计算机图形学

课程Project说明

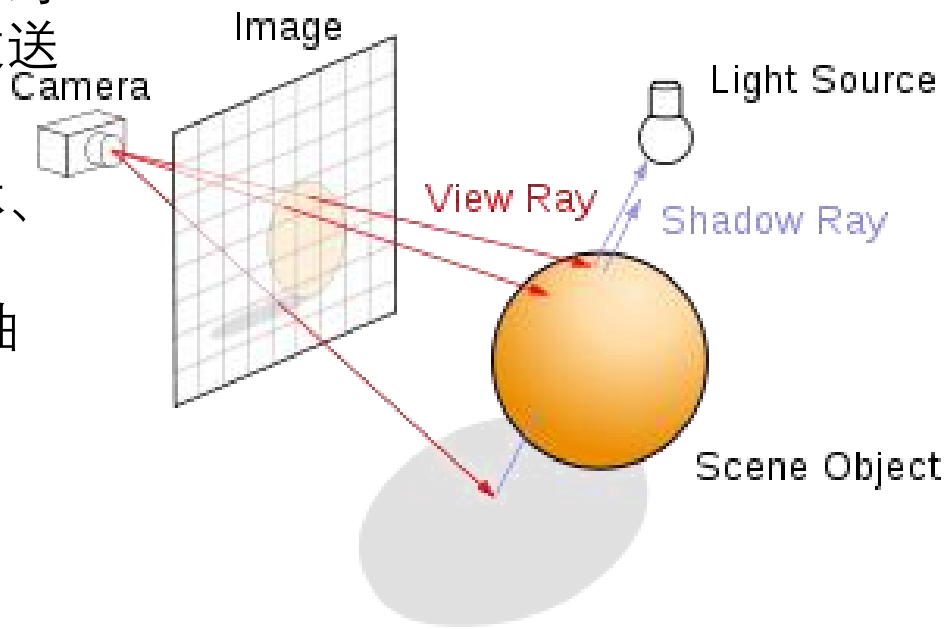
COMP130018.01

简介

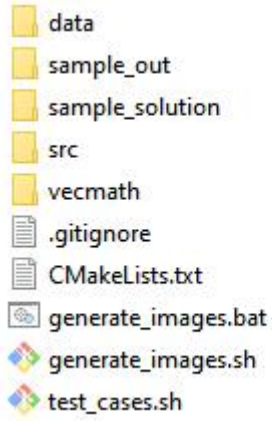
- 本学期共有三个Project：
- Project 1： 曲线和曲面造型技术， 占比**10%**
- Project 2： 光照模型与光线追踪， 占比**10%（今天发布）**
- Project 3： 占比**20%**
- 每组1-2人都需要在elearning上提交， 根据报告中的名字确定分组。
- TA联系方式：
  - 邹嘉文 24110240123@m.fudan.edu.cn
  - 卢健之 24110240058@m.fudan.edu.cn

Project 2 光照模型与光线追踪

- 在PJ2中，你将实现光线投射（ray casting）算法，并最终实现递归光线跟踪（光追，ray tracing）。光线投射为每个像素发送一条光线，并将其与场景中的所有对象相交。光线跟踪以同样的方式开始，但是递归地发送从场景中的对象交叉点反弹的额外光线。
- 你完成的程序将能够渲染几个基本的三维模型（球体、平面和三角形）和使用Phong着色模型加载的网格（mesh）模型，以及通过光线跟踪创建阴影和反射曲面的选项。
- 任务1：Phong着色模型
- 任务2：光线投射
- 任务3：光线追踪与阴影投射



Project 2 代码框架



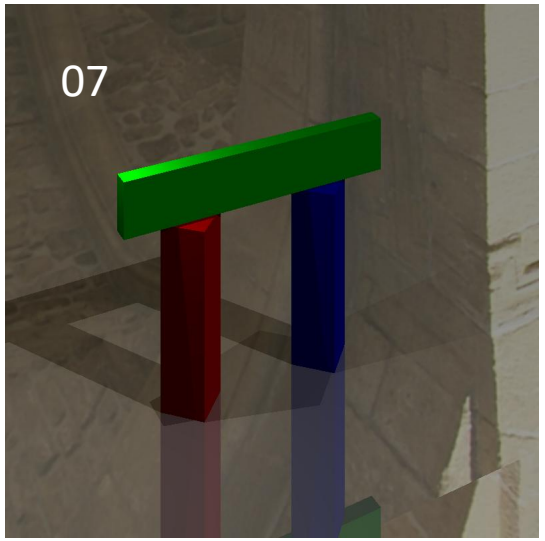
- 代码基于C++/OpenGL编写

1. data 为输入的模型文件
2. vecmath 为向量计算库
3. sample_solution 为实现的最终结果样例（包含linux/athena, Windows和Mac）
4. sample_out 为模型的正确输出结果
5. bat和sh文件为方便生成图像的脚本
6. src 为源代码

Project 2 代码框架

- 运行结果样例
- chmod u+x sample_solution/athena/a4
- ./a2 -input ../data/scene01_plane.txt -output 01.png -size 200 200
- ./a2 -input ../data/scene06_bunny_1k.txt -output 06.png -size 300 300 -bounces 4
- ./a2 -input ../data/scene07_arch.txt -output 07.png -size 300 300 -shadows -bounces 4

