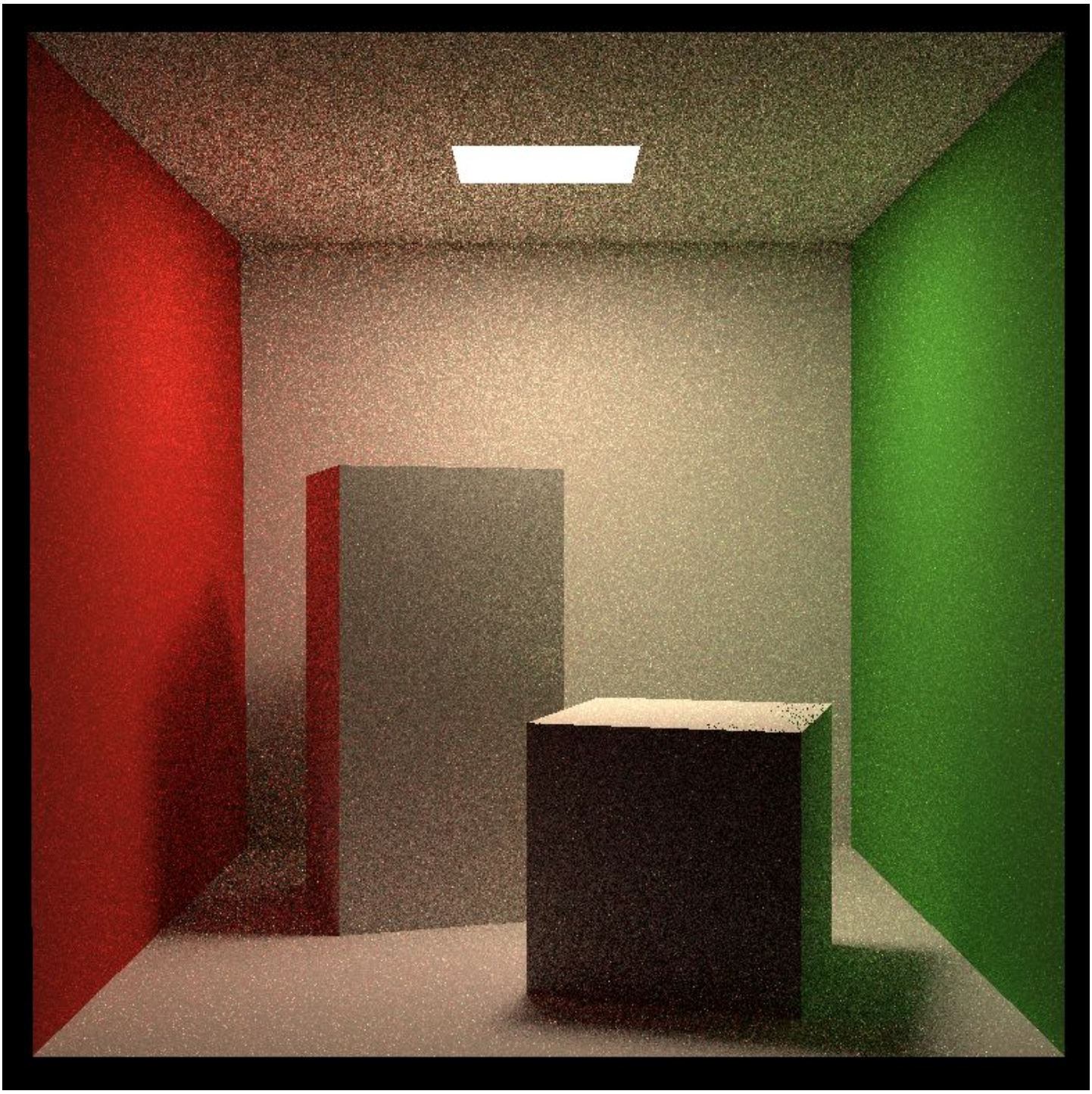


GAMES101 作业问题整理

b1gm0use
热爱运动和电子游戏的码农

（题图为作业7的渲染结果）

前言

整理一下写作业时出现的一些问题供大家参考，其中有些是框架导致的，有些对算法实现的理解偏差导致的，还有一些库的使用方法，总之都是一些琐碎的问题。**这里不会贴代码，也不会写具体的实现方案**，毕竟解决问题还得靠自己。

补充一下闫老师关于作业的三点要求：

> **1. Work alone for regular assignments**
> no copy-pasting from any other sources
> **2. Do not publish your code (on Github, etc.) for assignments using our skeleton code**
> **3. Do not post your solution online**
> Discussion / explanation is welcomed

