

作业 0: 虚拟机的使用

GAMES101, 2020 年春季

教授: 闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

截止时间为北京时间 2020 年 2 月 25 日 (星期二) 上午 10:00

注意:

- 任何更新或更正都将发布在论坛上，因此请偶尔检查一下。
 - 论坛链接<http://games-cn.org/forums/forum/graphics-intro/>
 - 你必须独立完成自己的作业。
 - 你可以在论坛上发布帖子求助，但是发布问题之前，请仔细阅读本文档。
 - 在截止时间之前将你的作业提交到 SmartChair 上。
-

1 虚拟机的使用

为了免去配置作业所需环境的麻烦，本次课程使用虚拟机，学生在虚拟机内编写，编译和运行代码。我们提供的文件为虚拟硬盘文件，使用虚拟机挂载该文件后，就可以保证所有人的环境是统一并且完善的，不需要再手动配置环境。在安装完虚拟机后，我们需要手动安装 Guest Additions 来增强虚拟机的功能。

1.1 安装虚拟机

这里我们使用 **Oracle VM VirtualBox** 虚拟机。

如果你使用 Windows 系统，你可以直接下载<https://download.virtualbox.org/virtualbox/6.1.4/VirtualBox-6.1.4-136177-Win.exe>，下载完成后按照指示完成安装。

如果你使用 Mac OS 系统，你可以直接下载<https://download.virtualbox.org/virtualbox/6.1.4/VirtualBox-6.1.4-136177-OSX.dmg>，下载完成后按照指示完成安装。

如果你使用 Linux 内核的系统，你可以查看https://www.virtualbox.org/wiki/Linux_Downloads，找到你使用的系统，按照对应的指示完成安装。

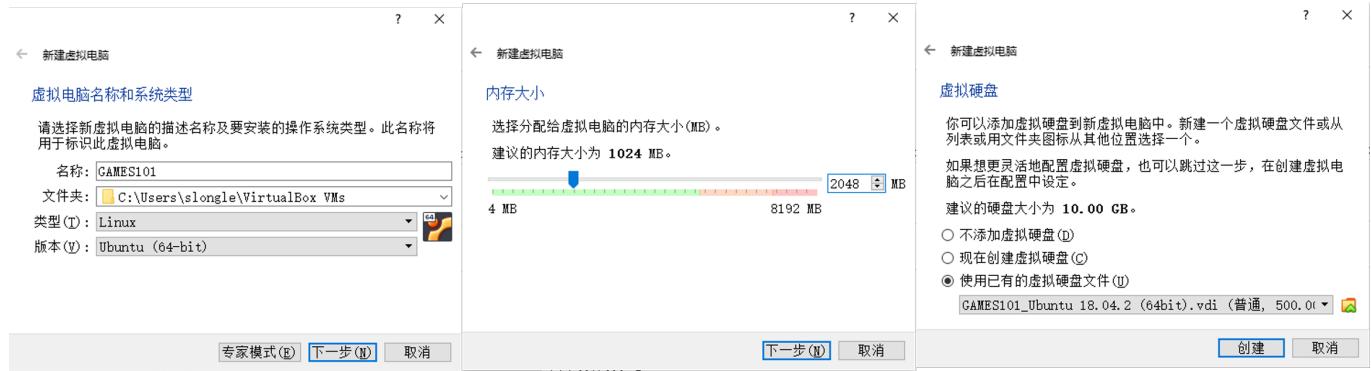
1.2 下载虚拟硬盘

虚拟硬盘文件的下载地址为 <https://cloud.tsinghua.edu.cn/f/103133da1bf8451b8ba6>，密码为 games101。下载完成后得到 **AMES101_Ubuntu 18.04.2 (64bit).rar**，将其解压后得到虚拟硬盘文件 **AMES101_Ubuntu 18.04.2 (64bit).vdi**。

1.3 配置虚拟机

打开 **Virtual Box**，点击新建，设置类型为 **Linux**，版本为 **Ubuntu-64 bit**，建议设置虚拟机的内存大小为 **2GB**，然后选择使用已有的虚拟硬盘文件，设置为

之前解压得到的 **Games101_Ubuntu 18.04.2 (64bit).vdi**, 最后点击创建就完成了虚拟机的配置工作。



之后就可以使用创建好的虚拟机了，选中刚刚创建好的虚拟机，点击右侧上方的启动按钮就可以打开虚拟机了，Ubuntu 系统的密码为 **Ilovegraphics**。

1.4 安装 Guest Additions

进入系统后，点击上方菜单的**设备**，点击**安装增强功能**，如下图所示。安装完成后，重启虚拟机系统就完成了 Guest Additions 的安装。



如果上面的方法安装失败了，可以使用 **ctrl+alt+t** 调出终端，使用如下命令安装 Guest Additions 功能。

```
sudo mkdir -p /media/cdrom  
sudo mount -t auto /dev/cdrom /media/cdrom/  
cd /media/cdrom/  
sudo sh VBoxLinuxAdditions.run
```

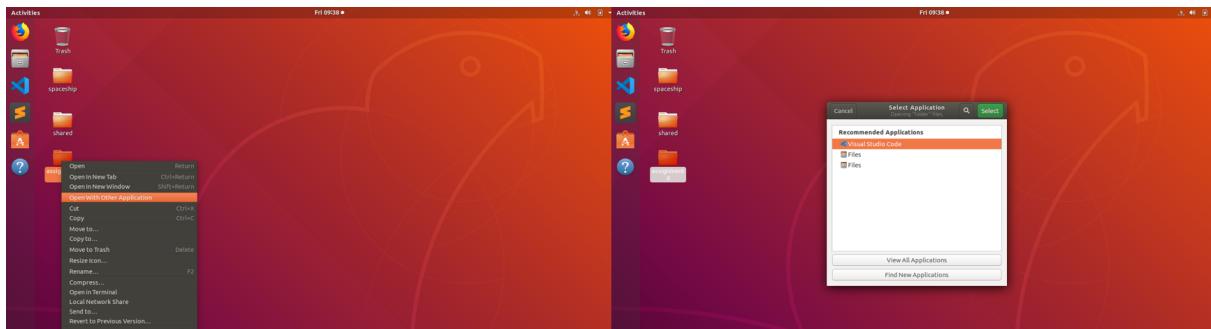
执行完毕后，重启虚拟机系统就完成了 Guest Additions 的安装。

1.5 作业框架的传输及编辑

作业框架的导入和导出有很多种方式，这里只提一种。当你在你的主机上下载好作业框架后，直接将其拖进虚拟机系统里。这里需要开启 Virtual Box 的拖放功能：进入虚拟机系统后，点击上方菜单的设备，将拖放功能设置为双向即可。



导入作业框架后，可以使用 Visual Studio Code 来查看和编辑。右键作业框架的文件夹，选择使用其他应用来打开，选择 Visual Studio Code，具体如下图所示



2 作业框架说明

本部分将体现在样例程序 main.cpp 中。

2.1 开发工具说明

虚拟机中已经自带 Visual Studio Code 与 Sublime 作为文本编辑器，**课程推荐使用 VSCode 编辑代码，并且在命令行中编译、运行程序。**

将框架拷贝到虚拟机中，打开 VSCode，选择 File->Open Folder 找到目标文件夹，选择打开即可。

2.2 C++ 语言使用注意事项

本部分只会简单介绍一些作业涉及的 C++ 语法内容，更多的 C++ 知识请通过 <https://devdocs.io/cpp/> 或者 Stack Overflow 查阅。

2.2.1 头文件

在 C 语言和 C++ 中，头文件或包含文件是一个文件，通常是源代码的形式，由编译器在处理另一个源文件的时候自动包含进来。一般来说，程序员通过编译器指令将头文件包含进其他源文件的开始（或头部）。

实践中，一般通过头文件来引入是现在其他文件中的函数模块。

```
1 #include <cmath>
2 #include <iostream>
```

样例程序中的上述头文件引入了 C++ 中输入、输出、数学计算所需要的必需模块。

2.2.2 函数体

函数体是程序进行指定目标的运算的模块，其中以 main 命名的函数被称为主函数，是程序运行的入口。

```
1 int main(){
2     float a = 1.0, b = 2.0;
3     std::cout << a << std::endl;
4     std::cout << a/b << std::endl;
5     std::cout << std::sqrt(a) << std::endl;
6     std::cout << std::acos(-1) << std::endl;
7     std::cout << std::sin(30.0/180.0*acos(-1)) << std::endl;
8     return 0;
```

上述代码的作用是在三行分别输出 a , $\frac{a}{b}$, \sqrt{a} , $\arccos(-1)$, $\sin(30)$ 的计算结果，并安全退出程序。请观察输出，并尝试解释这些结果（提示：C++ 三角函数运算使用弧度制）。

2.2.3 C++ 常见错误指南

1. Compile Error 编译错误：认真阅读编译器给出的报错信息，找到报错位置修改代码；如果无法自己解决，建议将报错信息拷贝到 Stack Overflow 查找类似情况。
2. undefined reference to xxx：一般是链接错误，检查 CMakeLists.txt 中是否包括了需要引入的模块。
3. Segmentation Fault：段错误，一般是数组越界、栈空间开销过大等问题导致。
4. Bus Error：总线错误，成因一般与段错误相似。
5. Math Error：一般是除数为 0 导致。

