研究生课程考试成绩单
（试卷封面）

院　　系	计算机科学与工程学院	专业	计算机科学与技术		
学生姓名	卓旭	学号	212138		
课程名称	信息可视化技术				
授课时间	2021 年 9 月至 2021 年 12 月	周学时	3	学分	3.0
简要评语					
考核论题	Project 1：球面与正方体表面的光照和观察 Project 2：体绘制的光线跟踪（Ray Casting）实现与课程综述报告				
总评成绩（含平时成绩）					
备注					

任课教师签名：_____

日期：

注：1. 以论文或大作业为考核方式的课程必须填此表，综合考试可不填。“简要评语”栏缺填无效。

2. 任课教师填写后与试卷一起送院系研究生秘书处。

3. 学位课总评成绩以百分制计分。

《信息可视化技术》课程实验报告与课程综述报告

212138 卓旭

前言

本课程的实验包括两个 Project：①显示球面和正方体表面，支持光照和观察（视见变换）等特性；②体绘制的光线跟踪（Ray Casting）算法实现。另外，第二个实验的报告内容同时作为课程的学习报告。

我的实验使用 C++和 OpenGL 实现。项目需要安装的第三方依赖库有：

- glfw3 [1]：对 OpenGL 的操作系统特定操作进行封装，便于与 GL 上下文交互；
- glad [2]：对 OpenGL 的显卡驱动特定操作进行封装，便于调用 GL 系列函数；
- glm [3]：提供了向量、矩阵类和相关运算，以及图形学常用矩阵的生成；
- rapidjson [4]：增强 Project 2 的可配置性，借助该库解析 JSON 配置文件。

以上依赖库均为现代 OpenGL 领域常用的库。对于核心算法的实现，均为手工编码实现，而非直接调用库函数完成。

C++的依赖安装配置较为复杂，但微软提供的包管理工具 vcpkg [5]给我们提供了极大的便利。在安装好 vcpkg 后，执行：`vcpkg install glfw3 glad glm rapidjson` 命令，即可安装所有依赖。

本课程实验的有关代码后续均开源到 github.com/z0gSh1u/seu-viz **，可至该网址取得。**

参考资料

[1] GLFW. GLFW3 [CP/OL]. 2021. https://www.glfw.org .

[2] Glad. GLAD [CP/OL]. 2021. https://glad.dav1d.de .

[3] g-truc. GLM [CP/OL]. 2021. https://glm.g-truc.net .

[4] Tencent. rapidjson [CP/OL]. 2021. http://rapidjson.org .

[5] Microsoft. vcpkg [CP/OL]. 2021. https://github.com/microsoft/vcpkg .

预备：自己设计的 OpenGL 辅助框架 ReNow 的介绍

在正式介绍两个 Project 的具体细节前，先介绍一个自己设计的 OpenGL 辅助框架：ReNow。该框架的雏形在本人本科的《计算机图形学》课程（东南大学，学号 09017227，杨武老师）实验中成型，代码可在 github.com/z0gSh1u/typed-webgl 处获得。其原先使用 TypeScript 语言开发，工作在 WebGL 下。借助本课程实验的契机，我开发了一个更完善的、工作在 C++和 OpenGL 下的版本，放置在 framework 目录下。

ReNow 的核心思想是使用 ReNowHelper 类接管、包装 OpenGL 的一些操作。包括 Program 的创建、Shader 的创建、Program 的切换、2D 材质的创建、VAO 与 VBO 的创建等。**ReNow 最关键的功能是提供** prepareAttributes **和** prepareUniforms **两个函数**，以低代码量与着色器进行交互，将数据传送到顶点着色器或片元着色器的 Attribute 或 Uniform 型变量上。使用例子如下：

```
float vtBack[nPoints * 2] = {0, 0, 1, 0, 1, 1, 0, 1}; // texture coords
GL_OBJECT_ID vtBackBuf = helper.createVBO();
helper.prepareAttributes(vector<APrepInfo>{
    {vtBackBuf, vtBack, nPoints * 2, "aTexCoord", 2, GL_FLOAT},
});
// ----------------------------------------------------------------------
mat4 perspectiveMat = glm::perspective(radians(fovy), aspect, near, far);
helper.prepareUniforms(vector<UPrepInfo>{
    {"uPerspectiveMatrix", perspectiveMat, "Matrix4fv"},
    {"uColor", normalizeRGBColor(RGBColor(29, 156, 215)), "4fv"},
});
```

上半部分表示：借助 vtBackBuf 这个 Buffer，将 vtBack 数组内容，共 nPoints * 2 个，传递到着色器名为 aTexCoord 的 Attribute 上，每 2 个数据为一组，类型为 GL_FLOAT。下半部分表示，将着色器程序中名为 uPerspectiveMatrix 的 uniform 的值，赋为 perspectiveMat 这个矩阵，内部使用的方法为 glUniformMatrix4fv；uColor 同理。通过这样的封装，减少了与着色器交互的代码量，代码更清晰。

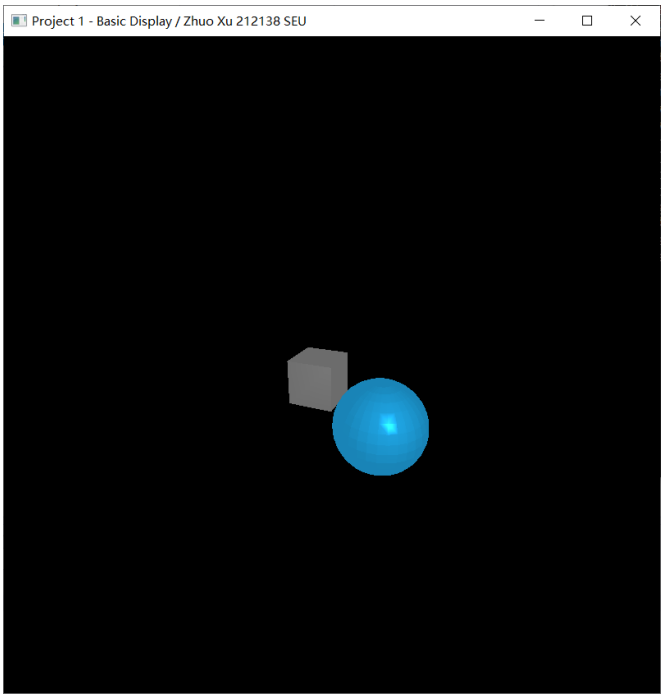
ReNow 框架还有一些辅助部分：相机（Camera.hpp）用于管理视点和 lookAt 矩阵，实现空间漫游；OBJ 文件解析器（OBJProcessor.hpp）用于从.obj 模型文件读取顶点、面等数据；Phong 光照模型（PhongLightModel.hpp）用于对光照相关变量和运算进行封装；以及工具函数库（Utils.hpp）。具体细节将在使用到时进一步介绍。

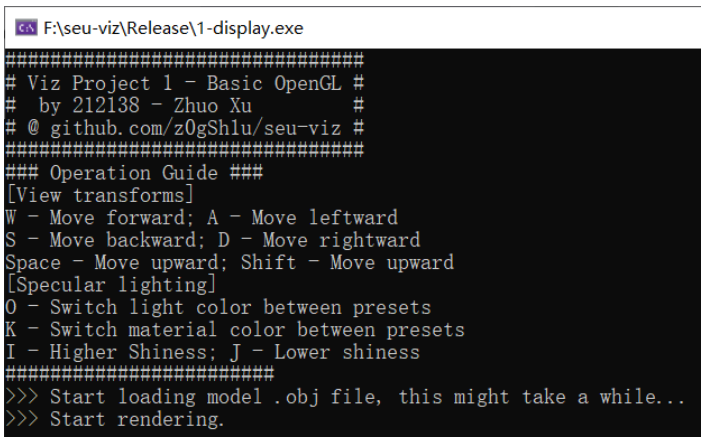
Project 1：显示球面和正方体表面，支持光照和观察

本实验内容为：

- 熟悉 OpenGL 编程和使用
- 显示一个球面和正方体表面，关注：
  - 数据结构
  - 颜色（RGBA）
  - 视见变换（改变视点观察正方体显示）
  - 光照模型（改变镜面反射参数观察球面显示效果）

本实验相关代码在 1-display 目录下。整体运行效果如下所示。上图为球面和正方体表面的显示效果，可见颜色和光照的作用效果。下图为操作说明，可通过 WASD-Space-Shift 键进行三维空间漫游，通过鼠标的移动改变视角；通过 OKIJ 键调整光照 Specular（镜面）分量的颜色和 Shiness 值。





首先介绍物体的数据结构。在本实验中，所有的面都是三角面片。为了进行基础绘制，我们需要顶点、面两个信息；为了添加光照，我们需要各个面的外法向量信息。

对于正方体，上述信息是简单的：

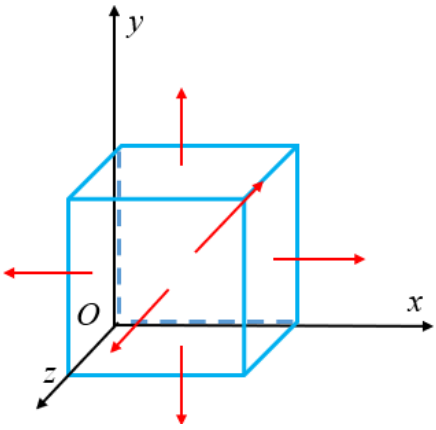


图 1 – 正方体模型示意图

规定正方体边长为 1，不难直接写出各个顶点的坐标，以及组成各个面的三角面面片的组成顶点，且外法向量（红箭头）是轴对齐的，也可以快速写出。此处不做过多阐述。

图 2 –正方体的 3D 模型（Windows 10 3D Viewer 打开）

下面对球的模型推导做简单说明。令球心在原点，半径为r，则球面的参数方程为：

$$\begin{cases} x = r\sin\alpha\sin\beta \\ y = r\cos\alpha \\ z = r\sin\alpha\cos\beta \end{cases}$$

其中参数$0 \le \alpha \le \pi, 0 \le \beta \le 2\pi$。

使用经纬线的交点以三个为一组，即可构建一系列三角形面片，作为球面的表示。经度和纬度的采样间隔决定了三角形面片的精细程度。在三个一组选择面的构成顶点时，可以采用如下原则达到不重不漏：对于一条纬线，以经度为 0 的点作为起点，向同纬度经度增加的方向再选取一个点。然后向纬度减少的方向旋转得到一个点，这三个点可以组成一个三角面片；再向纬度增加的方向旋转得到一个点，这三个点也可组成一个三角面片。完成后移动到下一经度。重复上述操作，可获得所有三角面片。

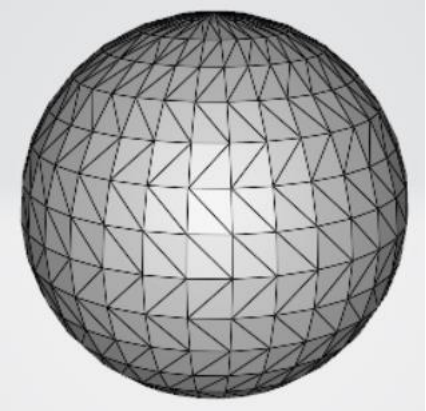


图 3 – 球模型的三角面片示意图

对于球模型的面片的外法向量，可以按照类似“垂径定理”的方式，面片的外法向量是球心与面片中心（可使用重心代替）的连线向量。由于每个三角面片的几何是一致的，因此重心容易求得并复用。在计算机图形学中，我们通常将法向量放到顶点上表示。**为了维持球面在光照下的光滑性，需将面片三个顶点的法向量设为相同。**综上，示意图如下：

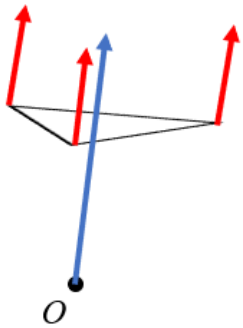


图 4 – 球面的三角面片的法向量计算示意图

虽然基于上述推导可以通过遍历的方式求出所有所需的物体数据结构，但现代图形学中，通常使用 3D 建模软件（如 3DS Max、Maya）来构建 3D 物体。因此，本实验中采用 3D 建模软件 Blender [4] 构建，导出.obj（波前）格式的模型文件。所有面片都是三角形，点坐标做了到[-1, 1]的归一化，以便于绘制。模型文件在 1-display/model 目录下。这样，即可获得顶点坐标、法向量、面片等。.obj 文件的解析工作由 framework/OBJProcessor.hpp 逐行读取完成。

关于.obj 文件的具体格式组成，此处做一些基本介绍，具体可参考维基百科页面[3]。以下为从球面模型 1-display/model/UnitCube.obj 中截取的一段和相应解释：

（见下页）

示例行	解释
v 1.000000 -1.000000 -1.000000	定义一个顶点，$(x,y,z) = (1,-1,-1)$
vt 0.625000 0.250000	定义一个材质贴图采样点，$(x,y) = (0.625, 0.25)$
vn 0.0000 -1.0000 0.0000	定义一个法向量，$(x,y,z) = (0,-1,0)$
f 4/12/6 6/8/6 8/6/6	斜杠首元：定义一个面，由 4、6、8 号顶点（v）组成； 斜杠中间元：定义一个材质贴图面，将 12、8、6 号采样点（vt）决定的图块贴到定义的面上； 斜杠末元：定义面的法向量是 6 号 vn。 注意.obj 中编号从 1 开始。