- -depth参数：生成深度图
- -normals参数：生成法线图
- -shadows参数：投射阴影
- -bounces参数：光追最大递归深度

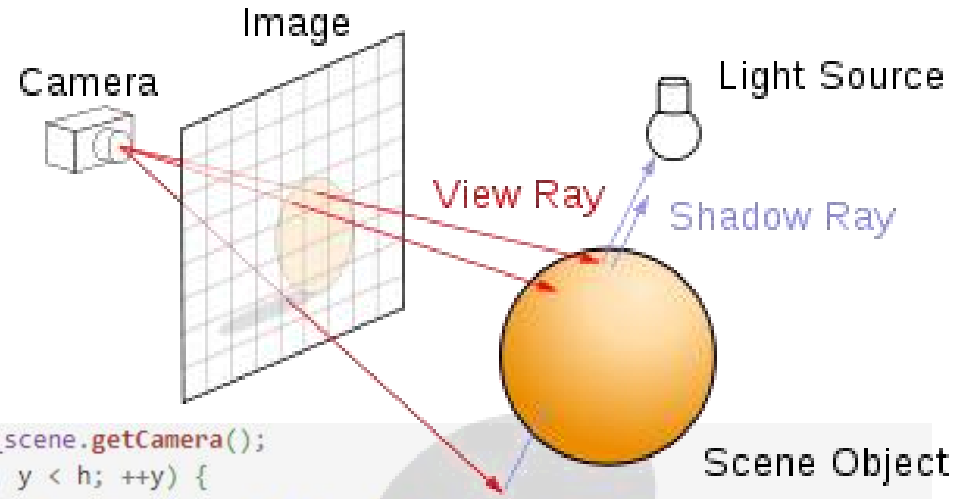


Project 2 代码框架

- **初始代码定义**
- 已经包含了一个**命令行参数解析器（ArgParser）**，以允许方便地访问命令行参数和标志。
- 已经包含了**场景解析器（SceneParser）**，它读取文本文件和关联的images/.obj来解析定义的场景。解析器会调用了几个构造函数和Group::addObject方法。你可以查看数据文件夹中的文本文件，以了解场景是如何定义的。
- **图像（Image）**类用于初始化和编辑图像的RGB值。该类还包括用于保存简单的PNG图像文件的函数。你的程序将输入场景文本文件，计算正确的像素颜色，并将其保存为PNG，计算大部分在渲染器（Renderer）类中。
- 已经包含了一个**光线（Ray）类和一个命中点（Hit）类**来操作相机光线及其交点。一条光线由它的原点和方向向量来表示。Hit类存储关于最近的相交点、法线、光线参数t的值和指向该交点处的对象的材质的指针的信息。
- **命中点（Hit）**数据结构必须用一个非常大的t值来初始化（例如std::numeric limit<float>()）。渲染过程中通过交点计算会修改命中点，以存储最近的光线参数t和相交对象的材质信息。

Project 2 代码框架

- **阅读Render()函数**
- **在Renderer::Render()中，已经提供了光线跟踪器的主循环。光线从相机出发，对于每个像素(ndcx，ndcy)发射。**Render()为每个相机光线调用traceRay()方法—你的任务是实现traceRay(),也就是根据光线方向求解像素的颜色。
- 仔细阅读视角相机类（PerspectiveCamera）。它已经实现，但是你应该理解代码是做什么的。




```
Camera* cam = _scene.getCamera();
for (int y = 0; y < h; ++y) {
    float ndcy = 2 * (y / (h - 1.0f)) - 1.0f;
    for (int x = 0; x < w; ++x) {
        float ndcx = 2 * (x / (w - 1.0f)) - 1.0f;
        // Use PerspectiveCamera to generate a ray.
        // You should understand what generateRay() does.
        Ray r = cam->generateRay(Vector2f(ndcx, ndcy));

        Hit h;
        Vector3f color = traceRay(r, cam->getTMin(), _args.bounces, h);

        image.setPixel(x, y, color);
        nimage.setPixel(x, y, (h.getNormal() + 1.0f) / 2.0f);
        float range = (_args.depth_max - _args.depth_min);
        if (range) {
            dimage.setPixel(x, y, Vector3f((h.t - _args.depth_min) / range));
        }
    }
}
// END SOLN
```

Project 2.1 Phong光照模型

- 任务1要求：
- 你将实现点光源和Phong反射率模型。
- 你需要实现Light.cpp中的PointLight::getIllumination()方法，Material.cpp中的Material::shade()方法，Renderer.cpp中的Renderer::traceRay()方法。
- 请参考教材《计算机图形学》P301第13.1章简单光照明模型或者《Fundamentals of Computer Graphics》4.5 Shading

Project 2.1 Phong光照模型

- **阅读getIllumination()函数**

首先，您将需要实现Light.cpp中的PointLight::getIllumination()方法。DirectionalLight::getIllumination()已经实现了。