建议：如果写作业时出现问题，建议还是先去BBS上搜索，99%的问题可能在论坛中都有人帮忙解决了。

作业1

1. zNear 和 zFar 符号问题：框架中初始化时使用的是 0.1 和 50 ，但是课件中使用的都是负值，为了和课件一致所以在计算投影矩阵之前使用了 zNear = -zNear, zFar = -zFar，否则得到的图形会关于原点旋转180度的效果。我的简单理解是，使用正数 zNear 和 zFar 会导致投影变换的视锥看向Z轴正向，这样绘制的三角形其实出现在了投影变化的反向视锥里，导致出现关于原点对称的问题。这个问题在论坛上也有讨论，可以参考这里：

http://games-cn.org/forums/topic/%e4%bd%9c%e4%b8%9a%...
games-cn.org/forums/topic/%e4%bd%9c%e4%b8%9...

1. Rodrigues' rotation formula 的推导过程：闫老师给出的附加材料没有看懂，Wikipedia 上的更加浅显易懂一些：

https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula
en.wikipedia.org/wiki/Rodrigues%27_rotation_formula

1. 角度与弧度的问题：C++ 标准库中的sin/cos 等计算，均以弧度为参数，角度转弧度可以用框架中提供的 MY_PI 常量。
2. Vector 和 Matrix 可以用 transpose() 方法进行转置，方便输出，有些计算的时候也需要转置矩阵。
3. Vector 和 Matrix 的 初始化方法：

Advanced initialization
eigen.tuxfamily.org/dox/group__TutorialAdvancedIniti...

1. Vector 和 Matrix 一般的初始化使用 comma initializer 就够用了，即使用 << 和 , 按行进行赋值，可以使用数字、Vector 和 Matrix 进行赋值。
2. 3x3 矩阵构造 4x4 矩阵的简单方法，即按行进行赋值：

```
rotation4 << rotation3.row(0), 0,
             rotation3.row(1), 0,
             rotation3.row(2), 0,
             0, 0, 0, 1;
```

或者先将整个矩阵初始化为一个单位矩阵，然后再修改其左上角矩阵

```
rotation4 = Eigen::Matrix4f::Identity();
rotation4.topLeftCorner(3, 3) = rotation3;
```

作业2

1. 判断点在向量的左右位置：math.stackexchange.com/...

2. 关于叉积的计算方法和物理意义： https://betterexplained.com/articles/cross-product/
3. 在三角形内部的判断方法（对于点P 和三角形 ABC来说）：
    1. 方法1： 比较常用的是依次判断点P 和 AB、BC、CA 三条边的位置关系，如果P点均在三条边的左侧或者右侧，则认为点P在三角形内部；
    2. 方法2：根据重心坐标计算 alpha, beta, gamma，如果三者均不小于0且和为1 则认为点P在三角形内部；
    3. 方法3：通过计算三个三角形的面积，即： PAB, PBC 和 PCA，如果三个三角形的面积之和与 ABC 面积相同，则认为点P在三角形内部，不过这种方法依赖浮点数的相等计算，可能会有精度问题。可以参考： youtube.com/watch?...
    4. 方法4：假设ABC三个定点逆时针排列，计算AB逆时针旋转90度的方向AB ‘，然后判断 AP 和AB’ 的点积是否大于等于0，即同在AB的左侧；然后依次计算三角形另外两边。
    5. 方法5：作业3的框架中实现了一种新的方法，有些类似于方法1，在点与边的位置关系判断时，使用了P点与第三点是否在边的同一侧的判断方式。（感谢同事给出的解释）
    6. 方法2 和 方法4 在 totologic.blogspot.com/... 中有详细介绍和实现，并且作者还提到了另外的一种参数化投影的方法，不太好简要描述，有兴趣的可以去看一下。
    7. stackoverflow.com/quest... 这篇讨论中的高赞方法是方法1，另外也有人提到了一些奇奇怪怪的方法，有兴趣的可以去看看。
4. 重心坐标插值的方法在老师讲义的 Lecture 9 P5 页开始，在 P11 给出了最终的计算方式，其实通过 P9 的面积比也很容易计算。面积的计算方法可以看上面叉积相关的链接。
5. 如果按照作业1的方法修改了 zNear 和 zFar，得到的 depth 其实和老师作业中说明的不一致，虽然都是正数，但是较大的数字离摄像机更近，需要调整判断深度的计算方法。
6. SSAA 的实现比较简单，其中可能存在的问题是对于 Index 的计算方法。如果扩大了以后的 buffer 计算有问题的话，可以把viewport 大小同样调成两倍大小，然后把SS后的Buffer 直接显示出来，这样比较容易发现 Buffer 中的问题。