本实验中图形填充纯色，不需要材质贴图相关信息。使用 Windows 10 自带的 3D 浏览器打开 UnitCube.obj 文件，即可清晰地看到各三角形面片（mesh 网格）：

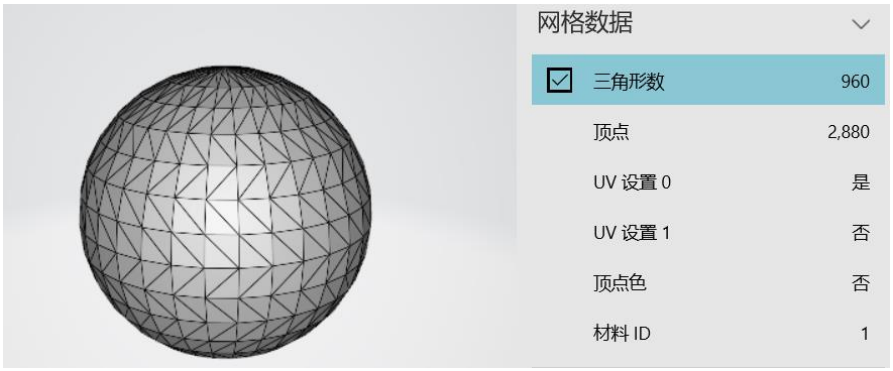


图 5 – 球面的 3D 模型

下面介绍绘制时颜色的设定。由于进行纯色绘制，故在着色器中使用 Uniform 型变量 uColor 存储颜色。正方体的颜色为灰色：(R, G, B, A) = (127, 127, 127, 1.0)，在向着色器传递时，需要将 RGB 分量归一化，即除以 255，结果为 (R, G, B, A) = (0.498, 0.498, 0.498, 1.0)。球体的颜色为蓝色：(R, G, B, A) = (29, 156, 215, 1.0)。二者的不透明度 A 均设为 1。

在讲解**视见变换的实现**前，首先对顶点着色器中顶点坐标的变换过程（pipeline）进行说明。见 1-display/shader/vMain.glsl，其中与几何相关的部分如下：

```
gl_Position = uPerspectiveMatrix * uWorldMatrix * uModelMatrix * aPosition;
```

aPosition 为传入的顶点原始坐标，**在图形学中常使用增广坐标和增广矩阵以简化计算**，即变换矩阵是 4×4 的，坐标是 4 个元素的；uModelMatrix（**模型矩阵**）用于实现对物理自身的变换，如自旋；uWorldMatrix（**世界矩阵**）用于实现世界坐标系的变换，是实现漫游观察的重点；uPerspectiveMatrix（**透视矩阵**）用于产生透视投影效果。

在本实验中，我们固定 uPerspectiveMatrix 为 glm::perspective(radians(fovy), aspect, near, far)的运算结果，其中 fovy 表示 y 向视野大小，设为 90（度）；aspect 为视场宽高比，设为窗口宽高比（1:1）；near 和 far 两个参数用于裁剪离视锥过近和过远的物体，可

以控制透视效果的强度，设为 0.05 和 2.8。

对透视矩阵的具体推导超出了本实验的目的，此处只给出它的一种形式：

$$\mathbf{perspective} = \begin{bmatrix} f/aspect & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & -(near+far)/d & -2\times near\times far/d \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

其中$f = 1/\tan(\text{radians}(fovy)/2)$, $d = far - near$。

为了避免同时渲染的球体和正方体中心默认重合，以及相互遮盖，导致难以区分两个物体，可以给二者分别设置不同的 uModelMatrix 以进行自身的预先变换。本实验中，球体的初始化模型矩阵是 scalem(vec3(0.2, 0.2, 0.2)) * translate(0, 0, 1.0)，即三轴 0.2 倍缩放，并在 z 轴平移 1.0；正方体的是 scalem(vec3(0.1, 0.1, 0.1)) * translate(0.5, 1.5, -1.0)。缩放矩阵和平移矩阵的形式是简单的：

$$\mathbf{scalem}(x,y,z) = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{translate}(x,y,z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

另外，还需要使用 glEnable(GL_DEPTH_TEST)语句启用 OpenGL 的深度测试功能，以在两个物体重合处获得正确的深度方向裁剪关系。

现在将介绍**世界矩阵** uWorldMatrix **的管理**，即实现视见变换、漫游观察的重点。我们可以使用对一个相机的维护来管理。uWorldMatrix 通常设为一个 lookAt 矩阵，其需要三个参数：***eye*** 表示相机的位置、***at*** 表示相机看向的方向、***up*** 表示相机向上的方向。要求 ***eye→at*** 和 ***eye→up*** 不能共线。一种等价的形式是，将 ***at*** 分解为 ***eye***+***front***，即维护“相机前”向量，且为了支持相机移动，还需要维护“相机右”向量 ***right***。

对 lookAt 矩阵的具体推导超出了本实验的目的，此处直接给出它的形式：

$$\mathbf{lookAt} = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & n_y & n_z & -\boldsymbol{n}\cdot\boldsymbol{eye} \\ u_x & u_y & u_z & -\boldsymbol{u}\cdot\boldsymbol{eye} \\ v_x & v_y & v_z & -\boldsymbol{v}\cdot\boldsymbol{eye} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

不难看出第一个等式其中左边的矩阵负责旋转，右边的矩阵负责平移；第二个等式中，$\boldsymbol{v} := \boldsymbol{at} - \boldsymbol{eye}$、$\boldsymbol{n} := \boldsymbol{v} \times \boldsymbol{up}$、$\boldsymbol{u} := \boldsymbol{n} \times \boldsymbol{v}$，粗体表示向量，带下标表示向量的分量，点号表示点积，叉号表示叉积，:=表示右手边运算后向量单位化。

相机的实现在 framework/Camera.hpp 中。计算机图形学中，常使用航空领域的欧拉角 Pitch-Yaw-Roll（俯仰角-偏航角-翻滚角）来管理相机的旋转。示意图如下：

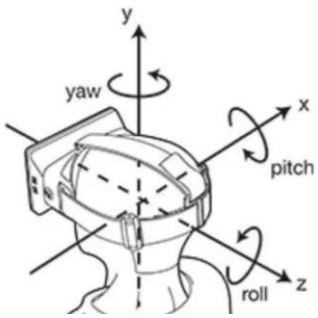


图 6 – 欧拉角示意图 [https://www.jianshu.com/p/1e38ad3d4bfb]

由于我们只需要围绕物体观察，以及上下抬头的运动，而不是在制作一款飞机驾驶游戏，需要横滚、翻越等运动，因此 Roll 角可以忽略，且 ***up*** 可固定为(0, 1, 0)。

首先只考虑相机的旋转，将相机置于原点(0, 0 ,0)，令 ***front*** 模为 1，几何如下图：

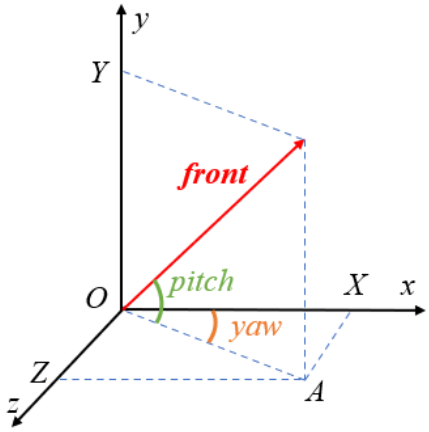


图 7 – 相机旋转的几何示意图

则有：

$$\begin{cases} OX = OA \times \cos yaw = \cos pitch \times \cos yaw \\ OY = \sin pitch \\ OZ = OA \times \sin yaw = \cos pitch \times \sin yaw \end{cases}$$

即获得了 ***front*** 向量。进一步地，使用$\boldsymbol{right = front \times up}$，即获得了相机右向量。上述运算完成后均应进行向量单位化。

这说明，**只需要将用户鼠标的操作转换为 Pitch-Yaw 角的变化，即可完成视见变换中的旋转部分。**例如，将横向的运动量按一定比例变为 yaw 角的变化，将纵向的运动量变为 pitch 角的变化。注意 pitch 角应维持在±90°以内，以避免空间翻转。相关实现在代码中的 mouseMoveCallback 函数。

当我们维护好 ***front*** 和 ***right*** 向量后，相机的空间位置移动是简单的。可使用如下公式表示（组分向量均为单位向量）：

$$\boldsymbol{pos'} = \boldsymbol{pos} + (\boldsymbol{right} \times x, \boldsymbol{up} \times y, \boldsymbol{front} \times z)$$

(x, y, z)是三轴的运动量，可通过用户按下相关按键的时间按一定比例变化后得出，相关实现在代码中的 keyboardHandler 函数。

至此，我们完成了视见变换的实现。下面给出一些从不同位置、不同角度观察场景的示意图。

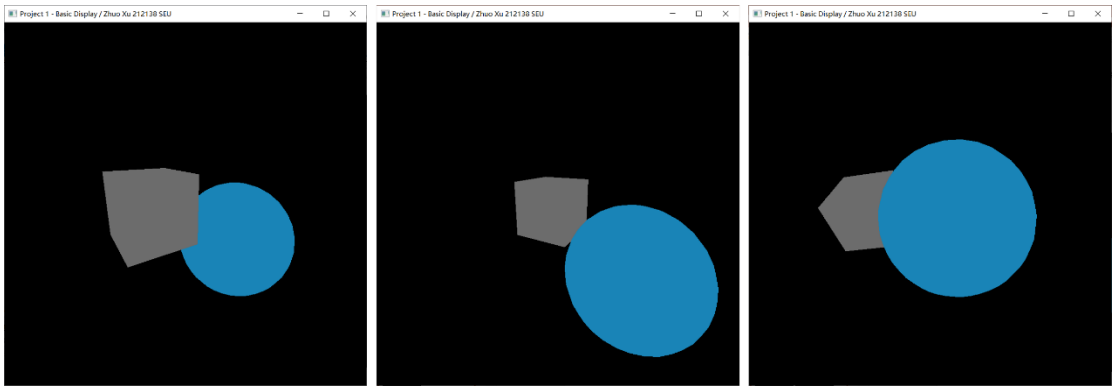


图 8 – 视见变换效果示意图

接下来，将介绍**光照效果的添加**。本实验的光照流程与本人本科的《计算机图形学》课程的光照相同。其使用《交互式计算机图形学：基于 WebGL 的自顶向下方法（第 7 版）》[2]中给出的 **Phong 光照模型**，并在着色器中实现（shader/vMain.glsl、shader/fMain.glsl）。下面对 Phong 光照模型进行介绍。

Phong 光照模型巧妙地将光照效果分为三个组分：**环境光（Ambient）**用于给物体赋整个环境下的基础的光照，该分量与法线无关；**漫反射（Diffuse）**添加了与光照位置和法线相关的分量；**镜面反射（Specular）**则给物体添加了高光，该分量与观察视点也有关。示意图如下：

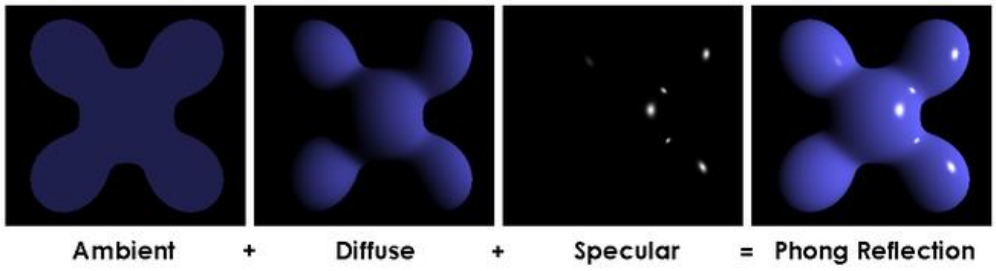


图 9 – Phong 光照模型示意图 [https://en.wikipedia.org/wiki/Phong_reflection_model]

环境光（Ambient）分量无需特别计算，因此只需要从主程序直接传递到着色器中即可。

漫反射（Diffuse）分量的计算是使用因子 Kd 对 100%发生漫反射的颜色进行加权。$\mathrm{Kd} = \max(\boldsymbol{L} \cdot \boldsymbol{N}, 0)$。$\boldsymbol{L}$向量是光源位置与光照点位置的连线向量单位化，$\boldsymbol{N}$向量是做了变形矫正后的光照点的法向量单位化，取 max 是因为$\boldsymbol{L} \cdot \boldsymbol{N} < 0$时光线没有打到片元外表面。该因子即表示**光线垂直于光照面的分量权重**（符合 Lambert 余弦定律）。变形矫正是指，在 uWorldMatrix 中进行某些修改时，可能导致法向量变形，因此需要对原始法向量乘以$(\mathbf{uWorldMatrix}^{-1})^{\mathrm{T}}$（先转置还是先求逆不影响）进行修正。具体数学原理略去。