这个函数输入在空间中的一个点并修改以下三个值：

(a)tolight：从场景中一个点指向到光源的方向矢量（归一化后的）。

(b)intensity：此时的照明强度（RGB）。

(c)distToLight：场景点与光源之间的距离。

距离场景点x_surf的距离为d的点光的强度由下面公式定义。

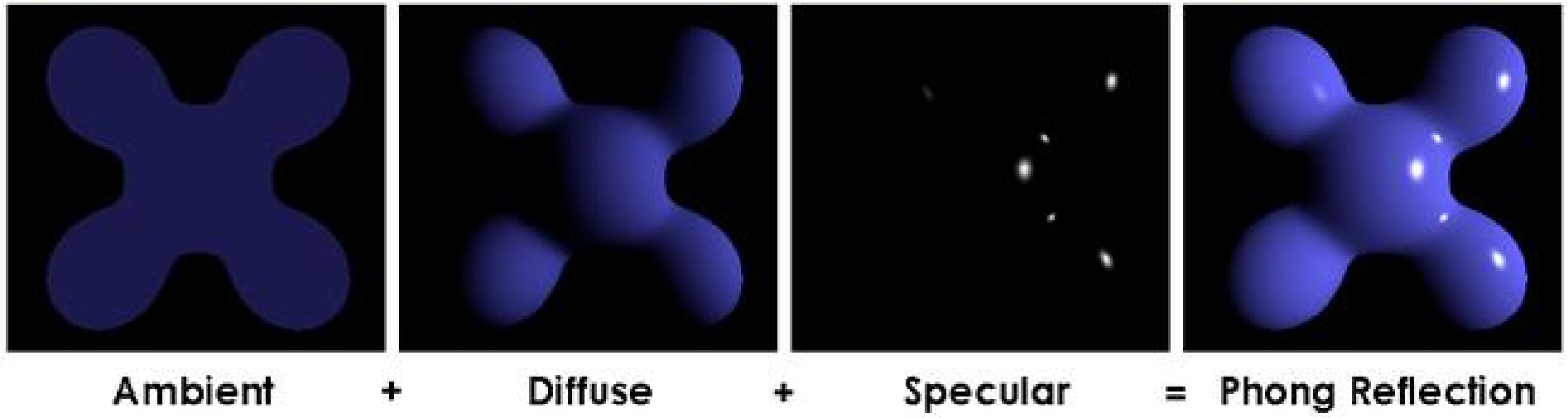
$$L(x_{\text{surf}}) = \frac{I}{\alpha d^2}$$

其中I是光源的颜色。α是衰减因子，光强随距离平方衰减，由PointLight类的_falloff决定。

```cpp
#include "Light.h"
    void DirectionalLight::getIllumination(const Vector3f &p,
                                           Vector3f &tolight,
                                           Vector3f &intensity,
                                           float &distToLight) const
    {
        // the direction to the light is the opposite of the
        // direction of the directional light source

        // BEGIN STARTER
        tolight = -_direction;
        intensity  = _color;
        distToLight = std::numeric_limits<float>::max();
        // END STARTER
    }
    void PointLight::getIllumination(const Vector3f &p,
                                     Vector3f &tolight,
                                     Vector3f &intensity,
                                     float &distToLight) const
    {
        // TODO Implement point light source
        // tolight, intensity, distToLight are outputs
    }
```

Project 2.1 Phong光照模型

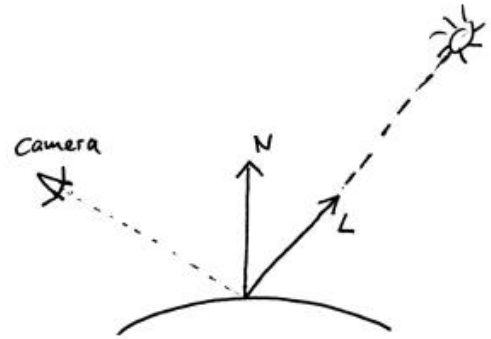
- Phong lighting 是一个简单的光照模型，也有别名叫做 Phong reflection model、Phong illumination。在这个模型下：我们把光分为 环境光（Ambient）、散射光（Diffuse）、镜面光照（Specular），著名的示意图如下：



Project 2.1 Phong光照模型

然后你将在Material.cpp中的Material::shade()实现漫反射（diffuse）着色器。对于给定的点光源生成照明值， Material::shade中将实现Phong着色模型。也就是说给定光线方向L和屏幕法向量N，当L·N小于0时，光源在切平面以下，不计算漫反射。我们计算漫反射阴影如下：

$$\text{clamp}(\mathbf{L},\mathbf{N}) = \begin{cases} \mathbf{L}\cdot\mathbf{N} & \text{if } \mathbf{L}\cdot\mathbf{N} > 0 \\ 0 & \text{otherwise} \end{cases}$$



结合漫反射材料的反射率k_diffuse和光强L，漫反射的强度为：

$$I_{\text{diffuse}} = \text{clamp}(\mathbf{L}\cdot\mathbf{N}) * L * k_{\text{diffuse}}$$

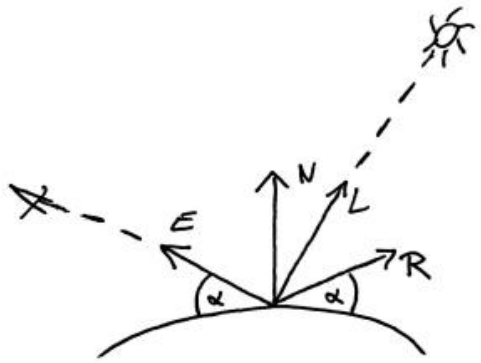
注意每个RGB通道单独计算。k_diffuse由Material类的_diffuseColor决定。

Project 2.1 Phong光照模型

接着你将实现Phong着色器的镜面反射（specular）。镜面反射强度取决于：光泽度s、表面对眼睛的方向E、完美反射矢量R、对光的方向L和表面法线N。镜面反射项的公式为：

$$I_{\text{specular}} = \text{clamp}(\mathbf{L}, \mathbf{R})^s * L * k_{specular}$$

这个公式与漫反射相似，但是，现在我们使用完美反射矢量R和L的夹角表示，这能使得高光随着相机的移动而移动。同时，我们将镜面反射计算s幂，较高的光泽s使高亮部分更窄，表面显得更有光泽，较小的s使表面显得更有哑光外观。k_specular由Material类的_specularColor决定。



Project 2.1 Phong光照模型

最后我们把这些项加起来就得到了Phong光照模型。一般我们的场景有环境照明L_ambient，和几个光源，我们需要把这些项加起来。其中环境光定义为：

$$I_{\text{ambient}} = L_{\text{ambient}} * k_{\text{diffuse}}.$$

Phong光照模型就是环境光，漫反射和镜面反射的结合，简单地将它们相加即可：

$$I = I_{\text{ambient}} + \sum_{i \in \text{lights}} I_{\text{diffuse},i} + I_{\text{specular},i}.$$