作业3

1. 官方论坛上的作业三常见问题（非常重要）：
    1. bump mapping 部分的 h(u,v)=texture_color(u,v).norm, 其中 u,v 是 tex_coords, w,h 是 texture 的宽度与高度
    2. rasterizer.cpp 中 v = t.toVector4()
    3. get_projection_matrix 中的 eye_fov 应该被转化为弧度制
    4. bump 与 displacement 中修改后的 normal 仍需要 normalize
    5. 可能用到的 eigen 方法：norm(), normalized(), cwiseProduct()
    6. 实现 h(u+1/w,v) 的时候要写成 h(u+1.0/w,v)
    7. 正规的凹凸纹理应该是只有一维参量的灰度图，而本课程为了框架使用的简便性而使用了一张 RGB 图作为凹凸纹理的贴图，因此需要指定一种规则将彩色投影到灰度，而我只是「恰好」选择了 norm 而已。为了确保你们的结果与我一致，我才要求你们都使用 norm 作为计算方法。
    8. bump mapping & displacement mapping 的计算的推导日后将会在光线追踪部分详细介绍，目前请按照注释实现。
2. Payload 的含义：简单理解就是在进行 Shading 的时候，打包传给 Fragment Shader 的数据合集，这些数据用一个 struct 组合在一起，减少了入参的个数。
3. Payload 内部成员中的 view_pos 表示当前 Shading point 在 View Space 下的坐标，也是通过插值得到的。不过这个命名并不好，容易跟视点坐标搞混。
4. 向量操作除了常见问题中列举的，还有一个是 squaredNorm()。
5. 实现顺序按照老师作业说明中示例图片的顺序实现即可，基本上后者对前者都有一定的依赖性：
    1. normal 最简单，用来确认模型加载及显示都没有问题；
    2. 然后是 phong 可以确认光照及反射是否实现正确；
    3. Texture 可以检查纹理加载以及采样的正确性；
    4. bump 主要依赖切线空间的代码实现；
    5. displacement 综合了 bump 和 phong，可以看到光照下的渲染结果。
6. 代码 rasterizer::draw() 函数中，将 normal 向量转换到 View Space 使用了 inv_trans = (view * model).inverse().transpose()，具体原因详见 虎书 4th 6.2.2 Transforming Normal Vectors。参考来源： http://games-cn.org/forums/topic/guanyuzuoye3-displacement-mappingdengdewenti/
7. 框架代码中计算TBN时，使用了一个简化的方法，就是直接根据 normal 计算出来一个 tangent，可以参考官方论坛上这个帖子、这个帖子和这个帖子的讨论。帖子中有同学解释了框

架代码的计算原理，也有人简要给出了一个来源，但是我还是没有完全理解这种算法的合理性。正常的计算TBN的方法是根据UV方向进行处理，这篇英文教程讲的非常详细，而且在最后还简单介绍了使用 Gram-Schmidt process 重新正交化 TBN 矩阵的方法。论坛同学提到这个这篇中文文章在直观上的解释很不错，但是具体推导过程比如前一篇英文教程来的通俗，可能是因为一直使用语言描述而没有用图形的方式来展示，两篇文章可以结合着来看。

8. 计算Tangent Space 下的法线时，框架要求 乘上 kh 和 kn 两个系数，但是这两个系数都没有解释，论坛上也有同学在问，然而却没有回答。

作业4

1. 实现 Bezier 的递归方法较为简单，建议测试的时候可以缓存一次的控制点的结果，不用每次去点击屏幕，这样测试起来方便很多。
2. AA的实现方法，老师在作业说明里已经写明了，按照当前坐标点的位置，计算周围整数坐标点的颜色值，有点反向双线性插值的感觉。需要注意的是，在和已经保存的颜色值比较时，需要取Max，而不是直接覆盖或者取平均。
3. 还有一点需要注意的是 $d = (x - x_0)^2 + (y - y_0)^2$ 时，d 的取值范围是 [0,2)，我在这里犯了错误，导致AA以后的图像一直有问题。