2.3 Eigen 库使用注意事项

Eigen 是本课程使用的线性代数运算库，官方文档为 <http://eigen.tuxfamily.org>。

2.3.1 头文件

如样例程序 main.cpp 所示，eigen 需要额外引入头文件 `<eigen3/Eigen/Core>`。

```
1 #include<eigen3/Eigen/Core>
```

2.3.2 向量、矩阵

关于本部分内容，请详细阅读官方文档的矩阵部分https://eigen.tuxfamily.org/dox/group__TutorialMatrixArithmetic.html以获得更全面、清晰的理解。

```

1 // Example of vector
2 std::cout << "Example of vector \n";
3 // vector definition
4 Eigen::Vector3f v(1.0f,2.0f,3.0f);
5 Eigen::Vector3f w(1.0f,0.0f,0.0f);
6 // vector output
7 std::cout << "Example of output \n";
8 std::cout << v << std::endl;
9 // vector add
10 std::cout << "Example of add \n";
11 std::cout << v + w << std::endl;
12 // vector scalar multiply
13 std::cout << "Example of scalar multiply \n";
14 std::cout << v * 3.0f << std::endl;
15 std::cout << 2.0f * v << std::endl;

```

上述关于 Vector 的使用样例展示了如何定义一个三维浮点向量并且进行输出、加减、数乘，请自行根据数乘的形式与向量点积的形式探索点积的用法。

```

1 // Example of matrix
2 std::cout << "Example of matrix \n";
3 // matrix definition
4 Eigen::Matrix3f i,j;
5 i << 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0;
6 j << 2.0, 3.0, 1.0, 4.0, 6.0, 5.0, 9.0, 7.0, 8.0;
7 // matrix output
8 std::cout << "Example of output \n";
9 std::cout << i << std::endl;
10 // matrix add i + j
11 // matrix scalar multiply i * 2.0
12 // matrix multiply i * j
13 // matrix multiply vector i * v

```

上述关于 Matrix 的使用样例展示了如何定一个三维浮点矩阵进行输出，请自行根据注释与 Vector 部分的经验探索矩阵加减、数乘、矩阵乘法、矩阵乘向量的用法。

3 作业描述与提交

3.1 作业描述

给定一个点 $P=(2,1)$, 将该点绕原点先逆时针旋转 45° , 再平移 $(1,2)$, 计算出变换后点的坐标 (要求用齐次坐标进行计算)。

3.2 编译

为方便之后的作业编写, 本次作业要求使用 `cmake` 进行编译。

首先, 编写好本次作业的程序 `main.cpp`。

然后, 在 `main.cpp` 所在目录下, 打开终端 (命令行), 依次输入:

- `mkdir build`: 创建名为 `build` 的文件夹。
- `cd build`: 移动到 `build` 文件夹下。
- `cmake ..`: 注意其中'.'表示上一级目录, 若为".."则表示当前目录。
- `make`: 编译程序, 错误提示会显示在终端中。
- `./Transformation`: 若上一步无错误, 则可运行程序 (这里的 Transformation 为可执行文件名, 可参照 `CMakeLists.txt` 中修改)。

3.3 提交

作业提交使用的平台为 **Smartchair** 平台, 地址为 <http://www.smartchair.org/GAMES2020Course-YLQ>, 平台的具体操作说明请在 http://games-cn.org/submit_homework/ 下载。

3.4 评分

由于本次作业主要目的是让同学们熟悉虚拟机的使用、使用 C++ 与 **Eigen** 库编写简单的程序和用 `cmake` 进行编译, 所以本次作业不进行评分。同学们只需将编写好的程序打包提交即可, 顺便可以熟悉 **Smartchair** 平台提交作业的流程。

作业 1: 旋转与投影

GAMES101, 2021 年春季

教授: 闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

发布日期为北京时间 2021 年 6 月 8 日（星期二）上午 **00:00**

截止日期为北京时间 2021 年 6 月 18 日（星期五）晚上 **23:59**

注意:

- 任何更新或更正都将发布在论坛上，因此请偶尔检查一下。
 - 论坛链接: <http://games-cn.org/forums/forum/graphics-intro/>。
 - 你必须独立完成自己的作业。
 - 你可以在论坛上发布帖子求助，但是发布问题之前，请仔细阅读本文档。
 - 在截止时间之前将你的作业提交到 SmartChair 上。
-

1 总览

到目前为止，我们已经学习了如何使用矩阵变换来排列二维或三维空间中的对象。所以现在是时候通过实现一些简单的变换矩阵来获得一些实际经验了。在接下来的三次作业中，我们将要求你去模拟一个基于 CPU 的光栅化渲染器的简化版本。

本次作业的任务是填写一个旋转矩阵和一个透视投影矩阵。给定三维下三个点 $v_0(2.0, 0.0, -2.0)$, $v_1(0.0, 2.0, -2.0)$, $v_2(-2.0, 0.0, -2.0)$, 你需要将这三个点的坐标变换为屏幕坐标，并在屏幕上绘制出对应的线框三角形（在代码框架中，我们已经提供了 `draw_triangle` 函数，所以你只需要去构建变换矩阵即可）。简而言之，我们需要进行模型、视图、投影、视口等变换来将三角形显示在屏幕上。在提供的代码框架中，我们留下了模型变换和投影变换的部分给你去完成。

如果你对上述概念有任何不清楚或疑问，请复习课堂笔记或在论坛上提问。

以下是你需要在 `main.cpp` 中修改的函数（请不要修改任何函数名和其他已经填写好的函数，并保证提交的代码是能编译运行的）：

- `get_model_matrix(float rotation_angle)`: 逐个元素地构建模型变换矩阵并返回该矩阵。在此函数中，你只需要实现三维中绕 z 轴旋转的变换矩阵，而不用处理平移与缩放。
- `get_projection_matrix(float eye_fov, float aspect_ratio, float zNear, float zFar)`: 使用给定的参数逐个元素地构建透视投影矩阵并返回该矩阵。
- [Optional] `main()`: 自行补充你所需的其他操作。

当你在上述函数中正确地构建了模型与投影矩阵，光栅化器会创建一个窗口显示出线框三角形。由于光栅化器是逐帧渲染与绘制的，所以你可以使用 **A** 和 **D** 键将该三角形绕 z 轴旋转（此处有一项提高作业，实现将三角形绕任意过原点的轴旋转）。当你按下 **Esc** 键时，窗口会关闭且程序终止。

另外，你也可以从命令行中运行该程序。你可以使用以下命令来运行和传递旋转角给程序。注意，在这样的运行方式下，是不会生成任何窗口的，输出的结果图像会被存储在给定的文件中（若未指定文件名，则默认存储在 **output.png** 中）。图像的存储位置在可执行文件旁，所以如果你的可执行文件是在 **build** 文件夹中，那么图像也会在该文件夹内。

命令行的使用方法如下：

```
1 ./Rasterizer //循环运行程序，创建一个窗口显示，且你可以  
2 //使用A键和D键旋转三角形。  
3  
4 ./Rasterizer -r 20 //运行程序并将三角形旋转20度，  
5 //然后将结果存在output.png中  
6  
7 ./Rasterizer -r 20 image.png //运行程序并将三角形旋转20度，  
8 //然后将结果存在image.png中。
```

2 代码框架

在本次作业中，因为你并不需要去使用三角形类，所以你需要理解与修改的文件为：**rasterizer.hpp** 和 **main.cpp**。其中 **rasterizer.hpp** 文件作用是生成渲染器界面与绘制。

光栅化器类在该程序系统中起着重要的作用，其成员变量与函数如下。

成员变量：

- `Matrix4f model, view, projection`: 三个变换矩阵。
- `vector<Vector3f> frame_buf`: 帧缓冲对象，用于存储需要在屏幕上绘制的颜色数据。

成员函数：

- `set_model(const Eigen::Matrix4f& m)`: 将内部的模型矩阵作为参数传递给光栅化器。
- `set_view(const Eigen::Matrix4f& v)`: 将视图变换矩阵设为内部视图矩阵。
- `set_projection(const Eigen::Matrix4f& p)`: 将内部的投影矩阵设为给定矩阵 p，并传递给光栅化器。
- `set_pixel(Vector2f point, Vector3f color)`: 将屏幕像素点 (x, y) 设为 (r, g, b) 的颜色，并写入相应的帧缓冲区位置。

在 `main.cpp` 中，我们模拟了图形管线。我们首先定义了光栅化器类的实例，然后设置了其必要的变量。然后我们得到一个带有三个顶点的硬编码三角形（请不要修改它）。在主函数上，我们定义了三个分别计算模型、视图和投影矩阵的函数，每一个函数都会返回相应的矩阵。接着，这三个函数的返回值会被 `set_model()`, `set_view()` 和 `set_projection()` 三个函数传入光栅化器中。最后，光栅化器在屏幕上显示出变换的结果。

在用模型、视图、投影矩阵对给定几何体进行变换后，我们得到三个顶点的正则化空间坐标 (canonical space coordinate)。正则化空间坐标是由三个取值范围

在 [-1,1] 之间的 x, y, z 坐标构成。我们下一步需要做的就是视口变换，将坐标映射到我们的屏幕中 (`window_width * window_height`)，这些在光栅化器中都已完成，所以不需要担心。但是，你需要去理解这步操作是如何运作的，这一点十分重要。