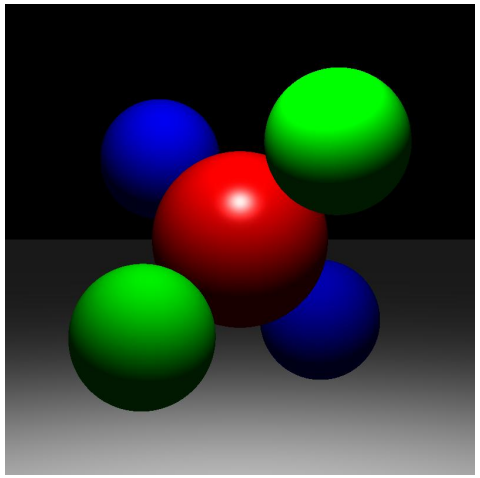
光强度值I是写入帧缓冲区的最终像素强度。注意因为环境光ambient在Scene类中定义，我们在Material::shade()里面不添加环境光的一项。

Project 2.1 Phong光照模型

- 最后我们将在Renderer::traceRay()里面完成Phong光照模型的计算。现在对于每个达到的点我们直接返回了材质的漫反射颜色，在Phong光照模型中我们需要遍历场景中所有的光源（_scene.lights）计算光照（shade），然后把他们与环境光（_scene.getAmbientLight()）累加起来。
- 现在你应该能看见场景1中的球体的光照了。

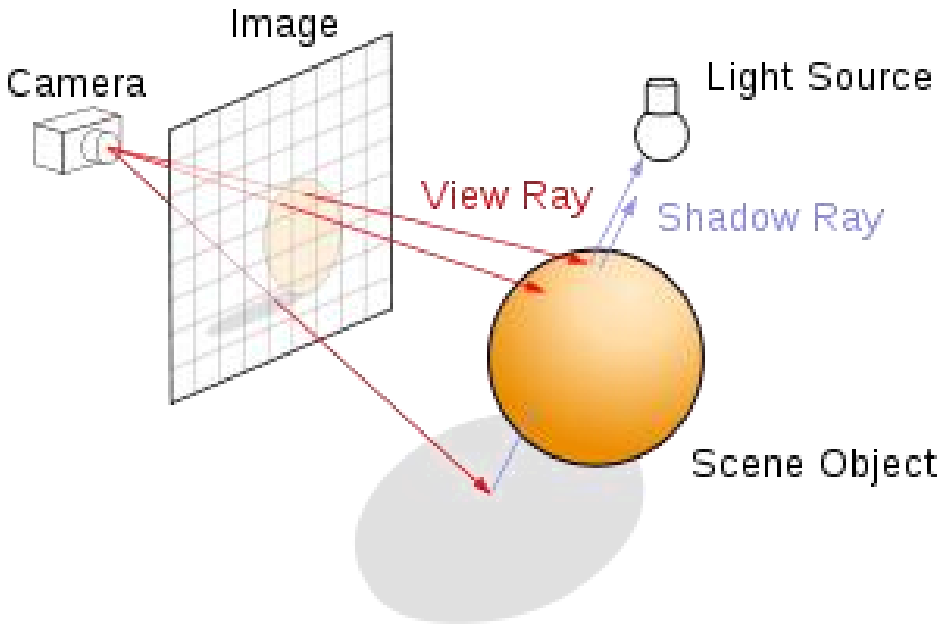
```
Vector3f
Renderer::traceRay(const Ray &r,
    float tmin,
    int bounces,
    Hit &h) const
{
    // The starter code only implements basic drawing of sphere primitives.
    // You will implement phong shading, recursive ray tracing, and shadow rays.

    // TODO: IMPLEMENT
    if (_scene.getGroup()->intersect(r, tmin, h)) {
        return h.getMaterial()->getDiffuseColor();
    } else {
        return Vector3f(0, 0, 0);
    };
}
```

下一节将提到的相交函数



Project 2.2 光线投射

- 任务1要求：
- 使用面向对象的技术，你将使你的光线追踪器灵活和可扩展。一个通用的Object3D类将作为所有三维物体的父类。你的工作是实现它们专门化的子类，并实现相交（intersect）函数。
- 你需要在Object3D.cpp中填写Plane，Triangle，Transform的Intersect()函数实现。
- 请参考教材《计算机图形学》P319第13.6节整体光照明模型与光线跟踪算法或者《Fundamentals of Computer Graphics》4.4 Ray-Object Intersection



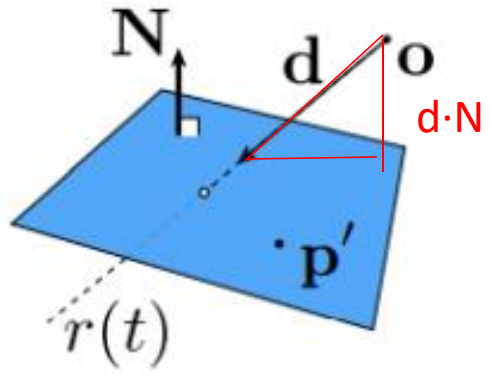
Project 2.2 光线投射

- **仔细阅读Object3D类**。您不能直接创建一个抽象类的实例，但是您可以通过实例化它的一个子类来使用该抽象类。它有一种方法来计算给定的光线是否与对象相交，并且它还存储了一个指向其材质（Material）类型的指针。
- **仔细阅读Sphere类**，它继承自Object3D并实现intersect()方法。代码已经实现了Sphere，而你将实现Object3D的其他子类。
- **仔细阅读Group类。**Group类是所有物体的超类，traceRay()方法对Group进行intersect就是对所有物体intersect。
- **仔细阅读Sphere类的相交（intersect）方法**里面，我们寻找沿光线的最近交点，用t参数化，给定物体，光线，上一个交点，判断是否与物体相交。tmin用来限制相交的范围。
- 如果发现一个交集使得t > tmin和t小于当前存储在命中点（Hit）数据结构中的交集的值，则根据需要更新Hit。请注意，如果新的交点比前一个交点更近，则需要修改Hit对象，并且相交必须返回true。
- 注意，Hit是通过引用传递的，这意味着在方法中修改它将修改传入的原始对象。我们通过对所有的物体intersect之后就知道光线与哪个物体有最近的相交了。