作业5

1. 从作业5开始了一个新的系列，不再使用 Eigen 和 OpenCV，这也让编译简单了不少。
2. 选择成像平面时，选择 z=-1 即可。
3. 由于不再使用 OpenCV，所以生成的图像才用了一种基于文本的图像格式 PPM，简要介绍可以看这篇文章： blog.csdn.net/kinghzkin...
4. 在 rayTriangleIntersect 函数中，几个参数有些晦涩，解释一下：
    1. v0, v1, v2 表示三角形的三个顶点，非常直观
    2. orig 和 dir 表示 光线的原点和方向
    3. tnear 其实是返回参数，是相交计算后得到交点的 t 值（P = O + Dt）
    4. u, v 是两个计算得到的重心坐标参数，
    5. 参考闫老师讲义 PPT Lecture 13 P29 的 Möller Trumbore Algorithm，其实 t, u, v 就是结果中的 t, b1, b2
5. rayTriangleIntersect 中计算得到的重心坐标参数，最终在 MeshTriangle::getSurfaceProperties() 中使用了，即传入参数中的 uv，用来插值计算这个点的纹理坐标。虽然最终确实是用来计算uv的，但这个传入的参数一路起名为uv，也是奇怪的，如果不是跟着代码下来，恐怕对这个起名也是一头雾水。
6. 场景里一共只有3个物体，一个 DIFFUSE_AND_GLOSSY 的 Sphere，一个 REFLECTION_AND_REFRACTION 的 Sphere，还有一个两个三角面组成的 DIFFUSE_AND_GLOSSY 的 MeshTriangle。
7. 这地板纹理颜色的计算很有意思，我本来以为是一个纹理贴图，然而不是。有兴趣的可以看看 MeshTriangle::evalDiffuseColor() 这个函数。
8. 一些C++的知识点（发现助教同学们真的喜欢用C++的新特性啊）
    1. 在 castRay() 函数中 使用了 C++17 的增加了初始化语句的新 if 语法，即 `if (auto payload = trace(orig, dir, scene.get_objects()); payload)` ，可以参考 en.cppreference.com/w/c... 进一步了解。
    2. 在 main() 函数中，创建各种 Object 使用到了 unique_ptr （C++11） 和 make_unique<>() 方法（C++14），简单来说，unique_ptr 主要目的是为了保持指针指向的对象所有权的唯一性，关于 unique_ptr 的简单使用可以参考 blog.csdn.net/shaosunri... ，关于两者之间差别可以参考 stackoverflow.com/quest...
    3. 在 trace() 函数中使用了 std::optional<hit_payload> 作为返回值，简单来说 std::optional （C++17）可以看成值类型加上一个指示是否初始化的标记位的结合体。如果希望深入了解的话，可以参考这篇文章： blog.csdn.net/hhdshg/ar...

作业6

1. SAH（Surface Area Heuristic）简单来说，是一种按照表面积进行概率估计的启发式BVH分割算法，目的是构建更好的BVH结构，降低BVH遍历成本。
2. SAH 相关说明内容，可以看这篇中文讲解，但是我觉得写的最明白的还是这篇英文文章，另外还推荐参考如下内容：
    1. PBR Book，包含算法实现，非常详细
    2. 关于BVH 的改进，并不是主要关于SAH 的，可以简单参考
    3. CMU 课程，图示非常清晰，有伪码实现，尤其说明了为什么使用 bucket 划分的方式进行计算。
3. SAH 实际实现中，可能会遇到如下问题
    1. t_trav （遍历成本）以及 t_isect （求交成本）的取值问题：在本次作业中影响不大，上面的几篇文章中大部分都给了取值的例子，可以参考使用。
    2. bucket 个数取值问题：bucket存在的意义是为了减少分割位置的计算，使用定长步长的分割线来替代Object边界位置的计算，一般来说不超过 32。过大的 bucket 数目会增加Cost 计算的成本，过小的 bucket 会使 SAH 算法退化到原始 BVH 分割算法。
    3. 一般来说，SAH构建到只剩少量Object时，可以直接 NAIVE 方法，可以避免过度计算。
    4. 下面是我使用的一些常量，仅供参考
        1. static constexpr int SAH_MIN_OBJECT_COUNT = 4;
        2. static constexpr int SAH_BUCKET_COUNT = 32;
        3. static constexpr float SAH_INTERSECTION_COST = 2.0;
        4. static constexpr float SAH_TRAVERSAL_COST = 1.0;
4. 理论上来说，SAH构建较为复杂，但是遍历时会有性能提升，但是作业框架中的模型较为简单，差异并不明显，可以选择一个较高面数的模型进行测试。我计算的结果构建耗时增加 20%，遍历耗时减少 11%，不过这也与算法本身实现以及编译优化有一定的相关性，论坛上也有人在讨论相关的性能变化情况，可以参考比较。
5. 框架中，一共构建了两次BVH，第一次是在初始化 MeshTriangle bunny 的时候，第二次是在调用 Scene::buildBVH() 的时候。bunny 中的 BVH 才是需要考虑的部分，Scene 中创建的 BVH 只有一个 Object，那就是 bunny。
6. 在实现 Bounds3::IntersectP() 函数时，框架中给出的参数有些奇怪，虽然在框架中以注释的形式给出了简单的说明，但是在使用上还是遇到了一些问题，这里解释一下
    1. invDir： 相对好理解一些，是 ray direction 各个分量的倒数，其实使用 ray.direction_inv 就可以得到，这里特意列出来可能是为了强调。主要目的是将除法运算转化为乘法运算，这样计算的时候能够快一些。
    2. dirIsNeg：这个参数比较迷惑，我也思考了好久。论坛上也有同学在问这个问题，但是没有有效的解答。其实这个参数是 direction is negtive 的缩写，表示 ray 的三个方向的分量是否向负轴方向，然而按照这个变量的定义来说，其实这个变量应该命名为 dirIsPositive才对。在使用上，参考闫老师讲义 Lecture 13 P36-P39 的内容，可以看到光线与 AABB 求交，其实就是对于 6个平面的求交计算，如果光线某个分量为负轴方向，则在这个分量上计算出来的 t_min 和 t_max 应当交换，此时就可以用到这个 dirIsNeg 来进行判断是否需要进行交换了。在我来看，除了上面的命名问题以外，这个变量应当以 Vector3f 的类型给出，而不是用 std::array。
7. 一些 C++ 的知识点
    1. std::array：（C++11）用来取代C数组的数据结构，用法与普通数组相同，但是提供了高级访问操作，简要介绍可以参考这篇文章，和数组的区别可以参考这篇关于 array、数组和vector的区别的文章。
    2. std::chrono：（C++11）是一套时间相关的库，来源自 boost，使用方法可以参考这篇文章。
    3. std::random_device：（C++11）一套新的随机数库，比原来的 rand() 高级了很多，可以先参考这篇基础入门文章，再深入了解可以继续看这篇原理性的文章。
    4. std::numeric_limits<>：（C++11）定义了基础数据类型的一些极值信息，取代C风格的宏常量定义，具体可以参考这篇文章。