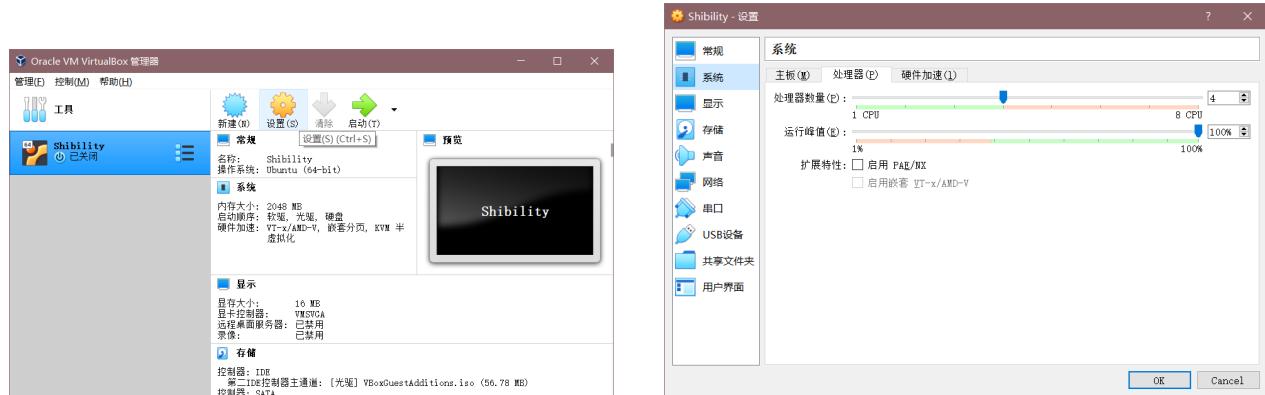
3 编译

如果使用自己的系统，本次程序作业中使用到的库为 Eigen 与 OpenCV，请确保这两个库的配置正确。

如果使用虚拟机并用 CMake 进行编译，请在终端（命令行）输入以下内容：

```
1 mkdir build      // 创建build文件夹以保留的工程文件。
2 cd build        // 进入build文件夹。
3 cmake ..         // 通过提供CMakeLists.txt文件的路径
                  // 作为参数来运行CMake。
4 make -j4        // 通过make编译代码，-j4 表示通过4个
                  // 内核进行并行化编译。
5 ./Rasterizer    // 运行代码。
```

如果没有对虚拟机的核心数进行过设置，可以在 Virtual Box 里点击“设置”，在弹出的窗口中选择“系统—处理器”，然后就可以设置虚拟机的核心数了。具体操作可见下图：



4 评分与提交

评分：

每次作业的评分，分为基础与提高两部分，即在作业批改时会给大家反馈两个成绩。由于作业不是强制要求必须提交，所以在完成全部作业后，我们会统计所有的基础分数。若基础分数及格则视为通过课程，反之视为不通过。

- [5 分] 正确构建模型矩阵。
- [5 分] 正确构建透视投影矩阵。
- [10 分] 你的代码可以在现有框架下正确运行，并能看到变换后的三角形。
- [10 分] 当按 **A** 键与 **D** 键时，三角形能正确旋转。或者正确使用命令行得到旋转结果图像。
- [提高项 5 分] 在 `main.cpp` 中构造一个函数，该函数的作用是得到绕任意过原点的轴的旋转变换矩阵。

```
Eigen::Matrix4f get_rotation(Vector3f axis, float angle)
```

- [-2 分] 惩罚分数：

未删除 `/build`、`/.vscode`、`Assignment1.pdf` 等与代码无关的文件。

未提交或未按要求完成 `README.md`。

代码相关文件和 `README.md` 文件不在你提交的文件夹下的第一层。

提交：

当你完成作业后，请清理你的项目，记得在你的文件夹中包含 `CMakeLists.txt` 和所有的程序文件（无论是否修改）。同时，请新建一个 `README.md` 文件

写清楚自己完成了上述得分点中的哪几点。最后，将上述内容打包，并用“姓名_Homework1.zip”的命名方式提交到 SmartChair 平台。

平台链接：<http://www.smartchair.org/GAMES101-Spring2021/>

作业 2: Triangles and Z-buffering

GAMES101, 2021 年春季

教授: 闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

发布日期为北京时间 2021 年 6 月 19 日 (星期六) 上午 10:00

截止日期为北京时间 2021 年 6 月 29 日 (星期二) 上午 10:00

注意:

- 任何更新或更正都将发布在论坛上，因此请偶尔检查一下。
 - 论坛链接: <http://games-cn.org/forums/forum/graphics-intro/>。
 - 你必须独立完成自己的作业。
 - 你可以在论坛上发布帖子求助，但是发布问题之前，请仔细阅读本文档。
 - 在截止时间之前将你的作业提交到 SmartChair 上。
-

1 总览

在上次作业中，虽然我们在屏幕上画出一个线框三角形，但这看起来并不是那么的有趣。所以这一次我们继续推进一步——在屏幕上画出一个实心三角形，换言之，栅格化一个三角形。上一次作业中，在视口变化之后，我们调用了函数 `rasterize_wireframe(const Triangle& t)`。但这一次，你需要自己填写并调用函数 `rasterize_triangle(const Triangle& t)`。

该函数的内部工作流程如下：

1. 创建三角形的 2 维 bounding box。
2. 遍历此 bounding box 内的所有像素（使用其整数索引）。然后，使用像素中心的屏幕空间坐标来检查中心点是否在三角形内。
3. 如果在内部，则将其位置处的**插值深度值** (interpolated depth value) 与深度缓冲区 (depth buffer) 中的相应值进行比较。
4. 如果当前点更靠近相机，请设置像素颜色并更新深度缓冲区 (depth buffer)。

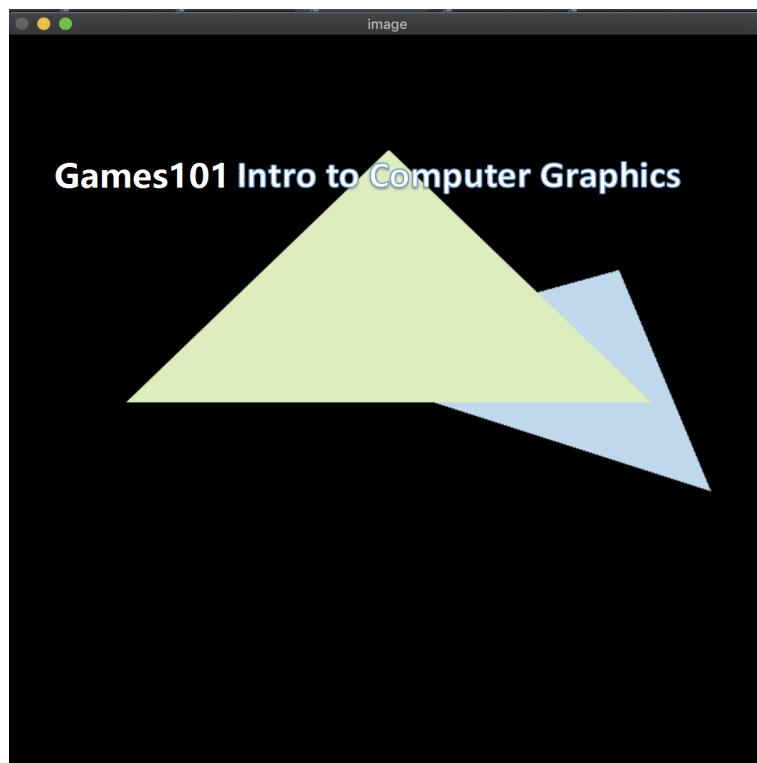
你需要修改的函数如下：

- `rasterize_triangle()`: 执行三角形栅格化算法
- `static bool insideTriangle()`: 测试点是否在三角形内。你可以修改此函数的定义，这意味着，你可以按照自己的方式更新返回类型或函数参数。

因为我们只知道三角形三个顶点处的深度值，所以对于三角形内部的像素，我们需要用**插值**的方法得到其深度值。我们已经为你处理好了这一部分，因为有关这方面的内容尚未在课程中涉及。插值的深度值被储存在变量 `z_interpolated` 中。

请注意我们是如何初始化 depth buffer 和注意 z values 的符号。为了方便同学们写代码，我们将 z 进行了反转，保证都是正数，并且越大表示离视点越远。

在此次作业中，你无需处理旋转变换，只需为模型变换返回一个单位矩阵。最后，我们提供了两个 hard-coded 三角形来测试你的实现，如果程序实现正确，你将看到如下所示的输出图像：



2 开始编写

在你自己的计算机或虚拟机上下载并使用我们更新的框架代码。你会注意到，在 `main.cpp` 下的 `get_projection_matrix()` 函数是空的。请复制粘贴你在第一次作业中的实现来填充该函数。

