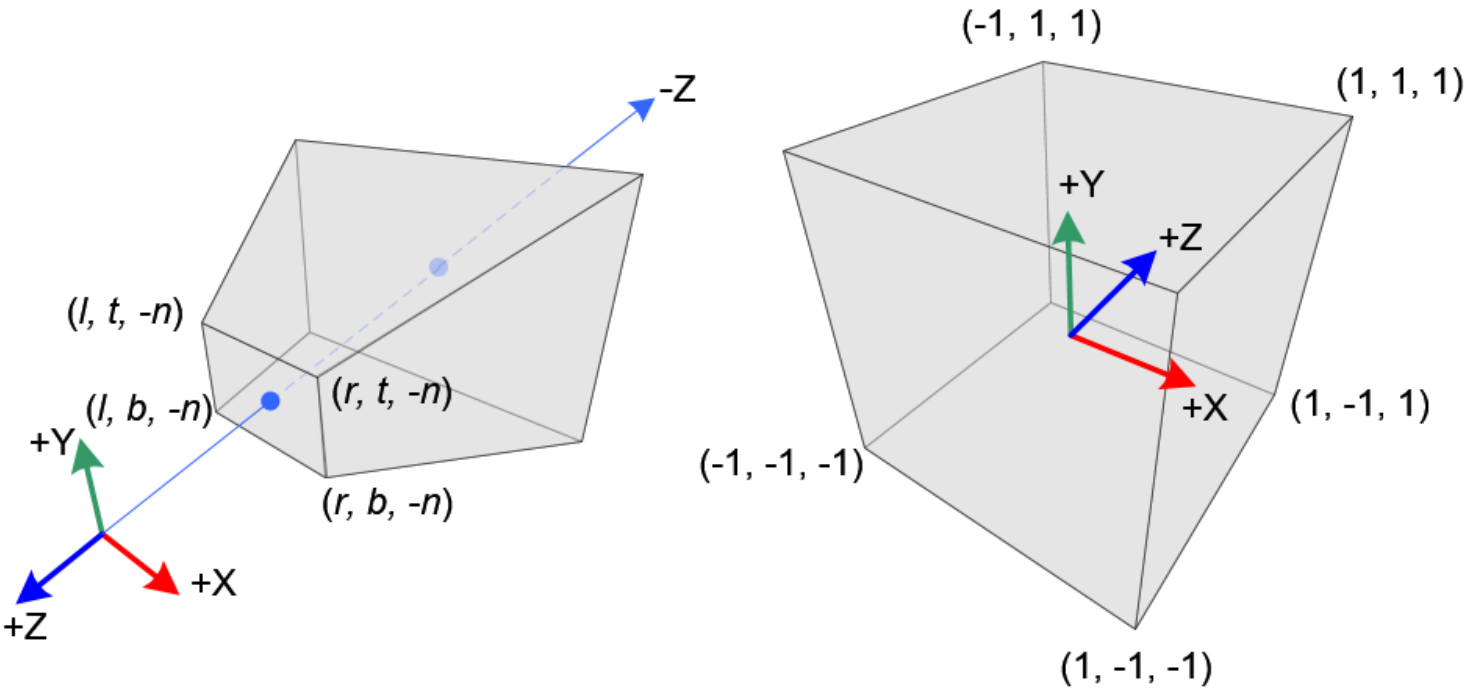
世界空间

观察空间/相机空间

剪裁空间

为什么会近大远小?
正交投影or透视投影





剪裁+标准化 -> 光栅化 -> 屏幕映射

坐标系统变换的数学表达有兴趣的同学可以参考这个链接

关于《二维纹理过滤》中的理论与数学模型

关于计算MIPMAP的层级

我个人的观点
使用坐标函数P在x或y方向的变化快慢来选取MIPMAP层级是正确的做法
但对P求偏导的概念在这里不太严谨
GLSL的内置函数dFdx代表的是在片段里x每偏移1个**像素**P的变化向量（同理dFdy），而P的取值在0-1之间，因此这里的结果必然小于等于1，$min(M, M + log_2(max(|dF\,dx|, |dF\,dy|)))$这个公式才有意义
所以在实现时直接用$dF\,dx(texCoords)$和$dF\,dy(texCoords)$代入这个公式便能得到正确的结果

实验要求

使用GLSL语言完成vertex.glsl和fragment.glsl两个着色器的实现，你可以在代码的注释里看到实现它们所需的必要步骤
你总共需要实现8种过滤方式：最近邻、双线性、MIPMAP双线性、三线性、2x各向异性、4x各向异性、8x各向异性、16x各向异性，键盘1-8可以在它们之间切换（交互处理已经包含在框架中，详见fragment.glsl的注释）
最近邻+双线性： 30%
MIPMAP双线性+三线性： 40%
各向异性： 30%

你可以提交整个项目，当然如果你所使用依赖的版本和框架代码均和前文的说明一致并未加改动，你也可

以只提交两个glsl文件
关于过滤函数的实现方法，要有足够的注释或干脆写个说明文档(让我能快速看懂就好)

关于OpenGL渲染流程的API调用原理与使用方法可以参考：
《OpenGL Programming Guide》
learnopengl的Getting Start章节
正如最开始所说，框架帮你屏蔽了这些，它们也无法指导你实现本次实验的数学模型，但也许可以帮你解决遇到的某些问题*XD*

这些关于GLSL语言的书or文档会为你实现本课程的相关实验提供很多帮助：
《OpenGL Shading Language 3rd edition》
opengl4.1的文档
前者比较详细，后者比较简洁，你可以在3-6章查阅GLSL的语法，**在7-8章查阅需要的内置函数or变量**

你也许会用到这个链接