作业7

1. 作业中 intersectP() 方法中，要注意在判断 t_enter 和 t_exit 时，需要使用 t_enter <= t_exit && t_exit > 0（在作业说明中其实有提到这个问题）。我在开始的时候没有遇到这个问题，是因为我判断相交使用的是我自己的算法，没有按照老师课件的方法实现。后来迁移作业6的算法时，遇到了画面全黑的问题，在论坛上查看其它同学的问题才发现是这个原因。论坛上Chen同

学写的这篇踩坑整理建议大家都看一看，如果有些坑看不懂，先放过去回头遇到了再来看，总有一天会看懂的。

2. 提高题中多线程实现方案，可以使用 std::thread 方案，也可以使用 OpenMP 方案。前者比较自由，可以自己控制线程的划分方法，后者改动比较少，但是得花时间学一下 OpenMP 的使用方法。论坛里也有同学使用std::future方案，个人感觉不太适合当前的这种计算密集的应用场景，所以不做推荐。下面分别简要介绍一下前面两者的使用方式：
    1. std::thread
        1. 注意多线程不要按照 spp 划分，因为像素数目已经足够运行在多个线程上了。而且对于一个像素来说，多个 spp 得到的结果需要写到同一个 frame_buffer 中，这样就需要对 frame_buffer加锁处理，会降低运行速度。
        2. 线程不要频繁创建销毁，这样带来的开销会很大。可以做成线程池，每次有任务就使用现有的线程运行，当没有线程可以使用的时候，就等待线程完成，直至所有任务都完成。
        3. 提供一个简单的抢占式实现思路：启动后创建与CPU核数相同的线程数，每个线程运行的函数都是一个while循环，在循环里 GetJob （注意加锁）获得一个像素点的坐标，然后计算这个像素点的坐标，直至无法获得新的Job时退出循环。主线程开启所有线程池以后，就开始等待所有线程结束，当Join了所有线程以后，表示计算完毕，开始写入图像。这样做的好处是按照像素划分线程，frame_buffer 不用加锁，抢占式的方法可以保证只要有Job的时候，所有线程都不空转，能够把CPU完全可以吃满。
    2. OpenMP
        1. 在 Linux 下安装可以使用 sudo apt install libomp-11-dev 安装 OpenMP 的开发库，在 Mac 下可以使用 brew install libomp 安装，但是编译时会出现问题，待研究。Windows 下开发可以参考这篇使用Visual Studio的文章。
        2. 可以参考这篇入门文章，按照其中的例子3进行修改即可，不过要注意并行化以后的for里面，不能存在必须串行化的计算，例如我遇到的问题是计算 frame_buffer 下标时，使用了 m++ 的方式，必须修改为按照 i,j 直接计算的方式，也即for里面的内容不能有相关性，否则并行计算时乱序计算会导致问题。
        3. 修改代码后，需要调整CMakeList.txt ，加入如下内容就可以编译使用了。

```
find_package(OpenMP)
if (OPENMP_FOUND)
    set (CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${OpenMP_C_FLAGS}")
    set (CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenMP_CXX_FLAGS}")
    set (CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS} ${OpenMP_EXE_LINKER_FLAGS}"
endif()
```