3 评分与提交

评分：

- [5 分] 正确地提交所有必须的文件，且代码能够编译运行。
- [20 分] 正确实现三角形栅格化算法。
- [10 分] 正确测试点是否在三角形内。
- [10 分] 正确实现 `z-buffer` 算法，将三角形按顺序画在屏幕上。
- [提高项 5 分] 用 `super-sampling` 处理 Anti-aliasing：你可能会注意到，当我们放大图像时，图像边缘会有锯齿感。我们可以用 `super-sampling` 来解决这个问题，即对每个像素进行 $2 * 2$ 采样，并比较前后的结果（这里并不需要考虑像素与像素间的样本复用）。需要注意的点有，对于像素内的每一个样本都需要维护它自己的深度值，即每一个像素都需要维护一个 `sample list`。最后，如果你实现正确的话，你得到的三角形不应该有不正常的黑边。
- [-2 分] 惩罚分数：
 - 未删除 `/build`、`/.vscode`、`Assignment2.pdf` 等与代码无关的文件。
 - 未提交或未按要求完成 `README.md`。
 - 代码相关文件和 `README.md` 文件不在你提交的文件夹下的第一层。

提交：

当你完成作业后，请清理你的项目，记得在你的文件夹中包含 `CMakeLists.txt` 和所有的程序文件（无论是否修改）。同时，请提交一份实验结果的图片与添加一个 `README.md` 文件写下是否完成提高题（如果完成了，也请同时提交一份结果图片），并简要描述你在各个函数中实现的功能。最后，将上述内容打包，并用“姓名_Homework2.zip”的命名方式提交到 SmartChair 平台。

平台链接：<http://www.smartchair.org/GAMES101-Spring2021/>。

作业 3: Pipeline and Shading

GAMES101, 2021 年春季

教授: 闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

发布日期为北京时间 2021 年 6 月 30 日 (星期三)**15:00**

截止日期为北京时间 2021 年 7 月 10 日 (星期六)**23:59**

注意:

- 任何更新或更正都将发布在论坛上，因此请偶尔检查一下。
 - 论坛链接: <http://games-cn.org/forums/forum/graphics-intro/>。
 - 你必须独立完成自己的作业。
 - 你可以在论坛上发布帖子求助，但是发布问题之前，请仔细阅读本文档。
 - 在截止时间之前将你的作业提交到 SmartChair 上。
-

1 总览

在这次编程任务中，我们会进一步模拟现代图形技术。我们在代码中添加了 Object Loader(用于加载三维模型), Vertex Shader 与 Fragment Shader，并且支持了纹理映射。

而在本次实验中，你需要完成的任务是：

1. 修改函数 `rasterize_triangle(const Triangle& t)` in `rasterizer.cpp`: 在此处实现与作业 2 类似的插值算法，实现法向量、颜色、纹理颜色的插值。
2. 修改函数 `get_projection_matrix()` in `main.cpp`: 将你自己在之前的实验中实现的投影矩阵填到此处，此时你可以运行 `./Rasterizer output.png normal` 来观察法向量实现结果。
3. 修改函数 `phong_fragment_shader()` in `main.cpp`: 实现 Blinn-Phong 模型计算 Fragment Color.
4. 修改函数 `texture_fragment_shader()` in `main.cpp`: 在实现 Blinn-Phong 的基础上，将纹理颜色视为公式中的 `kd`，实现 Texture Shading Fragment Shader.
5. 修改函数 `bump_fragment_shader()` in `main.cpp`: 在实现 Blinn-Phong 的基础上，仔细阅读该函数中的注释，实现 Bump mapping.
6. 修改函数 `displacement_fragment_shader()` in `main.cpp`: 在实现 Bump mapping 的基础上，实现 displacement mapping.

2 开始编写

2.1 编译与使用

在课程提供的虚拟机上，下载本次实验的基础代码之后，请在 SoftwareRasterizer 目录下按照如下方式构建程序：

```
1 $ mkdir build  
2 $ cd ./build  
3 $ cmake ..  
4 $ make
```

这将会生成命名为 `Rasterizer` 的可执行文件。使用该可执行文件时，你传入的第二个参数将会是生成的图片文件名，而第三个参数可以是如下内容：

- `texture`: 使用代码中的 `texture` shader.

使用举例: `./Rasterizer output.png texture`

- `normal`: 使用代码中的 `normal` shader.

使用举例: `./Rasterizer output.png normal`

- `phong`: 使用代码中的 `blinn-phong` shader.

使用举例: `./Rasterizer output.png phong`

- `bump`: 使用代码中的 `bump` shader.

使用举例: `./Rasterizer output.png bump`

- `displacement`: 使用代码中的 `displacement` shader.

使用举例: `./Rasterizer output.png displacement`

当你修改代码之后，你需要重新 `make` 才能看到新的结果。

2.2 框架代码说明

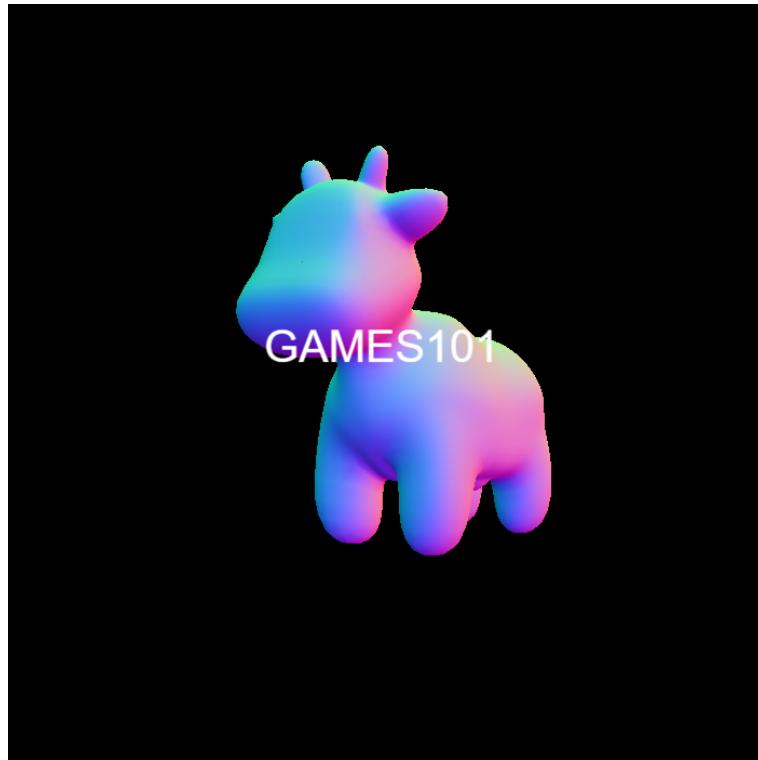
相比上次实验，我们对框架进行了如下修改：

1. 我们引入了一个第三方 `.obj` 文件加载库来读取更加复杂的模型文件，这部分库文件在 `OBJ_Loader.h` file. 你无需详细理解它的工作原理，只需知道这个库将会传递给我们一个被命名被 `TriangleList` 的 `Vector`，其中每个三角形都有对应的点法向量与纹理坐标。此外，与模型相关的纹理也将被一同加载。
注意：如果你想尝试加载其他模型，你目前只能手动修改模型路径。

2. 我们引入了一个新的 Texture 类以从图片生成纹理，并且提供了查找纹理颜色的接口：Vector3f getColor(float u, float v)
3. 我们创建了 Shader.hpp 头文件并定义了 fragment_shader_payload，其中包括了 Fragment Shader 可能用到的参数。目前 main.cpp 中有三个 Fragment Shader，其中 fragment_shader 是按照法向量上色的样例 Shader，其余两个将由你来实现。
4. 主渲染流水线开始于 `rasterizer::draw(std::vector<Triangle> &TriangleList)`。我们再次进行一系列变换，这些变换一般由 Vertex Shader 完成。在此之后，我们调用函数 `rasterize_triangle`。
5. `rasterize_triangle` 函数与你在作业 2 中实现的内容相似。不同之处在于被设定的数值将不再是常数，而是按照 Barycentric Coordinates 对法向量、颜色、纹理颜色与底纹颜色 (Shading Colors) 进行插值。回忆我们上次为了计算 z value 而提供的 [alpha, beta, gamma]，这次你将需要将其应用在其他参数的插值上。你需要做的是计算插值后的颜色，并将 Fragment Shader 计算得到的颜色写入 framebuffer，这要求你首先使用插值得到的结果设置 fragment shader payload，并调用 fragment shader 得到计算结果。

2.3 运行与结果

在你按照上述说明将上次作业的代码复制到对应位置，并作出相应修改之后（**请务必认真阅读说明**），你就可以运行默认的 normal shader 并观察到如下结果：