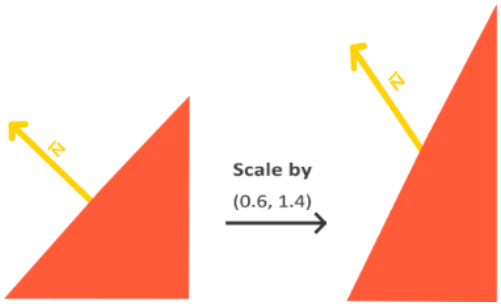


图 10 – 法向量变形示意图 [https://learnopengl.com/Lighting/Basic-Lighting]

镜面反射（Specular）分量的计算是使用因子 Ks 对 100%发生镜面反射的颜色进行加权。$\mathrm{Ks}=\begin{cases}0, \text{if } \boldsymbol{L}\cdot\boldsymbol{N}<0\\ (\max(\boldsymbol{N}\cdot\boldsymbol{H},0))^{shiness}, \text{otherwise}\end{cases}$。其中$\boldsymbol{L}$和$\boldsymbol{N}$的意义已于漫反射中说明；$\boldsymbol{H}:=\boldsymbol{L}-\boldsymbol{posToWorld}$，$\boldsymbol{posToWorld}$是 uWorldMatrix 作用后的顶点坐标，:=表示右手边运算后向量单位化；$shiness$用于调节高光程度，越大高光越集中。

在完成三个分量的计算后，求和即得到光照的综合颜色向量，将其乘到原始颜色上，即完成光照的应用。

光照的计算是比较复杂、充满细节的，仅凭一两页的介绍难以完全覆盖，具体的细节应参考源代码中的一系列 PhongLightModel 对象，以及着色器中的计算流程。另外，《交互式计算机图形学：基于 WebGL 的自顶向下方法（第 7 版）》[2]中将三个分量的原始色（100%发生的颜色）分解为两个 RGBA 颜色分量的乘积：质地 Material 和光色 Color。这样，可以对材料本身的颜色，以及光照本身的颜色进行分别的调整，两者相乘（相当于相关运算）即得到综合颜色 Product。

下面给出“改变镜面反射参数观察球面显示效果”的一系列示意图。共同参数如下：

光源位置	Ambient 质地	Ambient 光色	Diffuse 质地	Diffuse 光色
(0.1, 0.1, 0.1)	(230,230,230)	(240,240,240)	(60,60,60)	(240,240,240)

各图不同参数如下：

图序号	Specular 质地	Specular 光色	Specular Shiness
1	(255, 255, 255)	(255, 255, 255)	35.0
2	(255, 255, 255)	(255, 255, 255)	2.0
3	(255, 0, 255)	(0, 0, 255)	2.0
4	(255, 255, 255)	(255, 255, 0)	10.0

各图的显示效果均符合预期。图 1 中高光集中，颜色呈白色；图 2 中高光分散；图 3 中 B 通道相关度高，颜色呈蓝色；图 4 中 R-G 相关度高，颜色偏黄绿色且高光较集中。

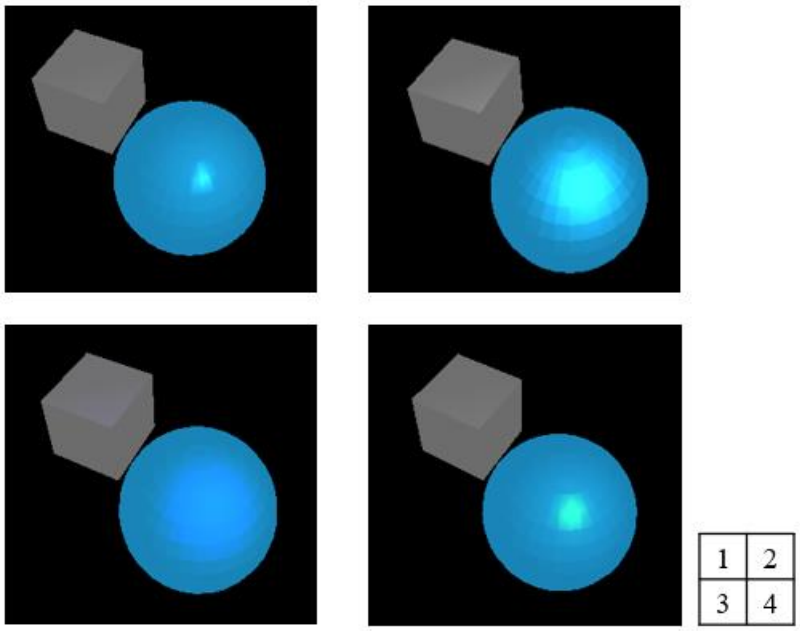


图 11 - 改变镜面反射参数观察球面显示效果

至此，Project 1 的所有要求与说明、讨论均完成。

参考资料

[1] 卓旭, 高钰铭, 沈汉唐. typed-webgl (东南大学《计算机图形学》课程实践作业) [CP/OL]. 2021. https://zxuuu.tech/seucg & https://github.com/z0gSh1u/typed-webgl .

[2] Edward Angel, Dave Shreiner. 交互式计算机图形学：基于 WebGL 的自顶向下方法（第 7 版）[M]. 张荣华 等译. 北京:中国工信出版集团. 2016.

[3] Wikipedia. Wavefront .obj file [EB/OL]. 2021. https://en.wikipedia.org/wiki/Wavefront_.obj_file .

[4] Blender. Blender [CP/OL]. 2021. https://www.blender.org/ .

其他声明

本报告中的所有外部引用图片均在图片题注中标注来源。对于没有标注的图片，均为笔者自行绘制或程序的执行结果，请规范引用。

Project 2：体绘制的光线跟踪（Ray Casting）算法实现与课程报告

目录

零、体绘制和光线跟踪的研究背景

在我们生活中，经常需要面对 3D 数据（体数据）。无论是医学领域的 CT、MRI 扫描，亦或是电子游戏中的三维物体。而我们最常用的显示设备：屏幕，只能进行二维的图像显示。因此，**我们需要一种将体数据渲染到二维平面的技术——体绘制技术**。

光线跟踪（或光线投射，Ray Casting）算法，是一种最常用的体绘制算法。其容易并行化、向 GPU 移植，以支持实时绘制，且算法流程清晰易懂，容易实现。本 Project 的主要目的，就是实现 Ray Casting 算法，并对体绘制的相关技术进行讨论与报告。

一、本实验的运行

本实验的有关代码存放在 2-raycasting 目录，使用的 CT 体数据需单独下载。程序的运行流程如下：

首先，编辑配置文件 config.json，示例如下：

config.json

```
{
    "WindowWidth": 512,
    "WindowHeight": 512,

    "VolumePath": "./model/lungct_C052_512_512_340.raw",
    "VolumeWidth": 512,
    "VolumeHeight": 512,
    "VolumeZCount": 340,

    "ImagePlaneWidth": 512,
    "ImagePlaneHeight": 512,

    "TransferFunction": "TF_CT_Bone",
    "MedianFilterKSize": 0,
    "SamplingDelta": 0.5,
    "EnableLighting": true,
    "KAmbient": 0.6,

    "MultiThread": 4
}
```

图 12 – config.json 文件示例

其中各配置项的说明如下表：

配置项	说明
WindowWidth、WindowHeight	显示窗口的宽、高，建议设为与视平面相等
VolumePath	体数据路径，体数据格式见后文
VolumeWidth、VolumeHeight、VolumeZCount	体数据的宽、高以及在 z 方向的层叠切片数
ImagePlaneWidth、ImagePlaneHeight	视平面的宽、高
TransferFunction	要使用的传递函数，具体见后文

（续于后页）

（续前表）

配置项	说明
`MedianFilterKSize`	在视平面上进行中值滤波的核大小，设为 0 表示不进行中值滤波去噪
`SamplingDelta`	进行 Ray Casting 时光线对体素的采样间隔，设为 1 会得到较粗糙的渲染，设为 0.5 会得到较精细的
`EnableLighting`	是否启用光照，设为 false 则不对体数据添加光照
`KAmbient`	Phong 光照环境光分量权重，更大的权重会增亮视平面
`MultiThread`	使用的多线程数

在配置好配置文件后，将启动项目设为 2-raycasting，使用 Release x86 模式编译运行解决方案 seu-viz.sln 即可。若一切无误，可看到体绘制渲染结果类似下图所示：

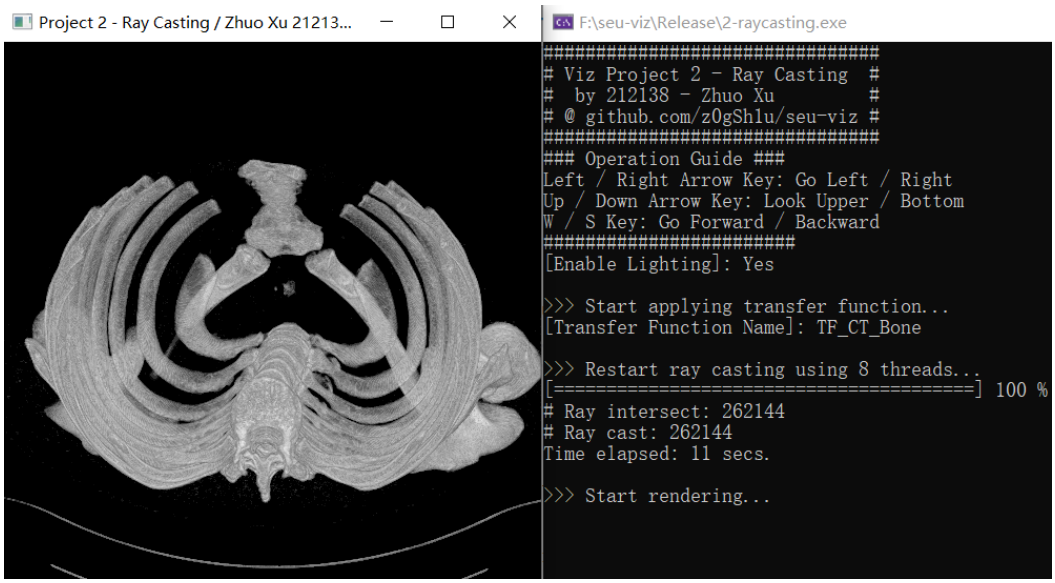


图 13 – 实验程序的运行示意图一瞥（骨传递函数，启用光照）

通过左、右方向键，可以左右旋转体数据观察；通过上、下方向键，可以上下翻动体数据观察；通过 W、S 键，可以在 z 轴前后移动观察。

下面，我将具体介绍本实验的所有流程和细节。

二、Ray Casting 算法的主要流程

首先，笔者将结合自己的所学以及罗立民老师的课件[1]：Viz7-体积绘制，对体绘制的 Ray Casting 算法（下面简称 RC 算法）的主要流程与思路进行说明。RC 算法是一种对三维数据（体数据）进行向二维平面绘制的算法，帮助我们对体数据进行观察。请注意，在图形学领域有一个名称类似的“光线追踪”（Ray Tracing）算法，因 NVIDIA 的 RTX 系列显卡使得其又开始受欢迎。它和此处的 RC 算法不是一个概念，应加以区分。

步骤 1：在物体坐标系中确定观察轴线。即确定光线源点和物体中心点。

步骤 2：确定成像平面。其与轴线垂直并由视锥（若为透视投影）限定。

步骤 3：确定光线方向向量。从成像平面的每一个像素投射出一条光线照向体数据，光线可用直线的参数方程表示。

步骤 4：确定光线与物体的相交情况，收集相关体素的信息。对于相交有三种处理方式：SSD、MIP、DRR。后文将具体介绍。

步骤 5：根据收集到的相关体素的信息，计算该光线对应的视平面像素值。

三、本实验的成像几何

在进行体绘制之前，有必要先对本实验采用的成像几何模型进行说明，具体如下：

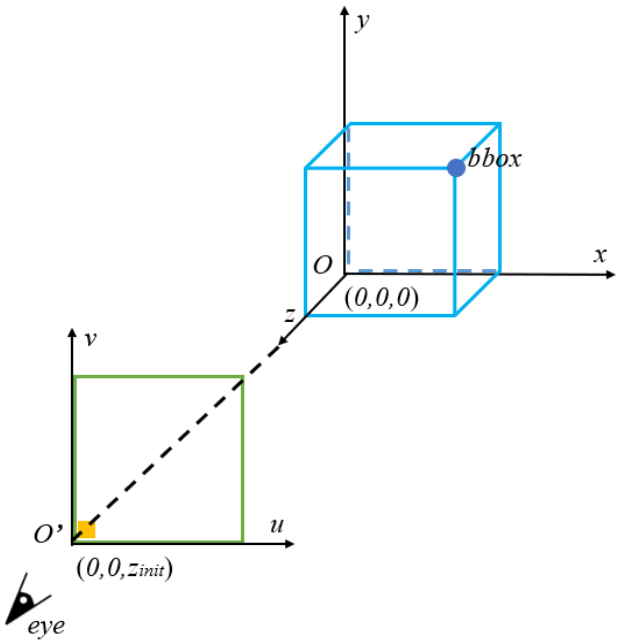


图 14 – 本实验使用的成像几何

世界坐标系为右手系 *O-x-y-z*。蓝色盒表示体数据的包围盒（Bounding Box），包围盒**左后底**部角放在世界坐标系 *O* 点（(0, 0, 0)）。进一步规定 *x*、*y*、*z* 轴分别为包围盒左右、上下、前后面的法向量（即包围盒轴对齐），则只需使用包围盒**右前顶**部角的坐标 *bbox* 即可确定包围盒。本实验使用的包围盒坐标的区间是**左闭右闭**的，故判断某点是否在包围盒中的代码如下：

```
bool inBBox(const vec3 &point, const vec3 &bbox) {
  return point.x >= 0 && point.x <= bbox.x && point.y >= 0 &&
         point.y <= bbox.y && point.z >= 0 && point.z <= bbox.z;
}
```