1. 框架随机数问题：论坛上thistl提到了Mac下随机数的问题，也就是这篇文章提到的问题，里面介绍的问题非常有趣。简单来说，在 Linux/Unix 上使用 random_device 时，会使用操作系统提供的熵池，当熵池耗尽以后，random_device 会将调用阻塞，直至有足够的熵可以使用为止，这也就导致了 random_device 是个非常耗时的操作，一般用来生成随机数种子。然而在作业框架中，被频繁调用的 get_random_float() 方法却一直在调用 random_device，造成了整体性能下降。修改方法也很简单，把 get_random_float() 里面前三行都改为 static 变量，或者改为全局变量也可以，总之在每次运行只需计算一次就够了。这一个小小的改动对性能用了巨大的影响，在 spp=4的情况下，改之前我的Mac 上运行时间为 204s，修改以后仅仅使用 4s 就完成了，也就是说**有了50倍的速度提升**。
2. 如果需要先了解一下 Microfacet 的基础知识，可以看一下知乎上文刀秋二的这篇文章，图文并茂讲的比较清晰。Microfacet 的实现方法，参考这篇 LearnOpenGL 的文章就足够了，前面的基础部分可以跳过去，从 BRDF 小节开始看起。文章中不仅介绍了基础知识，还附带了 OpenGL 的代码实现，可以轻易改成C++语言的实现。如果需要深入理解 Microfacet，可以参考毛星云的基于物理的渲染（PBR）白皮书，或者从知乎访问这一系列的文章。常用的 D 和 G 函数也可以参考 Epic 的 Brian Karis 做的相关整理文章。（2021年8月17日 更新）闫老师在 GAMES202 第10课中详细讲了关于 Microfacet 的算法，强烈建议先看这一讲的视频，有 GAME101 基础的话基本都可以听懂。
3. 一些在实现 Microfacet 时需要注意的问题：
    1. Ks 是 Specular 系数，其实就是 DFG 中的F，也就是菲涅尔项。
    2. 框架中定义的 Kd 其实是 Color，在 Microfacet 中，Kd 的值应该是 1-Ks。
    3. 最后得到的 fr = Kd * fr_diffuse + Ks * fr_specular。也就是要分别计算漫反射部分和高光部分，所以不要只顾计算 DFG 而忘了前面的漫反射部分。
    4. 推荐看一下论坛上关于 Microfacet 讨论的这篇帖子，里面提到了很多典型问题，非常有启发性。

5. （2021年8月17日 更新）**重点**： 在GAME202 第10讲的最后，闫老师批驳了这种使用 Kd 做光照损失补偿的计算方法，称其没有任何物理原理可言。使用 Microfacet 方法会导致物体变暗，尤其是 Roughness值较大的物体会变得很暗，主要原因是因为在计算时没有计算多次光线弹射导致的能量损失。闫老师在 GAMES202 中介绍了 Kulla-Conty 方法可以用于在实时渲染中进行能量补偿，可以参考。B站评论上有人说使用环境光进行补偿的方式是工业界经常使用的一种近似计算方法，不过个人感觉既然是在做学术界的作业，还是不要使用这在学术界上没有意义的计算方法。

4. Microfacet 在 Roughness 较小时，会产生噪点问题，在论坛的这篇帖子里有所讨论，
5. 重要性采样理论入门可以看这篇Scrachapixel上的文章，对重要性采样的原因解释的比较清楚，并且给出了一个清晰的案例，但是没有提供方法。