实现 Blinn-Phong 反射模型之后的结果应该是：



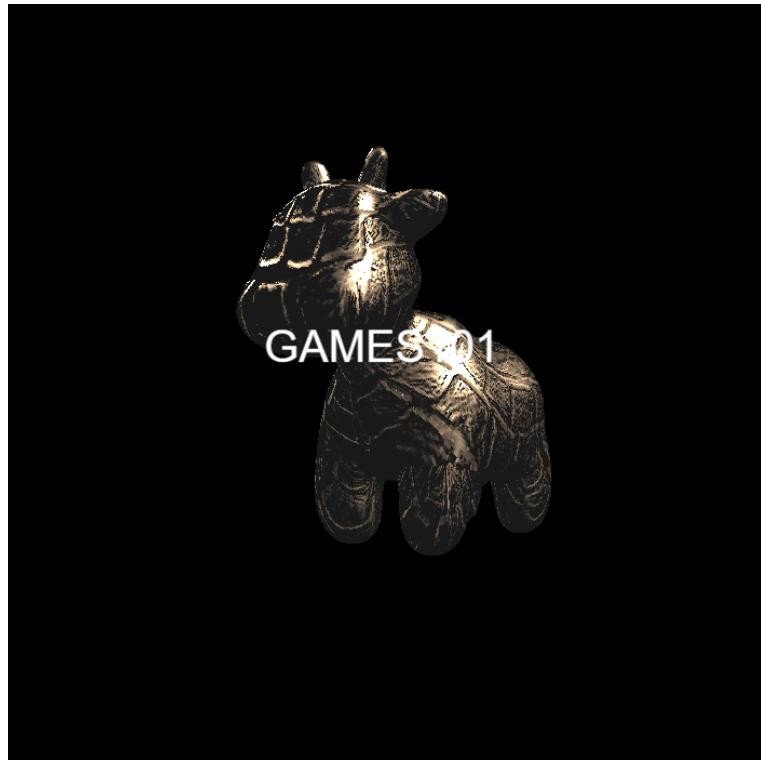
实现纹理之后的结果应该是：



实现 Bump Mapping 后，你将看到可视化的凹凸向量：



实现 Displacement Mapping 后，你将看到如下结果：



3 评分与提交

评分:

- [5 分] 提交格式正确，包括所有需要的文件。代码可以正常编译、执行。
- [10 分] 参数插值：正确插值颜色、法向量、纹理坐标、位置 (Shading Position) 并将它们传递给 `fragment_shader_payload`.
- [20 分] Blinn-phong 反射模型：正确实现 `phong_fragment_shader` 对应的反射模型。
- [5 分] Texture mapping: 将 `phong_fragment_shader` 的代码拷贝到 `texture_fragment_shader`，在此基础上正确实现 Texture Mapping.
- [10 分] Bump mapping 与 Displacement mapping: 正确实现 Bump mapping 与 Displacement mapping.

- [Bonus 3 分] 尝试更多模型：找到其他可用的 .obj 文件，提交渲染结果并把模型保存在 /models 目录下。这些模型也应该包含 **Vertex Normal** 信息。
- [Bonus 5 分] 双线性纹理插值：使用双线性插值进行纹理采样，在 Texture 类中实现一个新方法 `Vector3f getColorBilinear(float u, float v)` 并通过 `fragment shader` 调用它。为了使双线性插值的效果更加明显，你应该考虑选择更小的纹理图。请同时提交纹理插值与双线性纹理插值的结果，并进行比较。
- [-2 分] 惩罚分数：
 - 未删除 /build、/.vscode、Assignment3.pdf 等与代码无关的文件；
 - 未提交或未按要求完成 README.md；
 - 未按照要求完成/images 目录；
 - 代码相关文件和 README.md 文件不在你提交的文件夹下的第一层。

提交：

1. 当你完成作业后，请清理你的项目（删去：/build、/.vscode、/Assignment3.pdf 等），在你的文件夹中包含 CMakeLists.txt 和所有的程序文件（无论是否修改）；
2. 同时，请新建一个 /images 目录，将所有实验结果图片保存在该目录下；
3. 再添加一个 README.md 文件写清楚自己完成了以上七个得分点中的哪几点（如果完成了，也请同时提交一份结果图片），并简要描述你在各个函数中实现的功能；
4. 最后，将上述内容打包，并用“姓名_Homework3.zip”的命名方式提交到 SmartChair 平台。

平台链接：[http://www.smartchair.org/GAMES101-Spring2021/。](http://www.smartchair.org/GAMES101-Spring2021/)

作业 4: Bézier 曲线

GAMES101, 2021 年春季

教授: 闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

发布日期为北京时间 2021 年 7 月 12 日 (星期一) 上午**10: 00**

截止日期为北京时间 2021 年 7 月 22 日 (星期四) 上午**10: 00**

注意:

- 任何更新或更正都将发布在论坛上, 因此请偶尔检查一下。
 - 论坛链接: <http://games-cn.org/forums/forum/graphics-intro/>。
 - 你必须独立完成自己的作业。
 - 你可以在论坛上发布帖子求助, 但是发布问题之前, 请仔细阅读本文档。
 - 在截止时间之前将你的作业提交到 SmartChair 上。
-

1 总览

Bézier 曲线是一种用于计算机图形学的参数曲线。在本次作业中，你需要实现 **de Casteljau** 算法来绘制由 4 个控制点表示的 Bézier 曲线（当你正确实现该算法时，你可以支持绘制由更多点来控制的 Bézier 曲线）。

你需要修改的函数在提供的 **main.cpp** 文件中。

- **bezier**: 该函数实现绘制 Bézier 曲线的功能。它使用一个控制点序列和一个 OpenCV:: Mat 对象作为输入，没有返回值。它会使 t 在 0 到 1 的范围内进行迭代，并在每次迭代中使 t 增加一个微小值。对于每个需要计算的 t ，将调用另一个函数 **recursive_bezier**，然后该函数将返回在 Bézier 曲线上 t 处的点。最后，将返回的点绘制在 OpenCV :: Mat 对象上。
- **recursive_bezier**: 该函数使用一个控制点序列和一个浮点数 t 作为输入，实现 **de Casteljau** 算法来返回 Bézier 曲线上对应点的坐标。

2 算法

De Casteljau 算法说明如下：

1. 考虑一个 p_0, p_1, \dots, p_n 为控制点序列的 Bézier 曲线。首先，将相邻的点连接起来以形成线段。
2. 用 $t : (1 - t)$ 的比例细分每个线段，并找到该分割点。
3. 得到的分割点作为新的控制点序列，新序列的长度会减少一。
4. 如果序列只包含一个点，则返回该点并终止。否则，使用新的控制点序列并转到步骤 1。

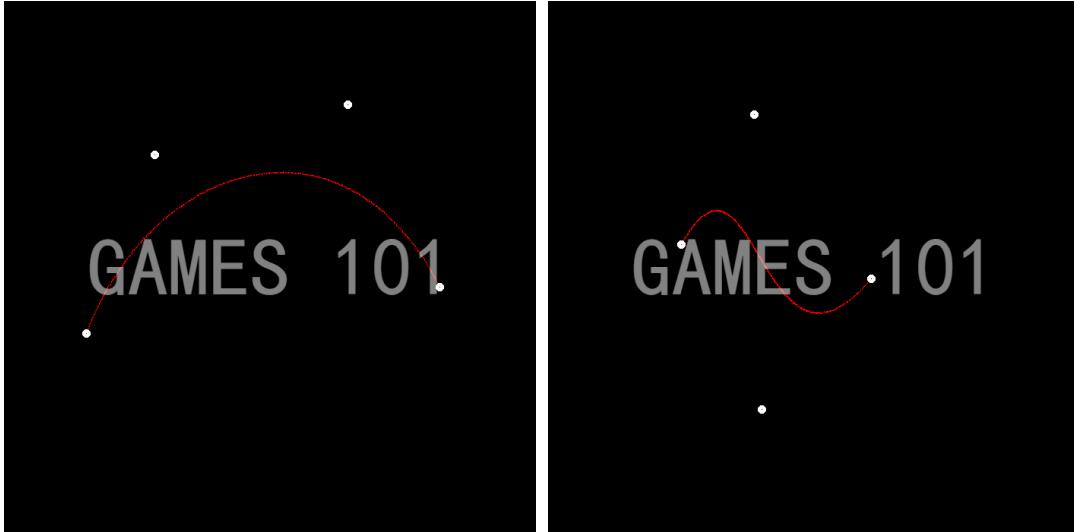
使用 $[0,1]$ 中的多个不同的 t 来执行上述算法，你就能得到相应的 Bézier 曲线。

3 开始编写

在本次作业中，你会在一个新的代码框架上编写，它比以前的代码框架小很多。和之前作业相似的是，你可以选择在自己电脑的系统或者虚拟机上完成作业。请下载项目的框架代码，并使用以下命令像以前一样构建项目：

```
1 $ mkdir build  
2 $ cd build  
3 $ cmake ..  
4 $ make
```

之后，你可以通过使用以下命令运行给定代码 `./BezierCurve`。运行时，程序将打开一个黑色窗口。现在，你可以点击屏幕选择点来控制 Bézier 曲线。程序将等待你在窗口中选择 4 个控制点，然后它将根据你选择的控制点来自动绘制 Bézier 曲线。代码框架中提供的实现通过使用多项式方程来计算 Bézier 曲线并绘制为红色。两张控制点对应的 Bézier 曲线如下所示：



在确保代码框架一切正常后，就可以开始完成你自己的实现了。注释掉 `main` 函数中 `while` 循环内调用 `naive_bezier` 函数的行，并取消对 `bezier` 函数的注释。要求你的实现将 Bézier 曲线绘制为绿色。

如果要确保实现正确，请同时调用 `naive_bezier` 和 `bezier` 函数，如果实现正确，则两者均应写入大致相同的像素，因此该曲线将表现为黄色。如果是这样，你可以确保实现正确。