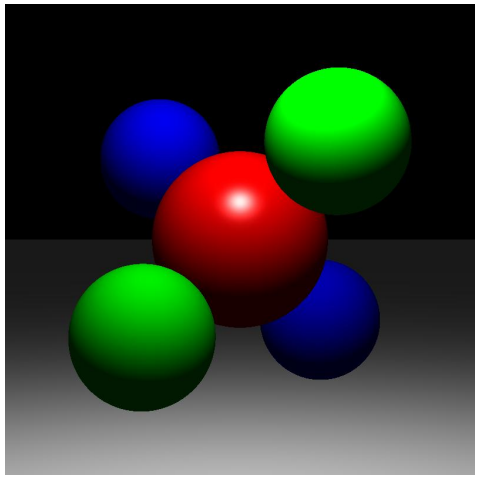
```
bool Sphere::intersect(const Ray &r, float tmin, Hit &h) const
{
    // BEGIN STARTER

    // We provide sphere intersection code for you.
    // You should model other intersection implementations after this one.

    // Locate intersection point ( 2 pts )
    const Vector3f &rayOrigin = r.getOrigin(); //Ray origin in the world coordinate
    const Vector3f &dir = r.getDirection();

    Vector3f origin = rayOrigin - _center;      //Ray origin in the sphere coordinate

    float a = dir.absSquared();
    float b = 2 * Vector3f::dot(dir, origin);
    float c = origin.absSquared() - _radius * _radius;

    // no intersection
    if (b * b - 4 * a * c < 0) {
        return false;
    }

    float d = sqrt(b * b - 4 * a * c);

    float tplus = (-b + d) / (2.0f*a);
    float tminus = (-b - d) / (2.0f*a);

    // the two intersections are at the camera back
    if ((tplus < tmin) && (tminus < tmin)) {
        return false;
    }

    float t = 10000;
    // the two intersections are at the camera front
    if (tminus > tmin) {
        t = tminus;
    }

    // one intersection at the front. one at the back
    if ((tplus > tmin) && (tminus < tmin)) {
        t = tplus;
    }

    if (t < h.getT()) {
        Vector3f normal = r.pointAtParameter(t) - _center;
        normal = normal.normalized();
        h.set(t, this->material, normal);
        return true;
    }
    // END STARTER
    return false;
}
```

Project 2.2 光线投射

- 第一步，实现平面（Plane）类。它是从Object3D派生出来的无穷平面。平面方程定义为P·n=d，d为相对原点的偏移量，n是法线，P是平面上的点。为了实现相交（intersect）函数，请确保除了相交距离t和颜色外，相交函数还会更新Hit存储的法向量值。
- 实现平面类之后就能够渲染场景1中球体和平面的模型。



$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}}$$



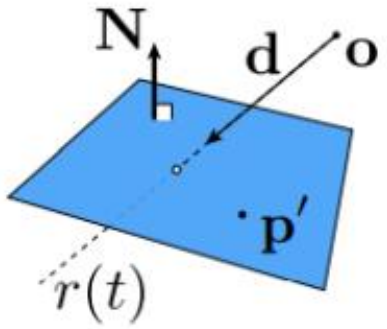
Project 2.2 光线投射

- 第二步，实现三角形（Triangle）类。它也是从Object3D派生出来的。构造函数输入3个顶点，每个顶点包括法线和材质。

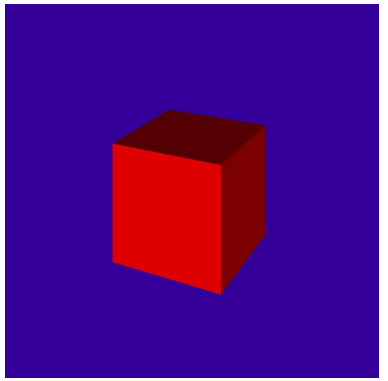
（1）判断射线是否与**三角形所在**平面相交。当光线r (t)与平面相交， 就有点 p 满足(p – p ’)·N =0 ， 同时p = o + t * d 。只有当(p – p ’)·N = 0 才能满足p 点在平面内。(p – p ’)这个向量垂直于法线方向。

（2）确定射线与平面相交后，再求得其交点p ,并判断该点是否在三角形内。

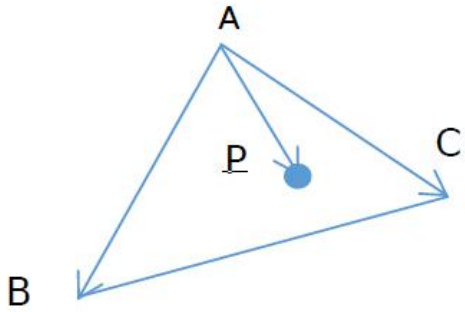
- 实现三角形类之后就能够渲染场景2中立方体的模型。



$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}}$$



Project 2.2 光线投射



- 射线与三角形相交的Möller-Trumbore 算法（https://www.oakchina.cn/2022/02/16/moller-trumbore/）
- 对于三角形ABC 内的任意一点P， 向量AP 、AB 、AC 线性相关。那么P的坐标可以表示为如下的公式。这代表着，P一定可以表示为三个顶点的线性组合，而且系数的和为1。

$$\overrightarrow{AP} = u\overrightarrow{AB} + v\overrightarrow{AC} \qquad P = (1-u-v)A + uB + vC$$