作业7.5 重要性采样

1. 本来这里的内容是写在作业7里的，然而越写越多，只要拿出来单独拆分一个小节了。重要性采样与作业7的关系不大，并且在作业7的说明中，写明了不用修改sample() 和 pdf() 两个函数，其实在我的理解中就是无需实现重要性采样的意思。
2. 重新理解DFG，建议读一下上面提到的毛星云的基于物理的渲染（PBR）白皮书，尤其是其中第四章和第五章。简单来说 DFG 三个部分各有各的意义。其中F项是变化最小的，一般来说讨论的较少，按照公式实现即可；D项是其中的核心，定义了使用的概率分布函数，所以重要性采样与D项的关系最为紧密；G项描述的是在D项定义的基础上，需要使用的遮蔽函数，主要目的是维持整个DFG的能量守恒。
3. 刚开始看重要性采样（Importance Sampling）的时候，找到了一堆网上的资料，然而网上的资料似乎都是为了有基础的人准备的，不是推导概率公式就是在贴代码，看了一轮以后毫无头绪，既不知道Why，也不知道How。挣扎着反复看了几遍，再加上再作业中尝试实现一下，目前得到了一个看似正确的结果。所以结合我自己对重要性采样的理解，为刚接触这个算法的新手整理了一些关于重要性采样的问题，以自问自答的形式展现。如果我这里的理解有任何的问题，欢迎在讨论区指出，感激不尽。
    1. 为什么需要重要性采样?
        1. 简单来说就是，以平均分布的形式进行采样不能满足蒙特卡洛算法在Specluar项上的要求。粗暴的增加采样点虽然可以得到改善的结果，带来的开销却是不可承受的，所以需要选择更加有效的采样形式来解决问题。
        2. 中文解释可以参考这篇知乎文章，这篇文章的后半部分突然转向了概率公式，对于新人来说可能会感觉云里雾里；
        3. 英文文章建议参考这篇Scrachapixel上的文章，对重要性采样的原因解释的比较清楚，并且在文章的最后给出了一个清晰的对比案例方便直观感受重要性采样的效果。
    2. 为什么实现 Microfacet 前不需要重要性采样?
        1. 因为框架中，原始材质都是 Diffuse 的，也就是说都是只有漫反射项的材质，对光照单独采样后，再对周围环境使用平均分布采样已经可以满足需要了，就不需要重要性采样了。而且对于平均分布的重要性采样就是平均分布，没有任何区别，处处都重要就是处处不重要。
    3. 重要性采样的理论基础是什么?
        1. 其实上面 Scrachapixel 的文章已经解释了，在进行采样的时候，要按照与被积函数一致的概率分布去采样。
        2. 另外就是关于 $D(m)$ 的约束： $D(m)cos(\theta_m)$ 对 半球面的积分为1，所以可以得到关于随机变量m的概率分布密度即为 $D(m)cos(\theta_m)$ ，这也是所有公式推导的基础
    4. 对谁进行重要性采样?
        1. 刚开始看重要性采样相关文章的时候，几乎每篇文章都是从 NDF 开始说起，导致我一直以为是对NDF进行采样（谁让NDF里面也带个Distribution呢）。其实从NDF开始说起的原因是因为 NDF 里面定义了整个 DFG 使用的概率分布，一般常用的有 GGX、Blinn 和 Beckmann，所以在采样的时候，需要按照 NDF 中定义的概率分布进行重要性采样。
    5. 重要性采样的方法有哪些?
        1. 其实这个问题的问法并不准确，这里的“方法”其实指的就是上述三种概率分布，对应的不同计算方式。三种方法原理都是一样的，只是因为概率分布不同导致的公式推导不同而已。推导方法在这篇 Sampling microfacet BRDF 文章已经写明了，而且在这篇知乎文章上也有推导，而且这一篇中对 Blinn 的重要性采样推导更清晰一些，推荐优先阅读。
    6. 如何在作业中实现重要性采样?
        1. 回到作业框架中，关于 BRDF 有三个核心函数，分别是 sample(), pdf() 和 eval()。
            1. **注意（重要）**： 框架中实现的三个函数， ω_i 和 ω_o 的定义与一般文章的使用的相反（这也在作业说明中提到了），**一般来说 ω_o 表示视点方向，ω_i 表示光照方向（本文**

中使用这种表述方式），但是在框架给出的函数定义中却不是，在使用中要尤为注意；
2. eval() 是计算 BRDF 中的 f_r 项，当使用 Microfacet 时，eval() 里面计算的就是 DFG；
3. sample() 是采样函数，即根据法线 N 和视角方向 ω_o 在半球面上随机选取一个入射方向 ω_i ，随机变量分布服从均匀分布（可以从get_random_float() 中得知）；
4. pdf() 是概率密度函数，即根据法线 N 、视角方向 ω_o 以及sample() 中采样获得的入射方向 ω_i 计算一下采样得到 ω_i 对应的概率密度。