你也可以尝试修改代码并使用不同数量的控制点，来查看不同的 Bézier 曲线。

4 评分与提交

评分：

- [5 分] 提交的格式正确，包含所有必须的文件。代码可以编译和运行。
- [20 分] De Casteljau 算法：
对于给定的控制点，你的代码能够产生正确的 Bézier 曲线。
- [5 分] 奖励分数：
实现对 Bézier 曲线的反走样。(对于一个曲线上的点，不只把它对应于一个像素，你需要根据到像素中心的距离来考虑与它相邻的像素的颜色。)
- [-2 分] 惩罚分数：
未删除 /build, /.vscode 和 assignment4.pdf。
未按格式建立 /images，缺少结果图片。
未提交或未按要求完成 README.md。
代码相关文件和 README 文件不在你提交的文件夹下的第一层。

提交：

- 当你完成作业后，请清理你的项目，记得在你的文件夹中包含 CMakeLists.txt 和所有的程序文件(无论是否修改)；
- 同时，请新建一个 /images 目录，将所有实验结果图片保存在该目录下；
- 再添加一个 README.md 文件写清楚自己完成了上述得分点中的哪几点(如果完成了，也请同时在 images 目录下提交一份结果图片并注明)，并简要描述你在各个函数中实现的功能；

- 最后,将上述内容打包,并用“姓名_Homework4.zip”的命名方式提交到 SmartChair 平台。

平台链接: <http://www.smartchair.org/GAMES101-Spring2021/>。

作业 5：光线与三角形相交

GAMES101, 2021 年春季

教授：闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

发布日期为北京时间 2021 年 7 月 23 日 (星期五) 上午**10: 00**

截止日期为北京时间 2021 年 8 月 2 日 (星期一) 上午**10: 00**

注意：

- 任何更新或更正都将发布在论坛上，因此请偶尔检查一下。
 - 论坛链接：<http://games-cn.org/forums/forum/graphics-intro/>。
 - 你必须独立完成自己的作业。
 - 你可以在论坛上发布帖子求助，但是发布问题之前，请仔细阅读本文档。
 - 在截止时间之前将你的作业提交到 SmartChair 上。
-

1 总览

在这部分的课程中，我们将专注于使用光线追踪来渲染图像。在光线追踪中最重要的操作之一就是找到光线与物体的交点。一旦找到光线与物体的交点，就可以执行着色并返回像素颜色。在这次作业中，我们需要实现两个部分：光线的生成和光线与三角的相交。本次代码框架的工作流程为：

1. 从 `main` 函数开始。我们定义场景的参数，添加物体（球体或三角形）到场景中，并设置其材质，然后将光源添加到场景中。
2. 调用 `Render(scene)` 函数。在遍历所有像素的循环里，生成对应的光线并将返回的颜色保存在帧缓冲区（`framebuffer`）中。在渲染过程结束后，帧缓冲区中的信息将被保存为图像。
3. 在生成像素对应的光线后，我们调用 `CastRay` 函数，该函数调用 `trace` 来查询光线与场景中最近的对象的交点。
4. 然后，我们在此交点执行着色。我们设置了三种不同的着色情况，并且已经为你提供了代码。

你需要修改的函数是：

- **Renderer.cpp 中的 `Render()`**: 这里你需要为每个像素生成一条对应的光线，然后调用函数 `castRay()` 来得到颜色，最后将颜色存储在帧缓冲区的相应像素中。
- **Triangle.hpp 中的 `rayTriangleIntersect()`**: `v0, v1, v2` 是三角形的三个顶点，`orig` 是光线的起点，`dir` 是光线单位化的方向向量。`tnear, u, v` 是你需要使用我们课上推导的 Moller-Trumbore 算法来更新的参数。

2 开始编写

在本次作业中，你将使用一个新的代码框架。和之前作业相似的是，你可以选择在自己电脑的系统或者虚拟机上完成作业。请下载项目的框架代码，并使用

以下命令像以前一样构建项目：

```
1 $ mkdir build  
2 $ cd build  
3 $ cmake ..  
4 $ make
```

之后，你就可以使用`./Raytracing` 来运行代码。现在我们对代码框架中的一些类做一下概括性的介绍：

- `global.hpp`: 包含了整个框架中会使用的基本函数和变量。
- `Vector.hpp`: 由于我们不再使用 Eigen 库，因此我们在此处提供了常见的向量操作，例如：`dotProduct`, `crossProduct`, `normalize`。
- `Object.hpp`: 渲染物体的父类。`Triangle` 和 `Sphere` 类都是从该类继承的。
- `Scene.hpp`: 定义要渲染的场景。包括设置参数，物体以及灯光。
- `Renderer.hpp`: 渲染器类，它实现了所有光线追踪的操作。

3 评分与提交

评分：

- [5 分] 提交的格式正确，包含所有必须的文件。代码可以编译和运行。
- [10 分] 光线生成：
正确实现光线生成部分，并且能够看到图像中的两个球体。
- [15 分] 光线与三角形相交：
正确实现了 Moller-Trumbore 算法，并且能够看到图像中的地面。
- [-2 分] 惩罚分数：
未删除 `/build`, `/.vscode` 和 `assignment5.pdf`。
未按格式建立 `/images`, 缺少结果图片。

未提交或未按要求完成 README.md。

代码相关文件和 README 文件不在你提交的文件夹下的第一层。

如果实现是正确的，你将得到下图：



提交：

- 当你完成作业后，**请清理你的项目**，记得在你的文件夹中包含 CMakeLists.txt 和所有的程序文件（无论是否修改）；
- 同时，请新建一个 /images 目录，将所有实验结果图片保存在该目录下；
- 再添加一个 README.md 文件写清楚自己完成了上述得分点中的哪几点（如果完成了，也请同时提交一份结果图片），并简要描述你在各个函数中实现的功能；

- 最后,将上述内容打包,并用“姓名_Homework5.zip”的命名方式提交到 SmartChair 平台。

平台链接: <http://www.smartchair.org/GAMES101-Spring2021/>。

作业 6: 加速结构

GAMES101, 2021 年春季

教授: 闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

发布日期为北京时间 2021 年 8 月 3 日 (星期二) 上午**10:00**

截止日期为北京时间 2021 年 8 月 13 日 (星期五) 上午**10:00**

注意:

- 任何更新或更正都将发布在论坛上, 因此请偶尔检查一下。
 - 论坛链接: <http://games-cn.org/forums/forum/graphics-intro/>。
 - 你必须独立完成自己的作业。
 - 你可以在论坛上发布帖子求助, 但是发布问题之前, 请仔细阅读本文档。
 - 在截止时间之前将你的作业提交到 SmartChair 上。
-

1 总览

在之前的编程练习中，我们实现了基础的光线追踪算法，具体而言是光线传输、光线与三角形求交。我们采用了这样的方法寻找光线与场景的交点：遍历场景中的所有物体，判断光线是否与它相交。在场景中的物体数量不大时，该做法可以取得良好的结果，但当物体数量增多、模型变得更加复杂，该做法将会变得非常低效。因此，我们需要加速结构来加速求交过程。在本次练习中，我们重点关注物体划分算法 Bounding Volume Hierarchy (BVH)。本练习要求你实现 Ray-Bounding Volume 求交与 BVH 查找。

首先，你需要从上一次编程练习中引用以下函数：

- `Render()` in `Renderer.cpp`: 将你的光线生成过程粘贴到此处，并且按照新框架更新相应调用的格式。
- `Triangle::getIntersection` in `Triangle.hpp`: 将你的光线-三角形相交函数粘贴到此处，并且按照新框架更新相应相交信息的格式。

在本次编程练习中，你需要实现以下函数：

- `IntersectP(const Ray& ray, const Vector3f& invDir, const std::array<int, 3>& dirIsNeg)` in the `Bounds3.hpp`: 这个函数的作用是判断包围盒 `BoundingBox` 与光线是否相交，你需要按照课程介绍的算法实现求交过程。
- `getIntersection(BVHBuildNode* node, const Ray ray)` in `BVH.cpp`: 建立 BVH 之后，我们可以用它加速求交过程。该过程递归进行，你将在其中调用你实现的 `Bounds3::IntersectP`.

2 开始实现

2.1 编译运行

基础代码只依赖于 CMake，下载基础代码后，执行下列命令，就可以编译这个项目：

```
1 $ mkdir build  
2 $ cd ./build  
3 $ cmake ..  
4 $ make
```

在此之后，你就可以通过 `./Raytracing` 来执行程序。

2.2 代码框架

我们修改了代码框架中的如下内容：

- Material.hpp: 我们从将材质参数拆分到了一个单独的类中，现在每个物体实例都可以拥有自己的材质。
- Intersection.hpp: 这个数据结构包含了相交相关的信息。
- Ray.hpp: 光线类，包含一条光的源头、方向、传递时间 t 和范围 range.
- Bounds3.hpp: 包围盒类，每个包围盒可由 pMin 和 pMax 两点描述（请思考为什么）。Bounds3::Union 函数的作用是将两个包围盒并成更大的包围盒。与材质一样，场景中的每个物体实例都有自己的包围盒。
- BVH.hpp: BVH 加速类。场景 scene 拥有一个 BVHAccel 实例。从根节点开始，我们可以递归地从物体列表构造场景的 BVH.