视平面坐标系为二维坐标系 *u-O'-v*。*O'*点落在 *z* 轴上，在世界坐标系(0, 0, z_{init})处。*u* 在列方向，与 *x* 同向；*v* 在行方向，与 *y* 同向，图像生长的方向为 *u*、*v* 正向。该定义下 *v* 坐标值代表的不是视图第 *v* 行，而是第 *H*-1-*v* 行（其中 *H* 为视图高，索引从 0 开始）。

进一步地，规定采用**平行光投影**方式而非透视投影。则光线的行进如下图所示（绿色为视平面，蓝色为体数据在 *z* 轴上的层叠）：

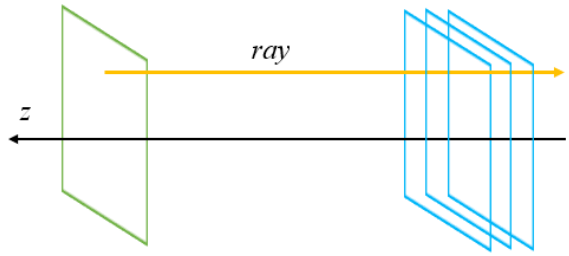


图 15 – 平行投影下的光线行进示意图

此时有如下性质，使得成像几何可以进一步简化：①视平面的能够激发出能穿过体数据的光线的像素的总体尺寸等于体数据包围盒六个面的最大面，即**视平面尺寸只要与最大面相等**，即能保证对体数据的完整观察；②**视平面在z轴上的前后位置变得不那么重要**，只要坐标大于 *bbox* 的 z 值，就不会进入包围盒内部，能够完整看到整个体数据。除非进入了包围盒内部或者运动到了光线无法穿过包围盒的位置，否则视平面的 z 值变化不影响成像结果。因此，**默认情况下，z_{init}被设为 VolumeZCount＋1**，这足以囊括整个体数据。

四、体数据的准备和读取

由于本 Project 准备采用**真实 CT 扫描数据**，而非手工构造的体数据进行实验，因此本 Project 对体数据的格式有如下要求：①通过堆叠一系列二维图像形成三维体数据；②二维图像的存储顺序是行扫描顺序，像素格式为 16 bit 无符号数，字节序为小端；③不应有非体素值的残余数据，通常保存为.raw 格式。另外，由于在存储体数据以及应用传递函数后的体素值时，需要开辟与体素数量正相关的、大的数组，鉴于操作系统对程序内存申请量的限制，不建议使用尺寸过大的体数据。

我为本 Project 准备的一份体数据可在东南大学网盘下载。**在使用该体数据时，请注意医学伦理问题：**

> https://pan.seu.edu.cn:443/link/26E21BE610148F43AE4862E2B815FEC4
> 有效期限：2026-12-08；访问密码：euGo

这是西门子 CT 机对病人胸腔部分进行扫描的图像序列，尺寸为**H/W/Slice=512/512/340**。其来源于 AAPM 的 Low Dose CT 挑战赛数据集 [4]中的 C052 号扫描序列，原始格式为 DICOM。通过我在研究生阶段编写的 CT DICOM 到 raw 文件格式的转换工具 ctdicom2raw [3]，可获得各切片的 raw 图像。进一步地，使用科研看图工具 ImageJ [9]，可将切片堆叠成三维格式，即得到体数据。其中各体素值为**进行 Rescaling 之前**的“CT 值”。例如，本数据集设备采用的 Rescale Slope / Intersect 函数为$y = 1x - 1024$，则将各体素值减去 1024 后即为真正的 CT 值（HU 值）。**借助 HU 值进行传递函数设计**，类似于加窗观察，更具备物理意义与指导价值，具体将在后文描述。

该体数据的 Slice 为横断面（Transverse Plane），扫描部位为胸腹联合。低 Slice 部分为人体上部，到肩膀位置；高 Slice 部分为人体下部，到纵膈附近可见肝脏的位置。按

照前文所述的成像几何，**按照 Slice 序号大小将切片放上 z 轴**，默认情况下光线从人体下穿越到人体上，如下图所示：

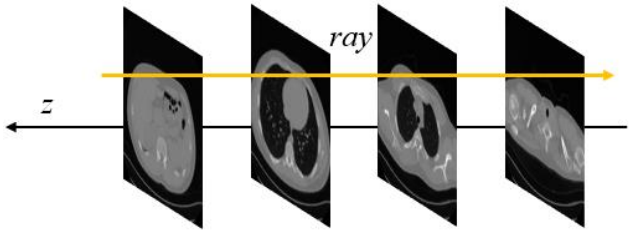


图 16 – 光线穿越示例体数据

在读取体数据时，首先创建一个大小与体素数相等的 uint16 型数组，然后逐两字节将 raw 图像的二进制数据填充到该数组中即可。至此，完成了体数据的获取和读取。由于使用一维数组存储三维数据，因此还需定义一个辅助函数用于计算特定坐标的体数据的一维索引。相关代码如下：

```
uint16 *volumeData;
volumeData = new uint16[VoxelCount];
readFileBinary(VolumePath, BytesPerVoxel, VoxelCount, volumeData);

// Get voxel index in volume data.
int getVoxelIndex(int x, int y, int z) {
  return PixelPerSlice * z + VolumeWidth * y + x;
}
```

五、使用传递函数对体数据进行分类赋色

传递函数在体绘制中有很重要的作用。**它的职责是对各个体素赋一个颜色值**，通常使用 RGBA 四通道。对于我们不感兴趣的部分，可以将 A（不透明度，Opacity）通道设为 0，这样即可透明地穿过该体素；对于我们感兴趣的部分，可以赋合适的 A 值，并按照需求或喜好赋合适的 RGB 颜色值。以上文提到的 CT 数据为例，可对 HU 值在骨的典型值附近范围的体素赋灰色值，A≠0，具体灰度可在一定范围内根据 HU 值线性调节；对其余类型的值赋 A=0。那么，进行 RC 算法后，即可得到对骨的观察，而其他解剖结构不被显示，就好像透明了一样。

典型的传递函数是一个关于体素值的分段函数。得益于我们使用 CT 扫描图像的先验，可以根据 HU 值设计传递函数，以观察感兴趣的解剖结构。本实验中，我在 transferFunction.hpp 中预定义了 3 个传递函数，说明如下。注意其中的体素值均为 Rescaling 之前的值，即 HU+1024；带波浪号的范围表示具体灰度值在其之间根据体素值线性分配；区间的开闭没有严格表示：

（续于后页）

传递函数名	分段策略		说明
TF_CT_Bone	< 1200	A = 0	关注骨的传递函数。在1200~2200 范围内显示为灰色。其余解剖结构透明。
	[1200, 2200]	A = 0.1 RGB = (180, 180, 180) ~ (240, 240, 240)	
	> 2200	A = 0	
TF_CT_MuscleAndBone	< 1040	A = 0	同时展示肌肉和骨的传递函数。对肌肉（[1040, 1155]）渲染为近似肉色。 对骨同 TF_CT_Bone，但使用较小的不透明度。 其余解剖结构透明。
	[1040, 1155]	A = 0.05 RGB = (255, 188, 155) ~ (255, 238, 205)	
	[1155, 1200]	A = 0	
	[1200, 2200]	A = 0.07 RGB = (180, 180, 180) ~ (240, 240, 240)	
	> 2200	A = 0	
TF_CT_Skin	< 880	A = 0	展示皮肤的传递函数。渲染为近似肉色。其余解剖结构透明。
	[880, 925]	A = 0.8 RGB = (255, 198, 165) ~ (255, 213, 180)	
	> 925	A = 0	

这些预定义的传递函数均写入配置文件后使用。如果需要添加新的传递函数，只要在 transferFunction.hpp 按类似格式编写完成后，注册到主程序的 TransferFunctionMap 即可调用。

对体数据应用传递函数的函数是代码中的 applyTransferFunction。它接收全体体素 volumeData，分别赋颜色后填充 coloredVolumeData 数组。

下面，我将展示一些在不同传递函数下绘制的结果图像。采样间隔均为 0.5，均启用光照，KAmbient 为 0.6。

（因图像尺寸较大，于下页继续）

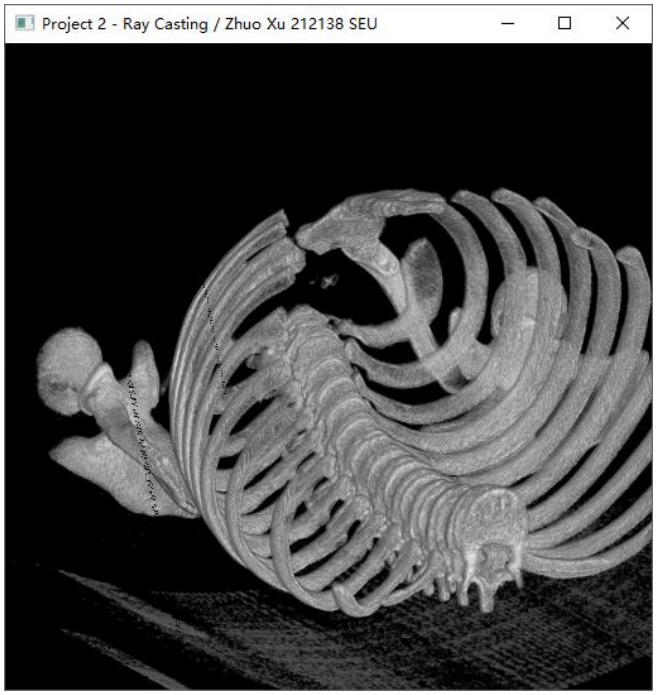


图 17 - TF_CT_Bone 下的观察，旋转到了左上侧面。
一些可见的黑点是体数据边沿欠采样造成的。

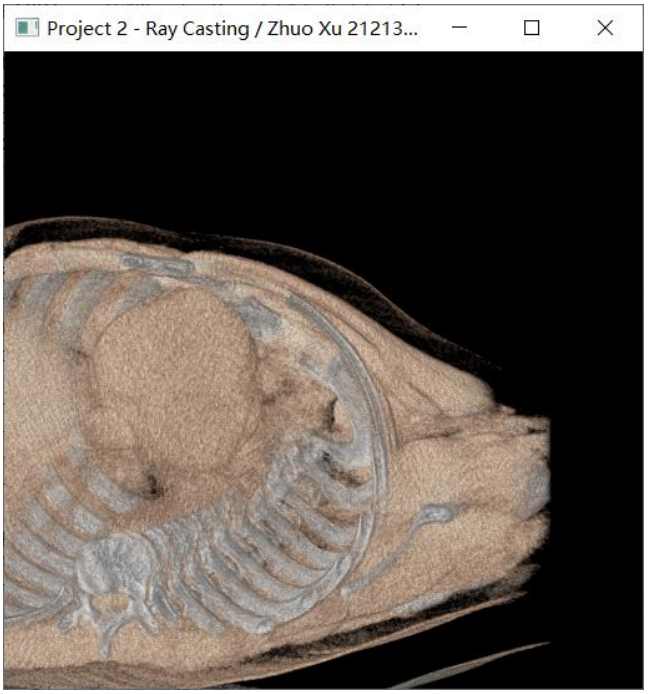


图 18 - TF_CT_MuscleAndBone 下的观察，旋转到了右侧面并进行了些许纵深移动。
可见肌肉、骨骼以及心脏。肋骨包围的空腔是肺部的部分，因含空气，
按 HU 值被设为透明了。

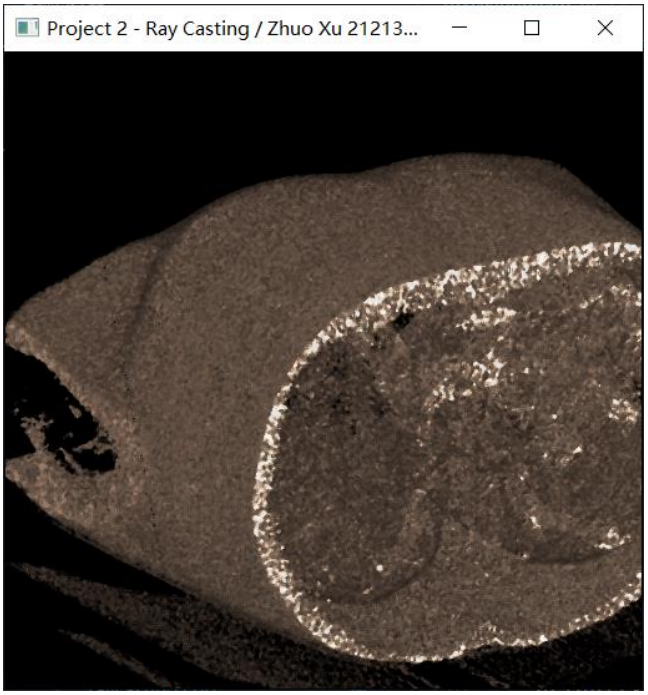


图 19 - TF_CT_Skin 下的观察，使用了核尺寸为 3 的中值滤波。

由于皮肤位于 X 线发生穿透的最前沿，散射、光照不平整等现象严重，因此绘制效果较差，只能看出外皮的大部轮廓。另外，断层结束的位置是“平整切片”的，所以光照下呈亮白色。