2. 那么这样就逐渐清晰了起来，根据上面的知识，再结合这篇 Sampling microfacet BRDF 文章来理解公式，可以知道：
1. 如果根据上面的 OpenGL 的文章实现的 DFG，那么 eval() 中 NDF 使用的便是是 GGX 分布，为了能够更加准确的采样，就需要根据 GGX 分布进行采样和计算概率密度。
2. 关于 sample()：
1. 我们使用 get_random_float() 得到的随机变量服从均匀分布（与文章中的 ϵ 对应）并不符合要求，所以需要转化为符合 GGX 分布的变量（与文章中的 θ 对应）。按照文章中的公式就可以根据均匀分布的随机变量得到一个符合GGX分布的随机变量 θ 来计算 ω_i 。
2. 由于计算的是isotropic（各向同性）的材质，所以另一个随机变量 ϕ 符合均匀分布，就不用进行重要性采样了。
3. 使用 θ 计算得到的 ω_m （ ω_m 表示微表面的法线方向，上面的文章中有说明）是 Local Space，后续计算需要使用 World Space，注意转换。
4. 在进行球面坐标到笛卡尔坐标转换时，需要注意法线所在的轴向，从 sample() 原来的代码可知法线轴向是Z轴，所以 $z = cos(\theta)$ 。在有些文章中使用的Y轴，这也是我在实现时遇到的一个问题。
5. 在 pdf() 中，我们需要计算与 sample() 中采样对应的概率密度，根据上文的描述，可以发现概率密度就是 $D(m)cos(\theta_m)$

3. 综合来说，可以参考这篇 StackExchange 上的回答，他给出了三个函数的实现，但是他的实现中有三个问题，前两个是上文提到的本地坐标系和法线轴向的问题，第三个是他使用了 $roughness^2$ 作为 α ，这与我们一般的认知不符。

7. 不同的文章使用的符号一般都略有差别，所以在这里简单整理一下一般符的使用方法：
1. 视点方向，向量，一般由 Shading Point 指向 Eye Point，常用符号： V, v, ω_o
2. 光照方向，向量，一般由 Shading Point 指向 Lighting Point，常用符号： L, ω_i
3. 法线方向/宏表面法线方向，向量，表示宏表面在 Shading Point 位置的法线方向，常用符号： N, n, ω_g
4. 半程向量/微表面法线方向，向量，表示微表面在 Shading Point 位置的法线方向，常用符号： H, h, m, ω_m
5. 概率密度/PDF，标量函数，常用符号 $p(\theta)$ ，也就是小写字母表示
6. 累积概率密度/CDF，标量函数，常用符号 $P(\theta)$ ，也就是大写字母表示
7. 球面极坐标与法线的夹角，标量，常用符号 θ
8. 球面极坐标另一个方位角，标量，常用符号 ϕ
9. 符合平均分布的随机变量 ϵ ， ξ 分别对应 θ 和 ϕ

8. 扩展阅读资料
1. Importance Sampling techniques for GGX with Smith Masking-Shadowing: Part 1
2. Importance Sampling techniques for GGX with Smith Masking-Shadowing: Part 2
3. How Is The NDF Really Defined?

作业8

1. 总体来说作业8比较简单，没有太多的问题。环境配置上，作业8增加了OpenGL的使用，如果使用 WSL2 环境的话，注意安装需要的开发库和运行时库，并且注意在MobaXTerm下显示GUI的问题。具体配置和使用方法可以参考我的 在 Win10 下配置 GAMES101 开发环境（WSL2）一文。
2. 在完成作业之前，建议看一下论坛上助教给出的解释，免得踩一些不必要的坑。
3. Euler 方法 和 Impicit Euler 方法都比较简单，按照作业说明或者闫老师讲义上的方法实现即可。
4. 框架中注释里写的 Add global damping 写在了计算 velocity 和 position 的下方，我感觉有些误导，我实现 damping 计算的方法其实是在计算加速度之前。
5. 按照助教给出的方法实现 Verlet 方法时，全局获得的力只有各个质心受到的重力，质心之间的

作用使用距离约束计算，不需要计算弹力。

6. 要注意在遍历每个质心的时候，第一个质心是被固定住的，无法运动，其他的质心按照循序依次处理即可。

（全文完）

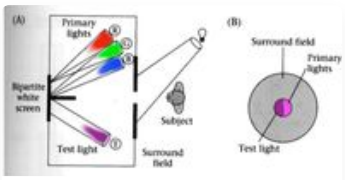
编辑于 08-17

计算机图形学 实时渲染 游戏开发

文章被以下专栏收录

GAMES系列课程作业与笔记

推荐阅读



GAMES-101 Lecture 20 笔记-下

木子 发表于木子君的读...

GAMES202 作业1框架代码结构分析

b1gm0... 发表于GAMES...

1 条评论 切换为时间排序

写下你的评论...

YXH咸鱼 06-16

帮大忙了😘

2