3 提交与评分

评分：

- [5 points] 提交格式正确，包含所有需要的文件；代码可以在虚拟机下正确编译运行。
- [20 points] 包围盒求交：正确实现光线与包围盒求交函数。
- [15 points] BVH 查找：正确实现 BVH 加速的光线与场景求交。
- [加分项 20 points] SAH 查找：自学 SAH(Surface Area Heuristic) ，正确实现 SAH 加速，并且提交结果图片，并在 README.md 中说明 SVH 的实现方法，并对比 BVH、SVH 的时间开销。（可参考 http://15462.courses.cs.cmu.edu/fall2015/lecture/acceleration/slide_024, 也可以查找其他资料）
- [-5 points] 未提交 README.md, 未提交 CMakeLists.txt, 未提交结果图片，未完整提交代码，提交包中多余文件（比如 /build, /.vs）未清除。



提交：

- 当你完成作业后，请清理你的项目，记得在你的文件夹中包含 CMakeLists.txt 和所有的程序文件（无论是否修改）；
- 同时，请新建一个 /images 目录，将所有实验结果图片保存在该目录下；

- 再添加一个 `README.md` 文件写清楚自己完成了上述得分点中的哪几点（如果完成了，也请同时提交一份结果图片），并简要描述你在各个函数中实现的功能；
- 最后，将上述内容打包，并用“`姓名_Homework6.zip`”的命名方式提交到 SmartChair 平台。
- 平台链接：<http://www.smartchair.org/GAMES101-Spring2021/>

作业 7: 路径追踪

GAMES101, 2021 年春季

教授: 闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

发布日期为北京时间 2021 年 8 月 14 日 (星期六) 下午**17: 00**

截止日期为北京时间 2021 年 8 月 24 日 (星期二) 下午**17: 00**

注意:

- 任何更新或更正都将发布在论坛上, 因此请偶尔检查一下。
 - 论坛链接: <http://games-cn.org/forums/forum/graphics-intro/>。
 - 你必须独立完成自己的作业。
 - 你可以在论坛上发布帖子求助, 但是发布问题之前, 请仔细阅读本文档。
 - 在截止时间之前将你的作业提交到 SmartChair 上。
-

1 总览

在之前的练习中，我们实现了 Whitted-Style Ray Tracing 算法，并且用 BVH 等加速结构对于求交过程进行了加速。在本次实验中，我们将在上一次实验的基础上实现完整的 Path Tracing 算法。至此，我们已经来到了光线追踪版块的最后一节内容。

请认真阅读本文档，按照本文档指示的流程完成本次实验。

2 调通框架

2.1 修改的内容

相比上一次实验，本次实验对框架的修改较大，主要在以下几方面：

- 修改了 main.cpp，以适应本次实验的测试模型 CornellBox
- 修改了 Render，以适应 CornellBox 并且支持 Path Tracing 需要的同一 Pixel 多次 Sample
- 修改了 Object, Sphere, Triangle, TriangleMesh, BVH，添加了 area 属性与 Sample 方法，以实现对光源按面积采样，并在 Scene 中添加了采样光源的接口 sampleLight
- 修改了 Material 并在其中实现了 sample, eval, pdf 三个方法用于 Path Tracing 变量的辅助计算

2.2 你需要迁移的内容

你需要从上一次编程练习中直接拷贝以下函数到对应位置：

- `Triangle::getIntersection` in Triangle.hpp: 将你的光线-三角形相交函数粘贴到此处，请直接将上次实验中实现的内容粘贴在此。
- `IntersectP(const Ray& ray, const Vector3f& invDir, const std::array<int, 3>& dirIsNeg)` in the Bounds3.hpp: 这个函数的

作用是判断包围盒 BoundingBox 与光线是否相交，请直接将上次实验中实现的内容粘贴在此处，并且注意检查 `t_enter = t_exit` 的时候的判断是否正确。

- `getIntersection(BVHBuildNode* node, const Ray ray)` in BVH.cpp: BVH 查找过程，请直接将上次实验中实现的内容粘贴在此处.

2.3 编译运行

基础代码只依赖于 CMake，下载基础代码后，执行下列命令，就可以编译这个项目：

```
1 $ mkdir build  
2 $ cd ./build  
3 $ cmake ..  
4 $ make
```

在此之后，你就可以通过 `./Raytracing` 来执行程序。请务必确保程序可以正常编译之后，再进入下一节的内容。

3 开始实现

3.1 代码框架

在本次实验中，你只需要修改这一个函数：

- `castRay(const Ray ray, int depth)` in Scene.cpp: 在其中实现 Path Tracing 算法

可能用到的函数有：

- `intersect(const Ray ray)` in Scene.cpp: 求一条光线与场景的交点
- `sampleLight(Intersection pos, float pdf)` in Scene.cpp: 在场景的所有光源上按面积 uniform 地 sample 一个点，并计算该 sample 的概率密度

- `sample(const Vector3f wi, const Vector3f N)` in Material.cpp: 按照该材质的性质，给定入射方向与法向量，用某种分布采样一个出射方向
- `pdf(const Vector3f wi, const Vector3f wo, const Vector3f N)` in Material.cpp: 给定一对入射、出射方向与法向量，计算 sample 方法得到该出射方向的概率密度
- `eval(const Vector3f wi, const Vector3f wo, const Vector3f N)` in Material.cpp: 给定一对入射、出射方向与法向量，计算这种情况下的 `f_r` 值

可能用到的变量有：

- `RussianRoulette` in Scene.cpp: `P_RR`, Russian Roulette 的概率

3.2 Path Tracing 的实现说明

课程中介绍的 Path Tracing 伪代码如下 (为了与之前框架保持一致，`wo` 定义与课程介绍相反):

```

1 shade(p, wo)
2   Uniformly sample the light at xx (pdf_light = 1 / A)
3   Shoot a ray from p to x
4   If the ray is not blocked in the middle
5     L_dir = L_i * f_r * cos_theta * cos_theta_x / |x-p|^2
6     / pdf_light
7
8   L_indir = 0.0
9   Test Russian Roulette with probability P_RR
10  Uniformly sample the hemisphere toward wi (pdf_hemi = 1
11    / 2pi)
12  Trace a ray r(p, wi)
13  If ray r hit a non-emitting object at q

```

```

12     L_indir = shade(q, wi) * f_r * cos_theta / pdf_hemi /
13     P_RR
14
15     Return L_dir + L_indir

```

按照本次实验给出的框架，我们进一步可以将伪代码改写为：

```

1 shade(p, wo)
2     sampleLight(inter, pdf_light)
3     Get x, ws, NN, emit from inter
4     Shoot a ray from p to x
5     If the ray is not blocked in the middle
6         L_dir = emit * eval(wo, ws, N) * dot(ws, N) * dot(ws,
7             NN) / |x-p|^2 / pdf_light
8
9     L_indir = 0.0
10    Test Russian Roulette with probability RussianRoulette
11    wi = sample(wo, N)
12    Trace a ray r(p, wi)
13    If ray r hit a non-emitting object at q
14        L_indir = shade(q, wi) * eval(wo, wi, N) * dot(wi, N)
15        / pdf(wo, wi, N) / RussianRoulette
16
17
18    Return L_dir + L_indir

```

请确保你已经清晰地理解 Path Tracing 的实现方式，再进入下一个环节的讨论。

4 结果与分析

本章节讨论得到结果与调试过程中需要特别注意的一些问题。

4.1 注意事项

1. 本次实验代码的运行非常慢，建议调试时调整 main.cpp 中的场景大小或 Render.cpp 中的 SPP 数以加快运行速度；此外，还可以实现多线程来进一步加快运算。
2. 注意数值精度问题，尤其注意 pdf 接近零的情况，以及 sampleLight 时判断光线是否被挡的边界情况。这些情况往往会造成渲染结果噪点过多，或出现黑色横向条纹。

4.2 参考结果

如果严格按照上述算法实现，你会发现渲染结果中光源区域为纯黑。请分析这一现象的原因，并且修改 Path Tracing 算法使光源可见。最终结果如下：



4.3 材质的拓展

目前的框架中拆分 sample, eval, pdf, 实现了最基础的 Diffuse 材质。请在不破坏这三个函数定义方式的情况下修改这三个函数，实现 Microfacet 模型。本任务不要求你实现复杂的采样手段，因此你依然可以沿用 Diffuse 材质采用的 sample 与 pdf 计算。

Microfacet 相关知识见第十七讲 Slides https://sites.cs.ucsb.edu/~lingqi/teaching/resources/GAMES101_Lecture_17.pdf.