- 在线性代数中要求解一个3x3的线性方程，你可以将其表示为Ax = b，并使用Matrix3f::inverse()来求解该方程。
- 然后我们就得到了P的重心坐标（https://zhuanlan.zhihu.com/p/361943207）表示。如果P在三角形内那么P必须满足三个系数都在0到1之间。

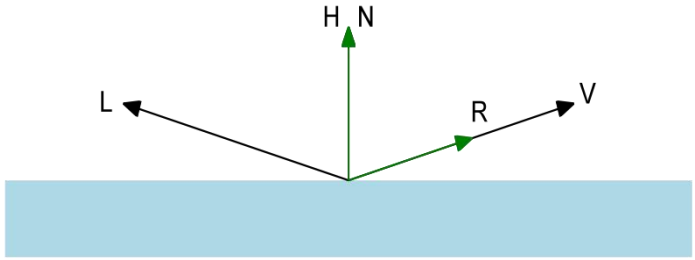
Project 2.2 光线投射

- 第三步，实现变换（Transform）类。请参考教材《计算机图形学》P130第7.8节三维几何变换或者PJ1的广义圆柱体。它也是从Object3D派生出来的。变换类存储一个指向子对象三维节点的指针。它还存储了一个4x4的变换矩阵M。这个矩阵将子对象从局部对象坐标移动到世界坐标。
- 但是对于复杂的子对象，例如具有许多顶点的网格，每当我们想要跟踪一条射线时，我们不能将整个对象移动到世界坐标。而将光线从世界坐标移动到局部对象坐标要计算小得多。
- 命中点（Hit）找到的时候，命中法线也会在对象坐标中。你必须将法线从局部坐标转换回世界坐标。请记住，和PJ1一样在变换法线方向时，你需要通过变换矩阵的逆转置来进行变换。
- 实现完变换类之后你就可以渲染场景1-5了。

Project 2.3 光线追踪与阴影投射

- 任务3要求:
- 在光线投射完成后，你将通过在渲染器（Renderer::Render）递归中进行函数调用来实现光线跟踪。你还可以通过投射阴影光线来确定可见性。
- 你需要实现Renderer::traceRay()函数中的递归调用并且实现阴影光线。
- 请参考教材《计算机图形学》P319第13.6节整体光照明模型与光线跟踪算法《Fundamentals of Computer Graphics》4.6 A Ray-Tracing Program和4.7 Shadows

Project 2.3 光线追踪与阴影投射



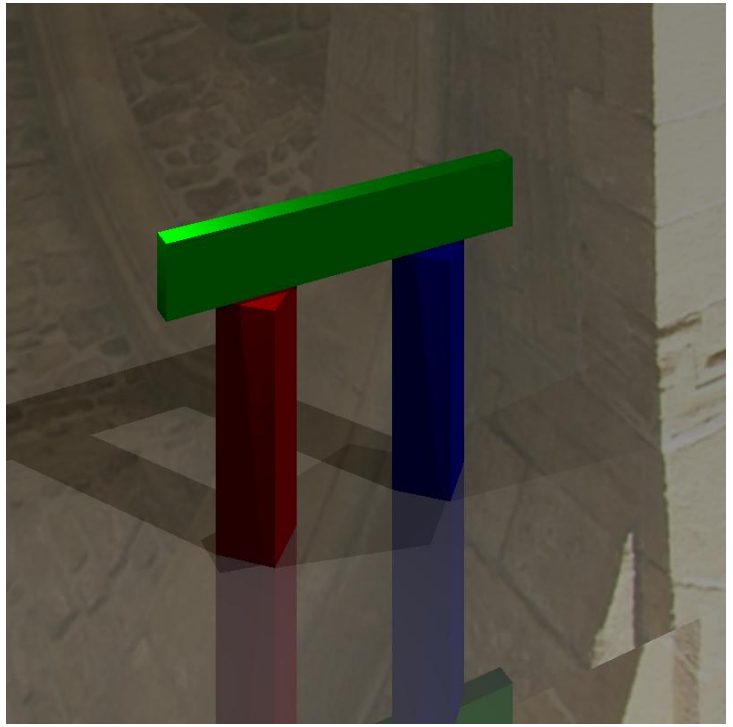
- 你将实现-bounces参数的功能，也就是光线追踪。
- traceRay将对**镜面材料(Specular)**进行递归调用。最大递归深度作为命令行参数传递-bounces max_bounces。请使用样例程序并修改这个参数尝试一下。注意，即使max_bounces=0，程序也会做光线投射。
- 请实现通过从当前交点（Hit）发送光线（Ray）到完美反射方向R的光线，使用修改的递归深度递归调用traceRay来跟踪辅助射线。
- 总的光强现在变为了直接光和间接光的加和。我们将反射射线所看到的颜色乘以镜面材料的反射率添加到为当前射线计算的颜色中。如果直接照明（环境照明、漫反射照明、镜面照明）是I_direct，则直接照明和间接照明的总强度为

$$I_{\text{total}} = I_{\text{direct}} + k_{\text{specular}} * I_{\text{indirect}}.$$

- 如果一条光线没有击中任何东西，只需通过SceneParser::getBackgroundColor(dir)返回背景颜色即可。
- 实现这个之后你可以渲染场景6的兔子模型了。