六、光线与包围盒的相交测试

在进行 RC 算法的过程中，关键步骤是按照光线发射的方向行进（marching），找到其穿越的所有体素，采样，反复进行色彩合成。因此，有两个关键的问题需要解决：**①判断光线是否有穿过体数据，即是否与包围盒相交；②光线对包围盒的入点在什么位置。**为此，我们需要对光线与包围盒进行相交测试。

本实验中，相交测试的主要函数为 intersectTest（三轴相交测试）和 _intersectTestOnePlane（单轴相交测试）。前者在 x、y、z 三个方向调用后者以对包围盒三个方向共六个面进行相交测试。光线（或者称射线）与包围盒的相交测试是图形学领域的重要问题。本实验采用 Eric Haines 在 1989 年的提出的“**厚板方法**”[6]进行相交测试。该方法能够对射线与轴对齐包围盒的相交进行测试。它的做法类似于我在本科《计算机图形学》课[2]上学到的，以及罗老师上课讲述示例代码时提到的 Cyrus-Beck 的直线裁剪算法。我主要参考了知乎网站上的文章《射线与包围盒的相交测试》[5]中的描述和伪代码，进行了略微的修正后进行了代码实现。首先将该算法描述如下：

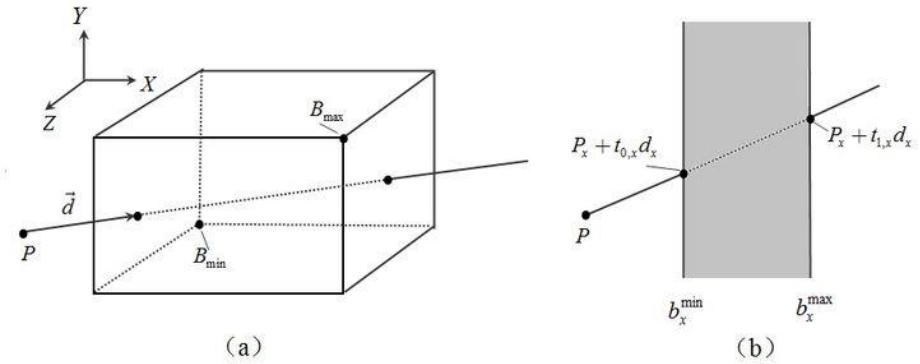


图 20 – **射线与轴对齐包围盒的相交测试示意图** [https://zhuanlan.zhihu.com/p/138259656]

记射线参数方程$L(t) = P + t\vec{d}$，$P = (P_x, P_y, P_z)$为起点，t为参数，$\vec{d} = (d_x, d_y, d_z)$为单位方向向量。

记轴对齐包围盒$(B_{\min}, B_{\max})$（表示包围盒的两个对角点），其中$B_{\min} = (b_x^{\min}, b_y^{\min}, b_z^{\min})$，$B_{\max} = (b_x^{\max}, b_y^{\max}, b_z^{\max})$，$b_i^{\min} \prec b_i^{\max}, i \in \{x, y, z\}$。本算法的“偏序小于”表示“先于”之意。

一个包围盒有 6 个矩形面，把两个平行的矩形看作一块厚板，则问题变为检测射线与垂直的 3 块厚板的相交。初始阶段，设$t_0 = -\infty$，$t_1 = +\infty$。**请注意，本算法讲述的包围盒均只有边界面，内部空心。**

首先处理 *YZ* 平面，如上图所示，相当于从 *P* 点从左往右观察包围盒，判断射线与左右面$X = b_x^{\min}$和$X = b_x^{\max}$的相交情况。若射线与这两个面相交，规定$t_{0,x} \prec t_{1,x}$，则如下式子满足：

$$\begin{cases} P_x + t_{0,x}d_x = b_x^{\min} \\ P_x + t_{1,x}d_x = b_x^{\max} \end{cases}$$

求解之有：

$$\begin{cases} t_{0,x} = (b_x^{\min} - P_x)/d_x \\ t_{1,x} = (b_x^{\max} - P_x)/d_x \end{cases}$$

则若$d_x \neq 0$（即射线在行进过程中有x方向的运动分量），则取$t_0 = \max(t_{0,x}, t_0)$，$t_1 = \min(t_{1,x}, t_1)$。若$d_x = 0$，那么射线与 *YZ* 平面平行，考虑P_x，若$P_x \prec b_x^{\min}$或$b_x^{\max} \prec P_x$（即射线起点x分量在包围盒以外），则一定不相交；否则仍可能相交（例如起点在包围盒内部，可能有一个射出点），暂时不做进一步处理。

在处理完 *YZ* 平面后，对 *XZ*、*XY* 平面也同理进行处理即可。

若至此，相交仍然是可能的，则存在如下两种情况：

① $0 < t_0 < t_1$，说明射线与包围盒有射入和射出**两个交点**，起点必然在包围盒以外，此时，t_0为射入点参数，t_1为射出点参数。

② $t_0 < 0 < t_1$，说明射线与包围盒**只在射出点相交**，起点在包围盒内部。此时，t_1为射出点参数，t_0为将射线退化为直线后在射线负向的相交点的参数。

通过参数，就可使用射线的参数方程$L(t) = P + t\vec{d}$确定相交点的 *x*、*y*、*z* 坐标了。

在进行完相交测试后，**若属于情况①**，则使用t_0确定的相交点作为体数据采样的起始点（在此之前的光线都没有穿过体数据）；**若属于情况②**，则使用光线起始点P作为体数据采样的起始点，对后续部分的体数据进行采样。而体数据采样的停止条件显然为采样点到达包围盒以外，也可用参数t_1确定的射出点来计算。

七、光线穿越体素与视平面像素值的合成

本节，我们将跟着一条光线一步步行进与采样，同时进行像素值的合成（blending）。相关的函数是代码中的 castOneRay。

对于视平面上一点(u, v)，将其变换到世界坐标系下，有：

$$(x, y, z)^{\mathrm{T}} = (u, v, z_{init})^{\mathrm{T}} \cdot \mathbf{R} + T$$

其中$\mathbf{R}$为 3×3 的旋转矩阵，T为平移向量。

首先考虑没有应用视见变换，改变视点观察的情况，取$\mathbf{R} = \mathbf{I}_3$（单位矩阵），$T = \mathbf{0}$（零向量）。按照前文的成像几何示意图，光线向$-z$方向（即$(0, 0, -1)$）行进，可将上述变换改写为参数方程：

$$(x, y, z)^{\mathrm{T}}(t) = \left(u, v, z_{init} + t\vec{d}\right)^{\mathrm{T}}$$

其中$\vec{d} = (0, 0, -1)$为单位方向向量，$t \geq 0$为参数。那么，通过对t的不断步进，就可以获得一系列世界坐标系下的坐标。如果这些坐标在包围盒内，就可以对体素进行采样、合成等操作了。而t的起始值与终止条件可参考上节：光线与包围盒的相交测试。**对于存在观察变换的情况，将在后文讨论。**

光线穿越体素过程中的算法处理有 SSD、MIP、DRR 等多种方案。

SSD（Shaded Surface Display）的做法是通过虚拟的外部光源，向体数据的满足特定表面条件的光线，其反射线射入观察者视野的结果作为体绘制的结果。例如，选择首个击中位置作为光线的反射点。

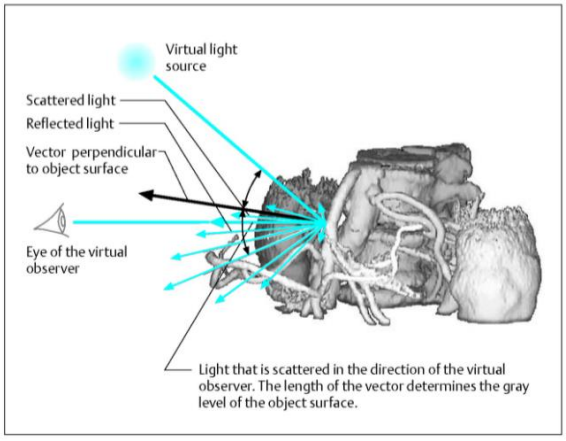


Fig. 2.20 Shaded surface display (SSD). A virtual light source illuminates the 3D object, and software calculates the light intensities that are scattered and reflected toward the observer.

图 21 – SSD 方案的示意图 [DOI: 10.1055/b-0034-79242]

MIP（Maximum Intensity Projection）是将视平面发出的光线行进过程中遇到的具有最大强度（Intensity）的体素的值作为绘制的结果。显然，该方式适合 CT 图像中的骨骼绘制、DSA/CTA/MRA 等血管造影技术中的血管绘制（有造影剂），等等。

DRR（Digital Reconstruction Radiography）是在光线行进过程中将路径上的体素值进行累加的结果作为绘制的结果。该方式可以绘制出透视的效果。

本实验采用的处理方案类似于 DRR，即顺着光线行进路径对上色后的体素值进行

采样与色彩融合，最终得到视平面上该像素点的颜色值。另外，本实验后续还对体数据添加了光照效果，这与 SSD 的方法也有关联。在实现了下述算法后，转换为 MIP 算法是简单的。

光线穿越体素过程中的算法流程描述如下：

第一步：初始化累积颜色 accumulated = (0, 0, 0, 0)。注意，本节中讨论的颜色均为 RGBA 四通道表示。

第二步：应用光线与包围盒的相交测试算法，获得t的起始值，进一步取得体素在$(x,y,z)^{\mathrm{T}}(t)$位置的应用传递函数（上色）后的值 sampleColor。

第三步：使用从前向后（Front To Back）的颜色合成公式，将 sampleColor 融合到 accumulated 中。本步骤的函数为代码中的 fusionColorFrontToBack，具体公式为：

$$\begin{cases} C_{out} = C_{in} + (1-\alpha_{in})\alpha_i C_i \\ \alpha_{out} = \alpha_{in} + (1-\alpha_{in})\alpha_i \end{cases}$$

其中C表示任一色通道（R/G/B），α表示 A 通道，下标为in的表示 accumulated 对应分量的原值，下标为out的表示 accumulated 对应分量的新值，下标为i的表示 sampleColor 的对应分量。

第四步：使用步进量$\delta > 0$步进$t \leftarrow t + \delta$，对新的上色后体素进行采样。步进量的调节为配置文件中的参数 SamplingDelta，更小的步进量能够带来更精细、丰满、硬实的绘制效果。但过小的δ会使得计算复杂度上升，且前序体素对 A 通道的贡献过大，导致后序体素难以被看到。另外，在对体素进行采样的过程中，采样坐标点$(x,y,z)^{\mathrm{T}}(t)$可能不落在体素的中心位置，而是在“亚体素”处，因此，需要对周围体素的上色值进行**插值**，才能得到采样点的颜色值。本实验采用的插值方法是**三线性插值**，其实现函数是代码中的 colorInterpTriLinear。它在 x、y、z 三个方向寻找相邻的上色后体素，进行线性插值。对于采样点(x,y,z)，输出值的公式如下：

$$\begin{aligned} v_{out} = {} & (1-x_d)(1-y_d)(1-z_d)v_{000} + x_d(1-y_d)(1-z_d)v_{100} \\ & + (1-x_d)y_d(1-z_d)v_{010} + (1-x_d)(1-y_d)z_d v_{001} + x_d y_d(1-z_d)v_{110} \\ & + x_d(1-y_d)z_d v_{101} + x_d y_d(1-z_d)v_{110} + x_d y_d z_d v_{111} \end{aligned}$$

其中$x_d = x - x_0$，$x_0 = \lfloor x \rfloor$，$x_1 = \max(x_0 + 1, bbox.x)$，对于$y$、$z$同理；$v_{ijk}$表示$(x_i, y_j, z_k)$处的上色后体素值。

第五步：反复重复第三步和第四步的操作，直到满足终止条件。终止条件为，采样点落到包围盒以外，或 accumulated 的 A 通道值已到达 1.0（即完全不透明，后续体素无法看到）。

第六步：将 accumulated 作为视平面(u, v)点的像素值。

注意，上述流程中涉及颜色运算的操作，应保持 RGBA 通道值落在[0, 1]区间。

通过上述算法，就能给视平面的每个像素赋一个颜色值了，这就是 RC 算法的结果。至此，我们获得了**没有改变视点观察的**（即观察轴线完全与成像几何示意图相同）、**没**