5 提交与评分

评分：

- [5 points] 提交格式正确，包含所有需要的文件；代码可以在虚拟机下正确编译运行。
- [45 points] Path Tracing：正确实现 Path Tracing 算法，并提交分辨率不小于 512*512，采样数不小于 8 的渲染结果图片。
- [加分项 10 points] 多线程：将多线程应用在 Ray Generation 上，注意实现时可能涉及的冲突。
- [加分项 10 points] Microfacet：正确实现 Microfacet 材质，并提交可体现 Microfacet 性质的渲染结果。
- [-5 points] 未提交 README.md, README.md 中未说明需要说明的信息，未提交 CMakeLists.txt, 未提交结果图片，未完整提交代码，提交包中多余文件（比如 /build, /.vs）未清除。

提交：

- 当你完成作业后，请清理你的项目，记得在你的文件夹中包含 CMakeLists.txt 和所有的程序文件（无论是否修改）；

- 同时，请新建一个 `/images` 目录，将所有实验结果图片保存在该目录下；
- 再添加一个 `README.md` 文件写清楚自己完成了上述得分点中的哪几点，并说明提交结果的分辨率与采样数、计算时间；如果实现了扩展功能，你还需要简要描述你在各个函数中实现的功能；
- 最后，将上述内容打包，并用“姓名_Homework7.zip”的命名方式提交到 SmartChair 平台。

作业 8 质点弹簧系统

GAMES101, 2021 年春季

教授: 闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

发布日期为北京时间 2021 年 8 月 25 日 (星期三) 上午**12 : 00**

截止日期为北京时间 2021 年 9 月 04 日 (星期六) 晚上**23 : 59**

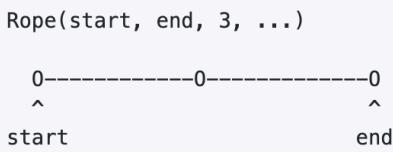
注意:

- 任何更新或更正都将发布在论坛上, 因此请偶尔检查一下。
 - 论坛链接: <http://games-cn.org/forums/forum/graphics-intro/>。
 - 你必须独立完成自己的作业。
 - 你可以在论坛上发布帖子求助, 但是发布问题之前, 请仔细阅读本文档。
 - 在截止时间之前将你的作业提交到 SmartChair 上。
-

1 总览

1.1 连接绳子的约束

在 `rope.cpp` 中, 实现 `Rope` 类的构造函数。这个构造函数应该可以创建一个新的绳子 (`Rope`) 对象, 该对象从 `start` 开始, `end` 结束, 包含 `num_nodes` 个节点。也就是如下图所示:



每个结点都有质量, 称为质点; 质点之间的线段是一个弹簧。通过创建一系列的质点和弹簧, 你就可以创建一个像弹簧一样运动的物体。

`pinned_nodes` 设置结点的索引。这些索引对应结点的固定属性 (pinned attribute) 应该设置为真 (他们是静止的)。对于每一个结点, 你应该构造一个 `Mass` 对象, 并在 `Mass` 对象的构造函数里设置质量和固定属性。(请仔细阅读代码, 确定传递给构造函数的参数)。你应该在连续的两个结点之间创建一个弹簧, 设置弹簧两端的结点索引和弹簧系数 `k`, 请检查构造函数的签名以确定传入的参数。

运行 `./ropesim`。你应该可以看到屏幕上画出绳子, 但它不发生运动。

1.2 显式/半隐式欧拉法

胡克定律表示弹簧连接的两个质点之间的力和他们之间的距离成比例。也就是:

$$\mathbf{f}_{\mathbf{b} \rightarrow \mathbf{a}} = -k_s \frac{\mathbf{b} - \mathbf{a}}{\|\mathbf{b} - \mathbf{a}\|} (\|\mathbf{b} - \mathbf{a}\| - l)$$

在 `Rope::simulateEuler` 中, 首先实现胡克定律。遍历所有的弹簧, 对弹簧两端的质点施加正确的弹簧力。保证力的方向是正确的! 对每个质点, 累加所有的弹簧力。

一旦计算出所有的弹簧力, 对每个质点应用物理定律:

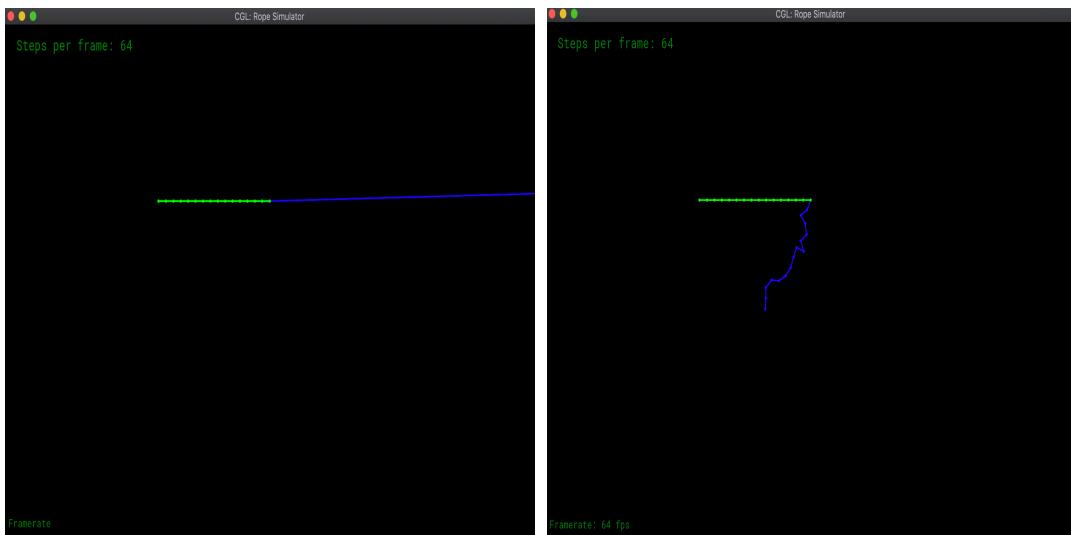
```

F=ma
v(t+1) = v(t) + a(t) * dt
x(t+1) = x(t) + v(t) * dt // For explicit method
x(t+1) = x(t) + v(t+1) * dt // For semi-implicit method

```

运行 `./ropesim`。仿真应该就开始运行了，但是只有 3 个结点，看起来不够多。在 `application.cpp` 文件的最上方，你应该可以看到欧拉绳子和 Verlet 绳子的定义。改变两个绳子结点个数（默认为 3 个），比如 16 或者更多。

运行 `./ropesim -s 32` 来设置仿真中每帧不同的仿真步数。尝试设置较小的值和较大的值（默认值为 64）。



1.3 显式 Verlet

Verlet 是另一种精确求解所有约束的方法。这种方法的优点是只处理仿真中顶点的位置并且保证四阶精度。和欧拉法不同，Verlet 积分按如下的方式来更新下一步位置：

$$x(t+1) = x(t) + [x(t) - x(t-1)] + a(t) \cdot dt \cdot dt$$

除此之外，我们可以仿真弹簧系数无限大的弹簧。不用再考虑弹簧力，而是用解约束的方法来更新质点位置：只要简单的移动每个质点的位置使得弹簧的长度保

持原长。修正向量应该和两个质点之间的位移成比例，方向为一个质点指向另一质点。每个质点应该移动位移的一半。



只要对每个弹簧执行这样的操作，我们就可以得到稳定的仿真。为了使运动更加平滑，每一帧可能需要更多的仿真次数。

1.4 阻尼

向显示 Verlet 方法积分的胡克定律中加入阻尼。现实中的弹簧不会永远跳动-因为动能会因摩擦而减小。阻尼系数设置为 0.00005, 加入阻尼之后质点位置更新如下：

$$x(t+1) = x(t) + (1 - \text{damping_factor}) * [x(t) - x(t-1)] + a(t) * dt * dt$$



1.5

你应该修改的函数是:

- rope.cpp 中的 `Rope::rope(...)`
- rope.cpp 中的 `void Rope::simulateEuler(...)`
- rope.cpp 中的 `void Rope::simulateVerlet(...)`

2 开始编写

2.0.1 依赖

本次作业需要预先安装 OpenGL, Freetype 还有 RandR 这三个库。可以通过以下命令进行安装:

```
$ sudo apt install libglu1-mesa-dev freeglut3-dev \
    mesa-common-dev
$ sudo apt install xorg-dev #会自动安装 libfreetype6-dev
```

2.0.2 开始吧

请下载工程的代码框架并通过下面的命令创建工程:

```
$ mkdir build  
$ cd build  
$ cmake ..  
$ make
```

之后，你应该可以使用命令`./ropesim` 来运行仿真。

3 评分和提交

评分:

- [5 分] 提交的格式正确，包含所有必须的文件。代码可以编译和运行。
- [5 分] 连接绳子约束，正确的构造绳子
- [5 分] 半隐式欧拉法
- [5 分] 显式欧拉法
- [10 分] 显式 Verlet
- [5 分] 阻尼
- [-2 分] 惩罚分数:

未删除 `/build`, `/.vscode` 和 `assignment8.pdf`。

未按格式建立 `/images`, 缺少结果图片。

未提交或未按要求完成 `README.md`。

代码相关文件和 `README` 文件不在你提交的文件夹下的第一层。

提交:

- 当你完成作业后，请清理你的项目，记得在你的文件夹中包含 `CMakeLists.txt` 和所有的程序文件 (无论是否修改);

- 提交的代码中应该包含带有阻尼的半隐式欧拉法和显式 Verlet，并且将显式欧拉法注释掉。
- 再添加一个 README.md 文件写清楚自己完成了上述得分点中的哪几点 (如果完成了，也请同时提交一份结果图片)，并简要描述你在各个函数中实现的功能；
- 最后，将上述内容打包，并用“姓名 Homework8.zip”的命名方式提交到 SmartChair 平台。

平台链接：<http://www.smartchair.org/GAMES101-Spring2021/>。