Project 2.3 光线追踪与阴影投射

- 你将实现-shadows参数的功能，也就是阴影计算。
- 要计算投射的阴影，在计算表面上面每一个点的颜色之前，也就是2.1节中每次shade()函数调用之前。
- 你将从这个点（Hit）向光源（light）发送光线。如果在光源之前有相交点（Hit），则当前的表面点处于阴影中，并忽略来自该光源的直接照明。
- 请注意，必须将阴影光线发送到所有光源。你必须将光线原点稍微远离表面，或者同等地将tmin设置为某个极小值，否则会和自己相交。
- 实现这个之后你可以渲染场景7的模型了。

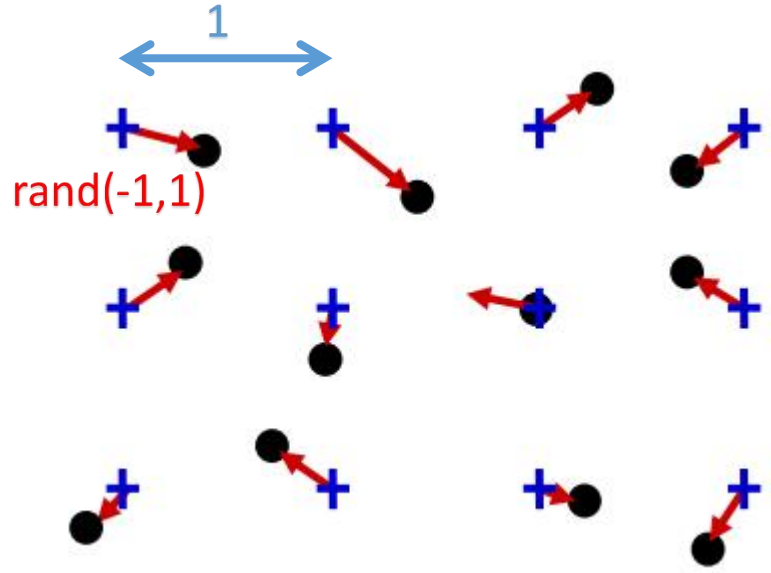


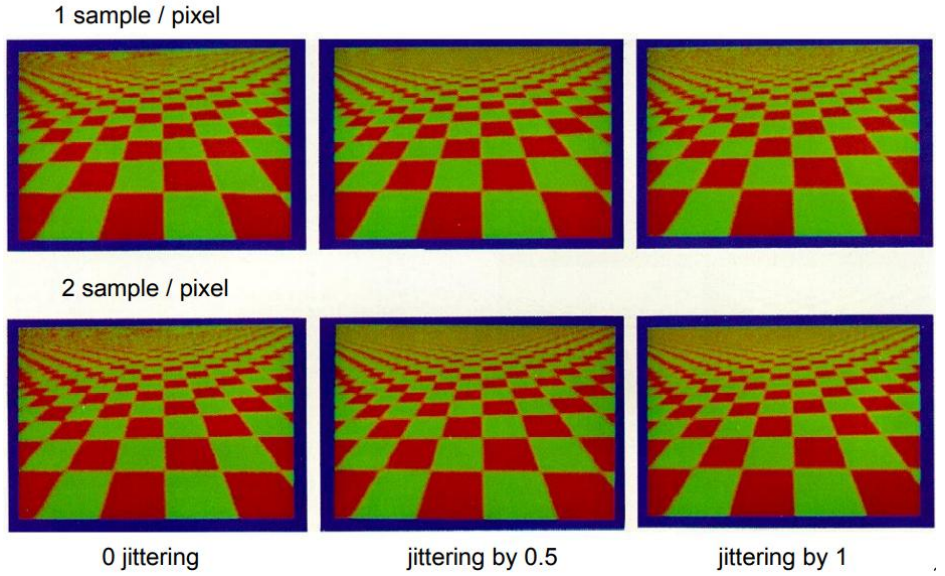
Project 2 拓展 - 抗锯齿的问题

- 抗锯齿是非常常见的图形学技术，在很多游戏的画质选项里面我们会遇到FXAA，TAA，DLSS和FSR等等抗锯齿的方法。我们实现的渲染器也存在很明显的锯齿，需要使用一定的方法来缓解。教材《计算机图形学》P326第13.6.5节光线跟踪算法的混淆与反混淆中提到了这个问题，我们可以通过增加采样和提升分辨率进行解决。
- 抖动采样是一种用于创建更逼真、更自然的图形的技术。它涉及随机偏移像素内样本的位置，以模拟相机或眼睛轻微移动的效果。抖动采样可以参考《虎书》13.4.1 Antialiasing。
- 提升分辨率通常与高斯模糊滤波结合使用，高斯模糊滤波是一种常见的图像平滑滤波器。高斯滤波可以参考《虎书》9.3 Convolution Filters。
- 请在Renderer::Render()函数中实现并使用-jitter（抖动采样）和 -filter（控制上采样和高斯滤波）两个参数分别进行控制，具体效果可以参考示例程序使用-jitter -filter参数的结果。
- 我们在*上采样3倍分辨率*下进行*16次充分抖动采样（jitter）*，然后进行*σ为1（也就是3x3像素变成1x1的像素，只考虑相邻的像素）的高斯滤波（filter）*。

