

《计算科学导论》课程总结报告

|  |  |
| --- | --- |
| 姓 名 | 刘海政 |
| 学 号 | 2207010108 |
| 专业班级 | 计科2201班 |
| 学 院 | 青岛软件学院、计算机科学与技术学院 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 课程认识  30% | 问题思考  30% | 格式规范  20% | IT工具  20% | 总分 | 评阅教师 |
|  |  |  |  |  |  |

2022年11月25日

# 1 引言

作为计算机专业的学生，注定一生都要与计算机打交道，而与计算机打交道不仅需要知道计算机怎么用，更应该从计算机的起源去了解计算机，了解计算机的发展历史，了解计算机的构成，了解计算机形成所蕴含的一系列知识，其中包含一系列复杂庞大的数学知识，同时计算机各硬件如芯片等的制作流程也十分繁杂，而计算科学导论这门课程便为我们学习计算机的学生提供了这么一个全面了解计算机的机会，让我们能够充分的了解计算机，了解计算机发展所包含的各种数学知识或是工程知识，让我们不仅会用计算机，更是了解计算机，懂计算机。

# 2 对计算科学导论这门课程的认识、体会

总体说明你的整体认识，再举一、二个例子，从某个角度进一步展开讨论，以支持你的认识。

课程最开始我们学习了**《计算科学导论》[1]**这本书,首先从哲学方面了解了“计算科学”一词的来源，其包含狭义广义两部分，狭义便是指计算机科学与技术，而广义上的计算科学则包含计算机科学与技术，除此之外有更多内涵。

通过对这本书的学习，我初步认识了对计算机形成做出杰出贡献的人才，如图灵、冯·诺依曼等等，了解名人的事迹激发我对计算机这一事物起源的好奇心，也开阔了我自身的知识面，同时通过对计算机发展历程的学习，我了解了历史上曾出现的四代计算机，从电子管计算机到晶体管计算机到集成电路计算机再到大规模和超大规模集成电路计算机，物理元器件不断更新的同时带来了计算机性能的跨越式提升，带来了计算机功能的日益丰富，对计算机之外的各学科革新都产生了巨大影响，如仿真系统的逐渐完善使得现在如核武器研制等可以通过计算机建立物理模型进行相对准确、可靠的模拟，不仅节省了物力，减少了环境污染，而且大大提高了核武器研制的效率。

课程上简单了解了图灵机，而后了解了人工智能、高级语言等。特别对机器指令与汇编语言的讲解让我记忆尤深，我首次了解到通过语言让机器听从指令的过程，了解了机器转换语言为指令的过程，也看到了如今语言相比早期直接输入机器指令所具有的快捷性及效率之高，激发了我对语言学习的热情。通过对计算科学导论的学习，我极大的丰富了自身掌握的知识，学习了部分软件基础、计算机应用基础等等，尤其对人工智能方面的学习激起了我的兴趣，更是第一次确切相信“真正的人工智能不可能实现”。

课程上讲述了范式，让我们了解到一个合理的科学知识系统的逻辑结构具有一元性、简单性、内部一致性、外部相容性、稳定性五大性质，同时结合科学哲学的讲述我浅略阅读**《科学革命的结构》[2]**一书对科学范式有了更深的了解。

课程上讲述了很多数学家的故事及其贡献，如丢潘图方程、希尔伯特23问，对布尔代数进行了初步了解学习，通过对数学家事迹的了解，及数学家对计算机发展做出巨大贡献的了解，我意识到了数学学习对学习计算机的重要性，计算机各种功能的实现离不开数学，离不开逻辑，由数学学习所锻炼出的逻辑思维对计算机你学习尤为重要。

《计算科学导论》书中曾提到过“我们说在通才教育观下，第一流的专业人才应该具备三个条件：（1）具有高尚的品德和良好的人文素养。（2）具有坚实的专业基础和身后的专业功底。（3）富有创新意识，具有科学的思维方法。”这一番论述对我个人日后发展的方向有极大的启发，我第一次完整的了解到一个专业人才所应具备的基本素养，对我日后学习具有重要启发意义。

在我个人看来，计算科学导论这门课程具有如下特点：

1. 概括性强。如人工智能、数据库、软件开发等多方面知识均有所提及，极大的丰富了学生的知识面。
2. 严谨。其中对数学家的讲解不止停留在数学家本身，更是对数学家所提出的相关数学知识进行了不失本意的严谨教学，让学生集体了解数学家，严谨学习数学知识。
3. 有发展思维。课程不仅对计算机知识进行了授课讲解，更是对计算机知识外的很多事情如计算机发展代数、人工智能浪潮、芯片制作流程等多方面事务进行了讲解，让学生不仅了解课程知识，更扩展了知识面了解到了一些有趣的知识。

## 2.1 人工智能初入门

通过**《计算科学导论》**一书，我对人工智能有了更深的了解，从1956年第一次提出人工智能这个概念至今已经走过了三次人工智能发展浪潮，出于对人工智能的好奇我开始了解人工智能。

我初步阅读了**《深度学习 - [智能时代](https://www.zhihu.com/search?q=%E6%99%BA%E8%83%BD%E6%97%B6%E4%BB%A3&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":582707329}" \t "https://www.zhihu.com/question/_blank)的核心驱动力量》[3]**,作者以亲历者视角回溯了深度学习浪潮在过去60年间的发展脉络与人工智能的螺旋上升，并前瞻性地预测了智能时代的商业图景，在作者的讲解下我大体了解到人工智能是如何进化的以及人工智能会如何演变，让我们对人工智能有了初步较深刻的了解，再次激发我对人工智能的兴趣。