有添加光照的体绘制效果。在此给出两张示意图，传递函数使用 TF_CT_Bone。

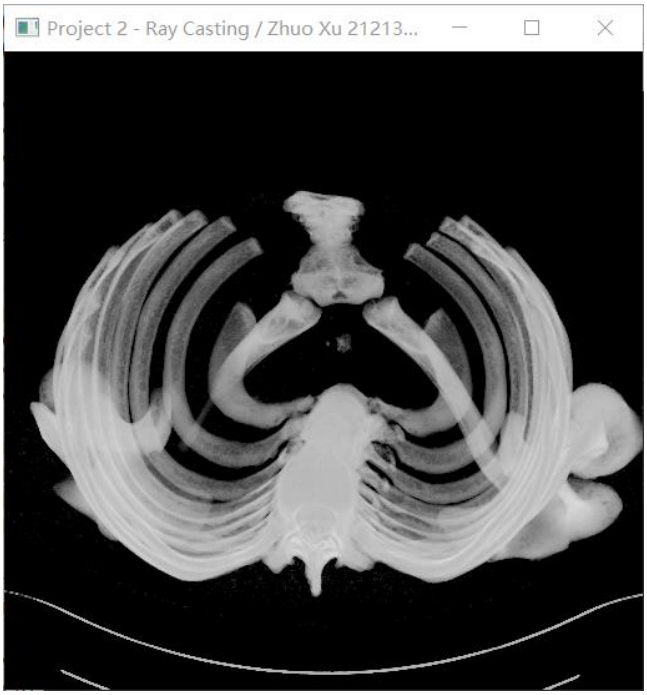


图 22 – **基础体绘制效果。**SamplingDelta = 1.0。

图 23 – **基础体绘制效果。**SamplingDelta = 0.5，
可见尤其是脊柱的绘制更加精细，骨的整体透明度下降。

八、改变视点观察（视见变换）

目前，我们的算法仅能从(u, v, z_{init})向$-z$方向发射光线，即从人体下方向观察。为了能够观察人体的上面、侧面等多方位的情况，需要支持三维空间的改变视点观察（漫游）的功能。为此，首先进行一些有关的数学推导。

对于视平面上一点(u, v)，将其变换到世界坐标系下，有：

$$(x, y, z)^{\mathrm{T}} = (u, v, z_{init})^{\mathrm{T}} \cdot \mathbf{R} + T$$

其中**R**为 3×3 的旋转矩阵，T为平移向量。那么，通过对**R**的维护，就能够实现对体数据左右、上下的旋转观察（运动是相对的，旋转视点与旋转体数据等效）；通过对T的维护，就能够在前后纵深方向观察体数据。

对于相机的基本原理和 lookAt 矩阵相关知识，在 Project 1 中已经详细说明。我们知道，lookAt 矩阵（4×4 增广）是相机的旋转变换与平移的综合表现：

$$\mathbf{lookAt} = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

其中左边的矩阵负责旋转，而右边的矩阵负责平移。由于我们使用向量T手工处理了平移操作，那么旋转矩阵**R**就**可以用 lookAt 矩阵的左上角 3×3 的矩阵表示**，即舍去最后一列的平移相关的向量。这样，我们依然只需要维护 eye（相机位置）、at（相机看向）、up（相机上）三个向量，即可借助 lookAt 函数获得旋转矩阵**R**，相关代码是 UntranslatedLookAt 函数。对于旋转矩阵，R、U、D三个向量的长度无关紧要，只需要方向正确即可，因此只需要在一个单位球（轨迹球）中维护这三个向量即可，代码中体现在 normalizedEyePos 变量。

对于平移向量T，在因观察点改变而发生旋转的过程中，它也应该跟着变化。因此，我们不妨做一些修改，将旋转矩阵整体作用到整体的坐标上，即：

$$(x, y, z)^{\mathrm{T}} = ((u, v, 0)^{\mathrm{T}} + T) \cdot \mathbf{R}$$

其中T的初始值$T_{init} = (0, 0, \mathrm{VolumeZCount} + 1)^{\mathrm{T}}$（初始位置在可以看到整个体数据的位置）。**在这样的推导下，前后（纵深）方向的运动对T的修改得以简化。**相关代码在 castOneRay 的前几行。

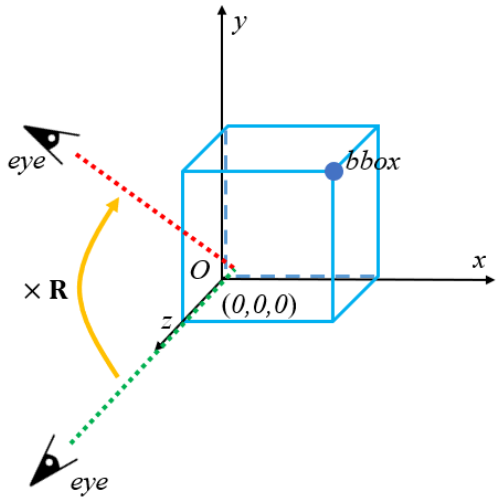


图 24 – **通过整体乘以旋转矩阵，可一次性完成视点变换，无需先旋转后平移。**

至此，我们可以通过公式完成观察视点的变换。固定相机上的方向为$+y$，只要维护

相机位置（方向键控制左右上下，W/S 键控制前后）以及相机看向点（由于不存在透视投影，不妨固定为原点），即可从不同位置不同方向看向体数据。

下面给出几张不同视角观察的示意图（添加光照，SamplingDelta=1）：

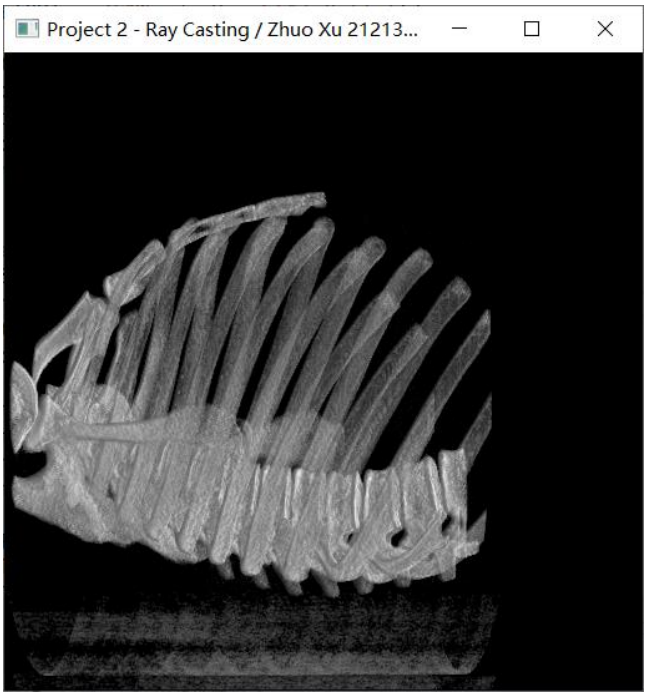


图 25 – TF_CT_Bone 下在对矢状面（Sagittal Plane）从左往右的观察。
由于轴向只有 340 个切片，因此右侧有黑边。

图 26 – TF_CT_MuscleAndBone 下对冠状面（Coronal Plane）从上到下的观察。
可见心脏、肝脏、乳房轮廓。由于脂肪和空气的 HU 值较低，传递函数的透过性较好，
因此乳后的肺部和气管部位的透明性较好。同样在下侧有黑边。

九、添加光照：使用简化的 Phong 模型与法线计算

虽然我们现在已经可以绘制出 RC 算法对体数据的执行结果，并从不同视点观察。但由于没有光照，渲染结果的立体感、景深感还不够强。因此，本节将讨论如何对体数据添加**光照效果**。

众所周知，光照的实现与三维物体的面片的法向量（法线）是分不开的。但由于我使用的体数据是一系列二维 CT 断层图像在 z 方向的堆叠，而不是通过几何计算构建的三维模型，因此体数据自身没有携带法向量信息。因此，我们需要使用一种方式近似计算体数据某一层次某一点的**法向量——使用梯度**。务必注意，此处使用的是体数据自身参与计算，而不是经传递函数处理后（上色后）的体数据。

首先说明梯度与向量的关系。高等数学和解析几何告诉我们，对于参数方程表达的空间连续曲线：

$$\begin{cases} x = \alpha(t) \\ y = \beta(t) \\ z = \gamma(t) \end{cases}$$

其切向量为$(\alpha'(t), \beta'(t), \gamma'(t))$。

而一组空间连续曲线坐落在的空间平面可用$F(x, y, z) = 0$表示。左右同时求导，有：

$$\frac{\partial F}{\partial x}\alpha'(t) + \frac{\partial F}{\partial y}\beta'(t) + \frac{\partial F}{\partial z}\gamma'(t) = 0$$

上式可改写为向量内积的形式：

$$\left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right) \cdot \left(\alpha'(t), \beta'(t), \gamma'(t)\right) = 0$$

这说明左边向量与右边的切向量垂直，即梯度为法向量。也就是说，**我们可以通过对三维体数据的梯度的计算，来求得某一点的法向量**。

在离散化实现时，我使用采样点的**前后两个点形成的一阶中心差分**来代替梯度：

$$\frac{\partial F}{\partial x} = f(x+1) - f(x-1)$$

则近似为采样点（中点）的法向量。相关代码见 calcNormal 函数。在 x、y、z 三个方向都按上述方法计算后，**可求得三维梯度向量，作为该点的法向量**。

有一些细节问题需要说明：

① 对于边界的点，前序或后序点不存在的，可用采样点替代，近似计算。

② 由于需要计算法向量的点不一定落在整数坐标上，导致前后的点也不一定落在整数坐标上，这需要再次进行插值，计算复杂度太高，且法向量的微小扰动对光照效果没有很大影响。因此，**本实验的做法是将采样点坐标向下取整后再进行上述计算**。

③ 对于一个面，法向量可指向内、外两个方向。而光照计算需要**外法向量**，即应指出面的方向，否则会造成光照计算错误。在目前的成像几何下，上述差分的含义是后序切片减前序前片，即梯度方向而非梯度反方向，故**求得的正是外法向量**。

④ 在光照计算中，求得的法向量应进行**单位化**（normalize），否则会造成光照计算错误。但受 glm 库（float）和体素数值自身（16 位无符号整数）的精度所限，在对部分过短的法向量进行单位化时会产生 NaN 量，此时只好返回未单位化的法向量了。

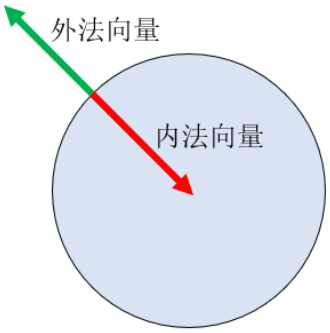


图 27 – 内、外法向量

有了法向量以后，就可以进一步进行光照的计算。本实验采用**一种简化的 Phong 光照模型**[7]。由于镜面反射分量（Specular）与视点有关，为降低复杂度，忽略该分量，这使得光源位置和视点位置无关紧要。对于环境光分量（Ambient），固定光颜色为白色，其权重 KAmbient 从配置文件中读取，默认为 0.6，以让体数据拥有基础亮度。对于漫反射分量（Diffuse），固定光颜色为白色，其权重 KDiffuse 使用 Phong 光照模型的公式 $\max(\langle \mathbf{n}, \mathbf{d} \rangle, 0)$计算（尖括号表示内积，$\mathbf{n}$为采样点的单位法向量，$\mathbf{d}$为光线的单位方向向量）。光照方向固定为$-z$方向。本部分的有关函数是代码中的 applyLighting。对于采样点的颜色的 RGB 分量，应用光照后变为：

$$RGB' = (\text{KDiffuse} \times \text{White} + \text{KAmbient} \times \text{White}) \times RGB$$

应用后需对分量范围裁剪到[0, 1]区间，A 通道保持不变。

当光照启用时，第七节讲述的光线穿越体素的过程中，**在色彩融合前（第二步与第三步之间）应该对采样点颜色进行光照计算与修改**。下面给出几张应用了光照前后的结果对比图（SamplingDelta = 0.5），可以看出，有光照的情况下绘制的结果更具真实感、前后景深感。另外，可见脊柱末端前端向外凸起，所以光照效果发亮（图 B），说明光照计算是正确的：