Project 2 拓展 - 抗锯齿的问题

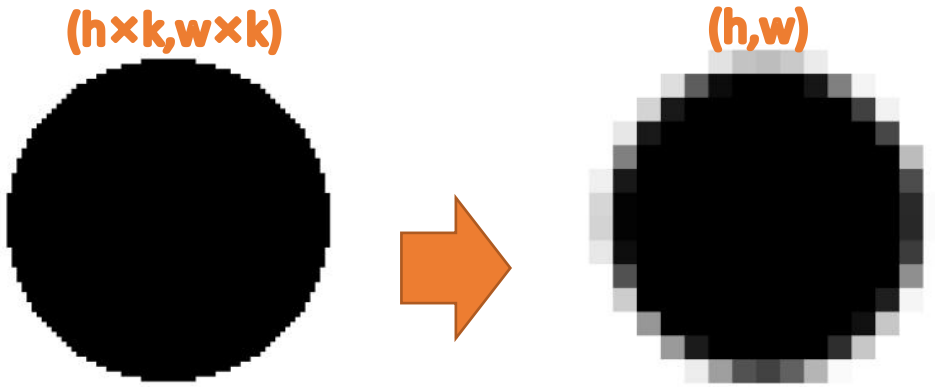
- 抖动采样（-jitter）：由皮克斯公司（Pixar）提出均匀采样+随机抖动，可以防止锯齿和摩尔纹（Moire）。距离为1时为充分采样。
- 我们在Render()的循环里面，重复*16次*的generateRay和traceRay过程，每次都对ndcx, ndcy加上一定的偏差，偏差的值从0到1的**像素宽度**随机选取。（注意代码中ndcx和ndcy已经除以整图宽高）

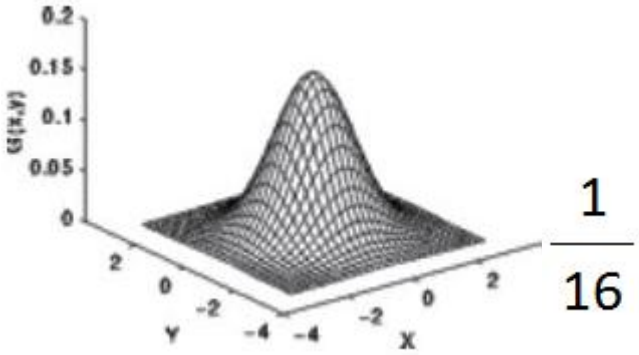




Project 2 拓展 - 抗锯齿的问题

- 高斯滤波（-filter）：
- 我们在Render()循环中先进行倍数是k的上采样（在高分辨率h×k,w×k下渲染，k=3），然后对于生成的Image类型，我们再下采样（得到低分辨率(m,n)图像）。
- 循环之后，下采样的时候需要一个低通（低频率通过）滤波器，从k×k个像素变成1个像素。
- 其中一个比较好的滤波器就是高斯滤波器，它的做法是将k×k个像素加权求和作为中间像素的值。其中权重通过高斯分布（二维正态分布）来计算得到。
- 请注意我们这里是离散的像素而且只考虑相邻的像素，所以可以简化为右边的矩阵。





$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, -\infty < x < \infty$$

$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

Project 2 提示

- **增量实现和调试。一次只实现一个步骤。一次只测试一个步骤。**
- 使用较小的图像大小来实现更快的调试。200×200像素通常足以意识到可能出了问题。使用更高的分辨率（例如800x800)来调试几何图形和阴影的细节。
- 不要犹豫地打印尽可能多的调试信息，如光线的方向向量、命中值等。
- 使用assert()来检查函数前提条件、数组索引等。
- 为了避免内存访问出错（Segmentation Fault），请确保不要尝试访问在图像宽度和高度之外的像素样本。边界上的像素将有一个裁剪过的支撑区域。

Project 2 参考资料

请务必仔细阅读初始代码，特别是src和vecmath，代码是最好的参考资料！
如果不清楚应该实现成什么样，请运行sample_solution下面的程序！

参考网站

- C++教程：https://www.cprogramming.com/tutorial/
- OpenGL教程：https://learnopengl.com
- GLSL 教程：https://www.lighthouse3d.com/tutorials/glsl-tutorial/

参考书籍：

- 计算机图形学 / 倪明田，吴良芝编著 北京 : 北京大学出版社, 1999
- Shirley, Peter, Michael Ashikhmin, Steve Marschner. Fundamentals of Computer Graphics. 3rd ed. A K Peters/CRC Press, 2009. ISBN: 9781568814698. (计算机图形学/虎书)
- Buss, Samuel R. 3D Computer Graphics: A Mathematical Introduction with OpenGL. 2003. ISBN: 9780521821032. (3D计算机图形学（OpenGL版）)

参考课程：

- MIT 6.837:

https://ocw.mit.edu/courses/6-837-computer-graphics-fall-2012/

- GAMES101: 现代计算机图形学入门

https://sites.cs.ucsb.edu/~lingqi/teaching/games101.html

Project 2 评分细则

项目完成度及正确性：（共计5分）

1. 光线投射：正确编译并输出Scene 1-5 的结果**（必做，每个样例正确显示得0.6分，共计3分）**

2. 光线追踪与阴影：正确编译并输出Scene 6-7的结果（每个样例正确显示得0.5分，共计1分）

3. 解决锯齿问题：添加抖动采样和滤波器来缓解锯齿问题（选做，共计1分）

项目报告：（共计5分）

撰写项目报告，**附上所有生成的图形的截图（报告中请尽量使用高分辨率图像）**，详细说明光线投射，光线追踪以及抗锯齿问题的解决方式**（必做，5分）**

Project 2 提交方式

- 可编译项目代码以及Linux编译程序（starter2文件夹）及Project2报告（PDF）请打包（zip）并上传Elearning
- 邮件/压缩包标题：2025图形学PJ2 姓名1 姓名2
- **项目报告DDL: 2025年5月8日中午11:59**

- 若对Project有疑问，可邮件/微信与TA联系
- TA办公地址：江湾校区交叉学科2号楼A4008室

- **严禁抄袭，包括网络上和同学的代码，一经发现Project作0分处理**
- **只实现必做功能也一定可以顺利通过，不要铤而走险，迟交会酌情扣分**