（因图像尺寸较大，于下页继续）

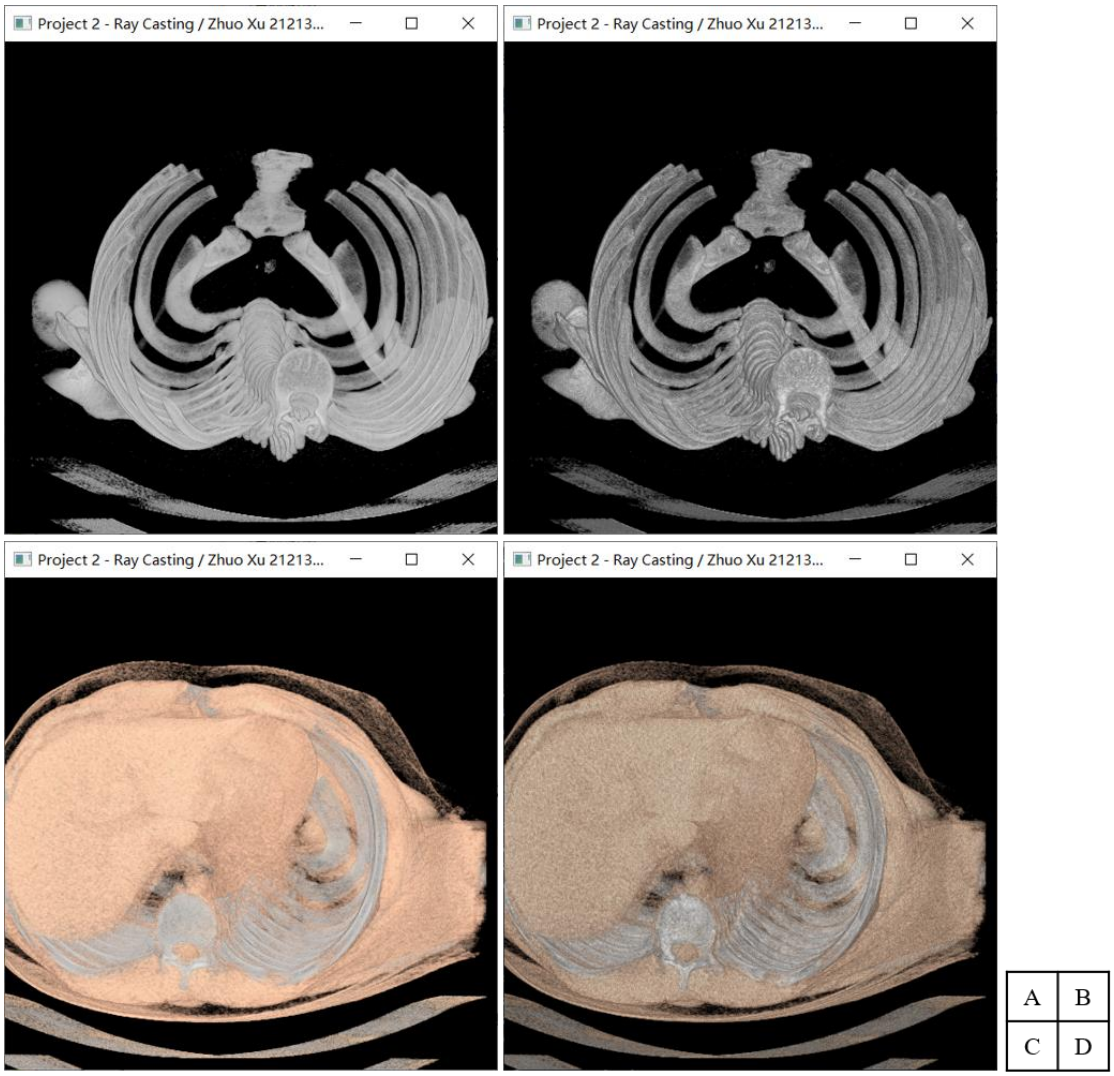


图 28 – 光照关闭与开启情况下的绘制结果比较。A、C 关闭；B、D 开启。

十、多线程并行优化

不难发现，RC 算法中，每条投射出去的光线的运算之间不存在相关性。即每条光线可以独立地射出，并求得视平面对应像素点的颜色值。这说明**光线穿越体素过程可以多线程并行执行**，以加快计算速度。

现代 C++的标准库（STL）为我们提供了操作系统无关的线程 API。使用 std::thread 类即可生成线程对象，调用.join 或.detach 方法即可执行线程；使用 std::mutex 类即可生成互斥锁；使用容器类存储 std::thread 的对象即可构造简单的线程池。为此，本实验进行了多线程优化，按视平面的总行数作为总任务量，N 等分给 N 个线程执行光线穿越体素算法。通过设置配置文件的 MultiThread 属性即可调整使用的线程数。相关函数是代码中的 castAllRays、castSomeRays、castOneRay。

多线程优化对 RC 算法的运行速度提升效果显著。在我的 PC 上（处理器：i5-7300HQ）进行的实验结果如下表所示：

线程数	总光线数	相交光线数	其他配置	耗时
1	262144	262144	SamplingDelta = 1	14 s
4			EnableLighting = false	6 s
8			TransferFunction = "TF_CT_Bone"	5 s

十一、可配置化

本实验对一些关键配置项做了可配置化处理，以增强程序的灵活性与可复用性，避免硬编码导致当需要替换体数据、改变参数时，造成对程序反复编译。配置文件即为前文反复使用的程序目录下的 config.json。我使用 RapidJSON 库来解析该 JSON 文件，读取配置并进行相关初始化。有关函数是代码中的 loadConfigFileAndInitialize。

十二、将视平面渲染到窗口

至此，我们获得了整个视平面的图像，其中各个像素就是对应像素点发出的光线进行 RC 算法后的结果。接下来，我们需要将这张 2D 的图像显示到电脑屏幕上。

OpenGL 提供了一个很简单的用于绘制图像的 API：glDrawPixels。只需要指定图像的宽高、像素格式、图像数组指针即可：

```
glDrawPixels(ImagePlaneWidth, ImagePlaneHeight, GL_RGBA, GL_FLOAT, imagePlane);
```

但根据 OpenGL 4 的官方文档（https://www.khronos.org/registry/OpenGL-Refpages/gl4/），该 API 已经被废除，不再保证后续支持。

本实验中，我使用绘制“背景图片”的方式，**将视平面图像作为一个贴图材质（texture）送入 GPU，然后使用** glDrawArrays **在将材质贴到窗口上**。

贴图发送方面，使用自己设计的 ReNow 框架的 createTexture2D 函数即可创建一个新的 2D 材质并获取材质编号。内部流程为：①使用 glTexParameteri 设置一系列 2D 材质的基本配置，包括插值方式等；②使用 glGenTextures 创建新的 Texture Object，激活之，将其绑定为当前的 GL_TEXTURE_2D；③使用 glTexImage2D 将视平面图像发送到 GPU，对应 Object 为上面产生的；④调用 glGenerateMipmap 生成 Mipmap 以改进不同尺度下的显示效果。上述相关代码在主渲染循环中。

绘制时的具体做法是，定义要画的顶点为$(x, y) = \{(-1,-1),(1,-1),(1,1),(-1,1)\}$（即整个 OpenGL 默认的裁剪空间）；定义材质采样点$(x, y) = \{(0,0),(1,0),(1,1),(0,1)\}$（即整张贴图的左下角开始逆时针一圈）。在着色器方面，构建一个只绘画 XY 平面（z 不妨取 1）的顶点着色器，以及在材质贴图上取样三角面片作为颜色填充的片元着色器，即可将图像绘制到窗口。具体的着色器在 shader 目录下。glDrawArrays 时，应使用 GL_TRIANGLE_FAN（三角扇）绘制方式，以只需使用上述四个顶点送入着色器，绘制出整张视平面。

十三、能不能再快一点儿？

从本节开始，将是课程报告讨论的主体部分。其中的一些讲述使用到的工具不一定

是本次实验的成果，而是一些实现思路或外部工具，在此说明。

RC 算法的时间效率方面不够尽如人意。因为需要跟踪的光线数量通常很大（例如 512×512 视平面要跟踪 26 万条以上），而且每当观察视点发生变化时，就要重新跟踪全体光线，代价很高，也使得自由变化视点观察时的流畅性、体验感很差。因此，对 RC 算法实现时的速度优化是很重要的研究内容。

虽然前面使用了多线程计算进行优化，但绘制速度仍不是让人特别满意。有没有什么办法再提速呢？我们注意到，目前所有计算都在 CPU 中完成，而不像 Project 1 中的光照计算一样在着色器（即 GPU）中完成。**GPU 的通用计算能力要强于 CPU，特别是对于这种大规模的、重复性的、可并发的操作，GPU 可同时调动大量流处理器进行运算。**本实验之所以没有转入 GPU 执行 RC 算法，是因为在 CPU 中执行更便于调试，程序更清晰，且笔者不够熟悉 GPU 相关编程。

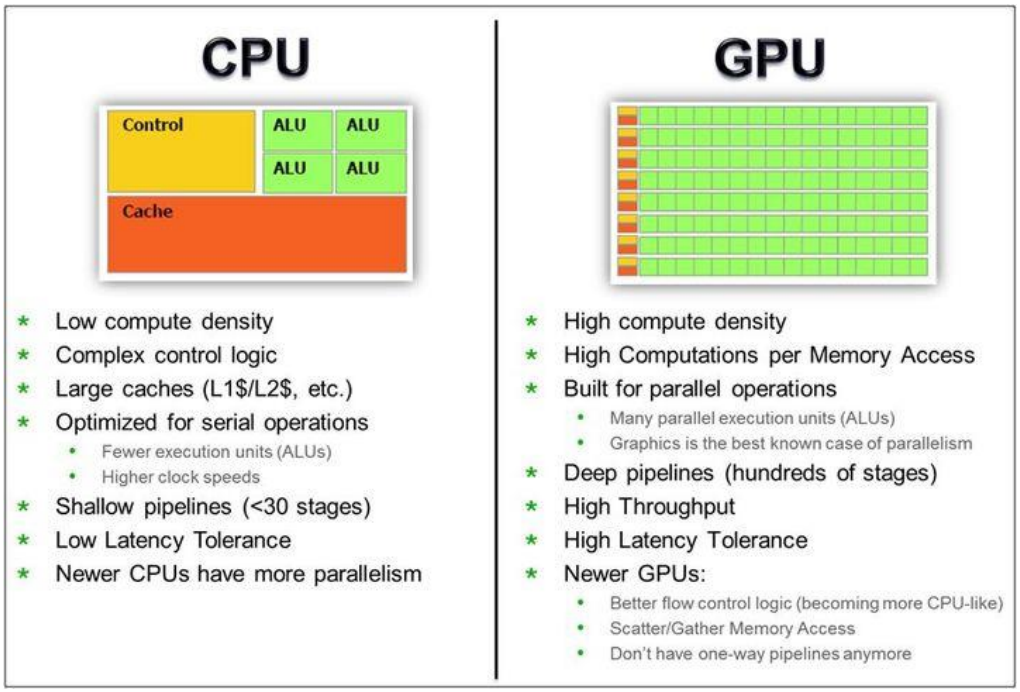


图 29 – CPU **与** GPU **的对比图** [https://www.zhihu.com/question/19903344/answer/711611928]

倘若要将 RC 算法放入 GPU 中进行，以 OpenGL API 为例，应首先将整个体数据作为 3D 材质（tex3D）传递到着色器中，然后将 RC 算法相关逻辑使用着色器语言 GLSL 实现。对于必要的参数，设为 uniform 型变量后从主程序传入。这样，RC 算法将借助 OpenGL 在 GPU 上得到执行，执行结果存放于显存中，可快速绘制到屏幕上。但是，上述流程使用顶点着色器与片元着色器进行计算，**这违反了它们的原始职能**，即不是作为绘制管线的一部分，对传入的顶点进行着色、光栅化等操作，而是在进行一些通用计算操作。

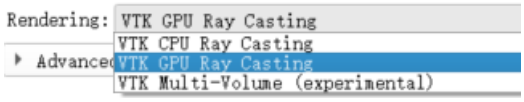


图 30 – 3D Slicer 软件中的 GPU 运算选项

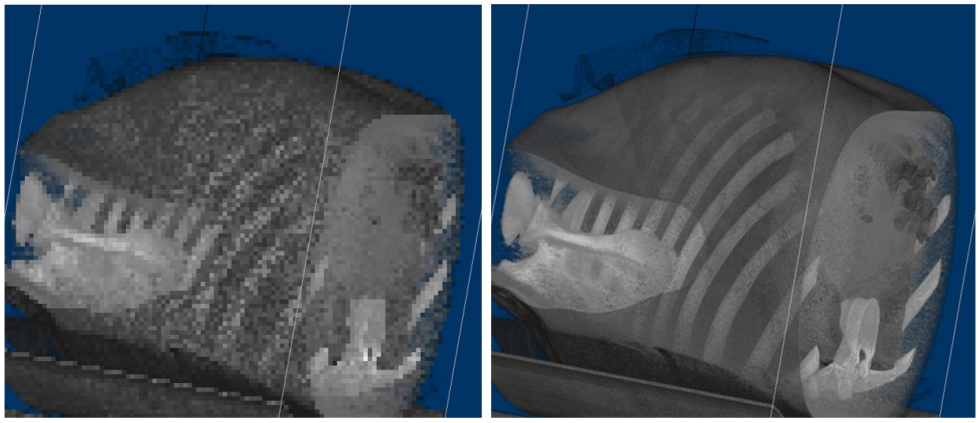
OpenGL 4.3 引入了一个新特性：**计算着色器（Compute Shader）**。它使得借助 OpenGL API 即可进行 GPU 上的通用并行计算，这样便不再需要“hacky”的方法去使用渲染管线上的着色器来进行运算，甚至可以支持 CPU 与 GPU 双侧执行的混合编程。

提到混合编程，就不得不提 NVIDIA 的 **CUDA 技术**。CUDA 语言使得我们可以使用与 C++完全类似的语法进行 GPU 编程，并且可以方便地进行 CPU+GPU 混合编程，简单地在 CPU 与 GPU 之间双向传递数据。以 a+b 程序为例，CUDA 编写的混合编程程序如下[8]：

```
// __global__标识符表示函数将在 GPU 中运行
__global__ void gpuAdd(int d_a, int d_b, int *d_c) {
    *d_c = d_a + d_b;
}
int main(void) {
    int h_c;
    int *d_c;
    // 分配显存的语法与 malloc 一致
    cudaMalloc((void**)&d_c, sizeof(int));
    // 通过函数调用即可在 GPU 中执行，双尖括号中的数值配置并行程度
    gpuAdd < <<1, 1 >> > (1, 4, d_c);
    // 将数据从显存提取到内存
    cudaMemcpy(&h_c, d_c, sizeof(int), cudaMemcpyDeviceToHost);
    printf("1 + 4 = %d\n", h_c);
    // 清理工作
    cudaFree(d_c);
    return 0;
}
```

可见，CUDA 抹平了与硬件交互相关的细节，极大地降低了 GPU 通用计算程序的编写门槛。因此，使用 CUDA 可以在不大规模改动源程序的情况下，实现高速的 RC 算法。

当然，在没有上述技术革新的情况下，我们还有一种最简单的处理方式来加速：降采样、**Coarse-to-Fine**。即先只对视平面上一部分像素进行 RC 算法，对于没有分配到像素值的，使用附近的像素的值进行代替，这样，可以给出大致的体绘制结果，减少用户等待时间。然后后台再对未 Cast 过的光线进行 RC 算法，一段时间后精细化显示结果。科研看图工具 ImageJ [9]的 Volume Viewer 插件就是这样处理的。



coarse　　fine

图 31 – ImageJ Volume Viewer 的绘制速度优化策略

十四、基于直方图的自动传递函数设计

目前，实验所使用的传递函数都是根据 HU 值知识提前设计好的、内置的。那么，对于任一一个体数据，有没有自动设计较为合适的传递函数，以提供最初的基本显示的方法呢？

我们可以根据体数据的灰度（更准确的说法是：值）直方图来自适应地给出一些传递函数。由于有 RGB 三个色通道，以及 A 一个不透明度通道需要调节，所以具体做法有两种进路。

一是固定全局 RGB 基调色（例如灰色），自适应给出 A 通道的分配方法。以前文使用的体数据为例，其值直方图如下（横轴为值，纵轴为相对计数，纵轴量程不定）：

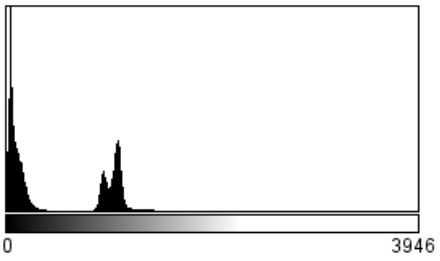


图 32 – lungct 数据的值直方图

不难看出直方图中存在 2（或 3）个峰。假设我们期望看到这些峰位置的所有体素，那么可以给这些峰的位置赋较低的 A 通道值。特别是峰高的，A 通道值应尽可能低，以避免大规模遮蔽其后的部分，从而达到前后通透，尽可能多地显示感兴趣体素。而对于分配的色通道，可按体素值的大小在全灰色灰度范围内按比例分配。一些示意公式和示意图如下：

$$\begin{cases} color_{RGB} = (255,255,255) \times \dfrac{v}{v_{max} - v_{min}} \\ color_A = A_{base} \times (1 - p_v) \end{cases}$$

其中v表示体素值，p_v表示体素值v的出现频率，A_{base}为手动设置的基准不透明度。实际实现时，可对分配的 A 值在体素值方向进行一定的平滑（如移动平滑、中值滤波、小波

去高频等），以避免 A 值的突变，影响绘制效果。

针对上述直方图，ImageJ Volume Viewer 就进行了类似于上述准则的算法下设计的 A 通道，示意图如下所示。横轴表示体素值，黑色波表示体素值的出现频率，橙色曲线表示应用的 A 通道值。可见，对于集中分布的大频率体素值，应用的 A 值较低，其余位置 A 值较高，且 A 值的变化比较平滑。按该传递函数绘制，结果图如图 31 所示。

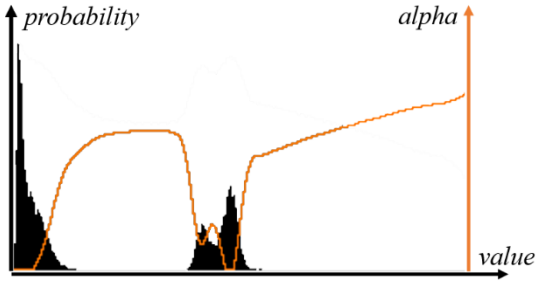


图 33 – ImageJ Volume Viewer 自动设计的传递函数 A 通道

二是预置一组调色板，然后对直方图的各个峰，分配一个颜色。这样，可以得到不同颜色的彩色图像，且有按照值进行分割、染色的效果。

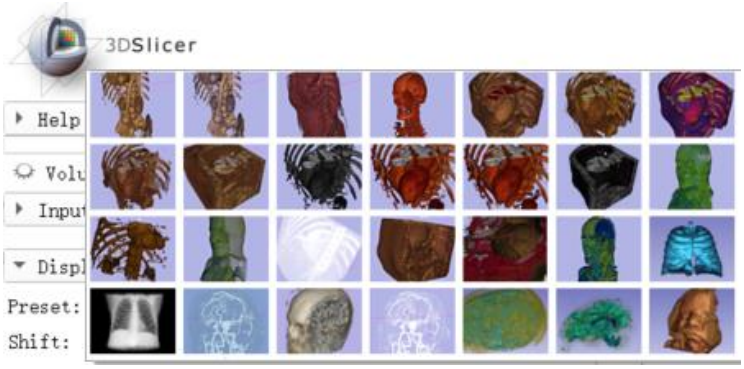


图 34 – 3D Slicer 软件[10]中的传递函数预置

至于如何分辨出直方图有几个峰、峰的位置在哪里，可以使用横向扫描线方法，或纵向扫描线方法（记录先升后降的区间），此处不作具体介绍。

十五、中值滤波

本实验中，在视平面渲染到屏幕上前，允许进行中值滤波来平滑图片，去除噪声。相关函数是 medianFilter(MedianFilterKSize)。以 3×3 尺寸的卷积核为例，将中心元素替换为 9 个元素的中值。示意图如下所示。

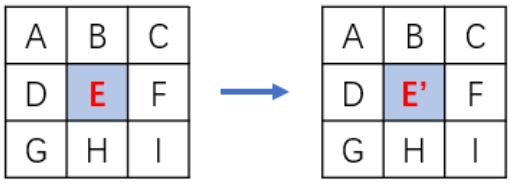


图 35 – 中值滤波示意图

十六、进一步的思考：体绘制——过去，现在与将来

前文，我们详细介绍了光线跟踪（Ray Casting）算法的背景、动机、功能、方法原理，以及实现的全流程与各项细节，并在实验说明、讨论的流程中穿插阐述了对 RC 算法和体绘制的讨论。在全文的最后，我给出一些进一步的思考与总结。

过去直到今天，在能直接进行三维显示的设备取得重大技术突破，大规模普及之前，将三维数据绘制到二维屏幕的技术仍然是不可取代的。相比于直接阅读一系列切片，观察绘制的结果是好理解的多的方式。

如下图所示，是一个 3D DSA（数字血管减影造影术）的体数据的层切片与体绘制结果，右图为我们清晰展示了血管的形态，便于我们发现其中的结、血栓位置。再比如我们前文反复使用的示例体数据，倘若需要诊断肋骨骨折，在层切片中是很难阅读与发现的，而一旦用体绘制技术绘制出来，骨折就会明显地展示在医生面前。

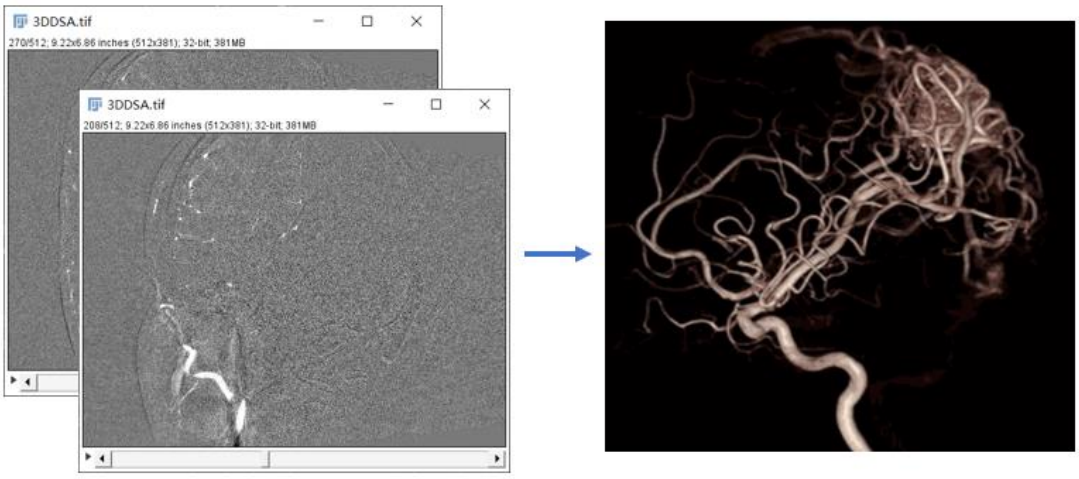


图 36 – 3D DSA **数据体绘制示意图**

[http://www.360doc.com/content/17/1207/00/46184408_710661936.shtml]

今天的体绘制技术在硬件的高速发展和软件的进步下，绘制速度快，绘制效果好，清晰度高。新 OpenGL、CUDA 等技术降低了高性能 GPU 计算程序的绘制门槛，WebGL 等技术甚至能让我们在浏览器下进行体绘制渲染，让人们在手机上、平板电脑、家用电脑上，也能快速进行影像数据查看和疾病诊断。这是非常方便且有意义的进步。

放眼未来，图形学其他方面的进展也可能在体绘制上得到应用，例如经常谈论的真实感绘制技术等。另外，火热的深度学习技术也开始有与体绘制的结合。例如深度学习领域的生成对抗模型（GAN），能够从零生成出与数据集类似的图像，亦或是优化传统体绘制方法的结果。这让人不禁感叹一个领域真是常学常新！共勉！

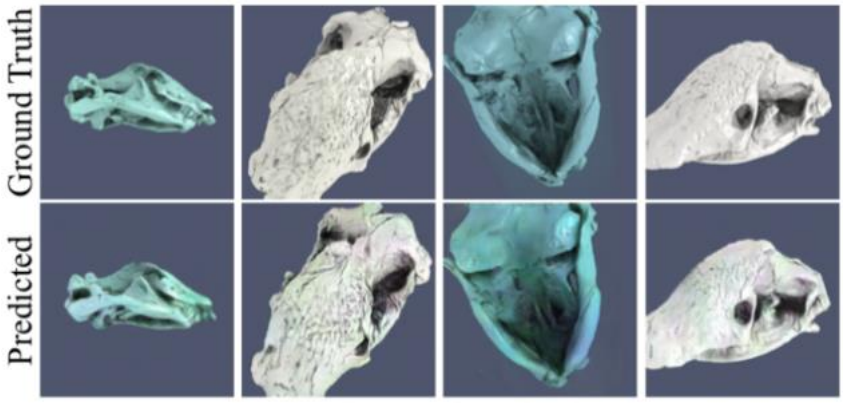


图 37 – 使用传统方法进行的体绘制（上行）和 GAN 预测的体绘制结果（下行）
[https://medium.com/multiple-views-visualization-research-explained/learning-to-volume-render-with-gans-62c0c43e4fb1]

至此，本实验报告和综述报告的全部内容就结束了。下面是附录部分。

附录

参考资料

[1] 罗立民. 《信息可视化技术》课程课件 [EB/OL]. 2021. （内部资料）.

[2] 卓旭, 高钰铭, 沈汉唐. typed-webgl (东南大学《计算机图形学》课程实践作业) [CP/OL]. 2021. https://zxuuu.tech/seucg & https://github.com/z0gSh1u/typed-webgl .

[3] 卓旭. ctdicom2raw [CP/OL]. 2021. https://github.com/z0gSh1u/ctdicom2raw .

[4] AAPM. Low Dose CT Grand Challenge [EB/OL]. 2021. https://www.aapm.org/grandchallenge/lowdosect .

[5] chopper. 射线与包围盒的相交测试 [EB/OL]. 2021. https://zhuanlan.zhihu.com/p/138259656 .

[6] Eric Haines. Essential Ray Tracing Algorithms. An introduction to ray tracing [M]. pp. 33. 1989.

[7] Edward Angel, Dave Shreiner. 交互式计算机图形学：基于 WebGL 的自顶向下方法（第 7 版）[M]. 张荣华 等译. 北京:中国工信出版集团. 2016.

[8] Bhaumik Vaidya. Hands-On GPU-Accelerated Computer Vision with OpenCV and CUDA. Packt. 2018.

[9] ImageJ. ImageJ and Fiji [CP/OL]. 2021. https://imagej.net .

[10] Slicer. 3D Slicer [CP/OL]. 2021. https://www.slicer.org .

其他声明

本报告中的所有外部引用图片均在图片题注中标注来源。对于没有标注的图片，均为笔者自行绘制或程序的执行结果，请规范引用。