而后我浅略了解了神经元、全连接网络，阅读了**《Wide & Deep Learning for Recommender Systems》[4]**,初步对wide\_deep模型有了较全面的了解，对全连接网络中深层与浅层的关系有了较全面的了解。

而后基于对wide\_deep模型的探讨通过阅读**《Adam: A Method for Stochastic Optimization》[5]**,我对Adam有了较为全面的了解，自此对人工智能所包含的部分基础知识有了较好的理解。

# 3 进一步的思考

通过查询各方面的文献，对实时光线追踪渲染有了更全面的了解。

将光线追踪算法应用于图形渲染最初是由Arthur Appel于1968年提出，最开始名字叫做ray casting，但由于技术硬件等方面限制，此时的光线追踪难以实现，但随着科技的发展，1993年NVIDIA推出Voodoo加速卡1995年微软推出了Direct3D，这使得光栅化渲染更加容易实现，1999年NVIDIA首次提出GPU概念，能够快速实现光栅化渲染，2001年NVIDIA支持Direct3D 8标准，2002年Direct3D 9发布，至此，渲染管线的渲染目标可以不再是帧缓存，可以是自定义纹理。2006年Direct3D 10发布，2010年，Direct3D 11推出，正式为图形渲染引入一条新管线 —— 计算管线（Compute Pipeline），计算管线是独立于光栅化渲染管线的另一条管线，它利用了 GPU 通用计算特性，通过计算着色器（Compute Shader），可以完成一些非图形渲染任务。2011年，AMD推出首款集成显卡，2015年Direct3D 12推出，直到2018年NVIDIA推出搭载加速光线追踪核心RT core的Turing架构，同年Direct3D 12推出DXR，至此实时光线追踪渲染可以在光栅化的基础上以较高的效率实现。

光线追踪的技术起源于人们对在计算机中构建出不逊色于现实世界环境表现的需求，光线追踪技术是由几何光学通用技术衍生而来。它通过追踪光线与物体表面发生的交互作用，得到光线经过路径的模型。简单地说，3D技术里的光线追踪算法，就是先假设屏幕内的世界是真实的，显示器是个透明的玻璃，只要找到屏幕内能透过人眼的光线，加以追踪就能构建出完整的3D画面。

而说到光线追踪就不得不提到光栅化，光栅化是指把景物模型的数学描述及其色彩信息转换至计算机屏幕上像素的过程。使用光栅化，我们可以将几何图形转化成屏幕上的像素。Direct3D使用扫描线的渲染来产生像素。当顶点处理结束之后，所有的图元将被转化到屏幕空间，在屏幕空间的单位就是像素。点，线，三角形通过一组光栅规则被转化成像素。光栅规则定义了一套统一的法则来产生像素。光栅得到的像素一般会携带深度值，一个RGB Adiffuse颜色，一个RGB specular颜色，一个雾化系数和一组或者多组纹理坐标。这些值都会被传给流水线的下一个阶段像素的处理，然后注入到渲染目标。由于实时3D渲染程序要求对用户的即时操作做出迅速反应，因此通常要求每秒至少20帧以上的渲染速率，这也使得高效率的“光栅化”渲染技术成为当今最受青睐的3D即时成像技术。但是光栅化的缺点也很明显，那就是无法计算真实的光线，导致很多地方失真。

光线追踪算法分为两种：正向追踪算法和反向追踪算法。

其中，正向追踪算法是大自然的光线追踪方式，即由光源发出的光经环境景物间的多次反射、透射后投射到景物表面，最终进入人眼。

反向追踪算法正好相反，它是从观察者的角度出发，只追踪那些观察者所能看见的表面投射光。就目前而言，所有3D制作软件的光线追踪算法都是采用反向追踪法，原因是这种算法能够最大程度地节省计算机的系统资源，而且不会导致渲染质量的下降。

在现实世界中射到物体表面上通常有三种情况：折射，反射，吸收。光线在经过反射到不同的地方，被选择性吸收，从而光谱发生改变，再多次反射与折射，最终进入我们自己的眼睛。而光线追踪技术要做的就是模拟这一过程。

在渲染中，光线追踪会赋予每一个像素几条甚至几十条光线，然后光在场景中传播，与场景中各个物体产生交汇，而场景中的物体在事先就已经被开发者设置好了属性，从而导致光束发生各种改变，最终聚集在屏幕上。从而被我们感知。

而反向追踪方式则是计算最终射入我们眼睛的光线的反向光路，即眼睛—物体—光源的过程。因为这么做的话，可以省掉很多并不需要计算的光路，在光线追踪中，并不是每一束光都有用。有些光最终并没有射入我们自己的眼睛，但是如果依旧计算的话就会造成不必要的计算资源浪费。在光线追踪中，光同样也被分类，假设一条主光线是不可见的，那么系统则会抛弃掉整条光路，如果可见的话，那么辅助光线（反射，折射，阴影）的计算就开始了。在完成主光线的判别之后，辅助光线的其他属性（透明度，色彩）操作也已经基本完成了，目前实时光线追踪渲染便是以反向追踪方式来实现。

纵使目前实时光线追踪渲染发展已经较为成熟，但想要再次提高真实性和效率还存在很多难点

1、透明渲染问题

在目前的实时光线追踪领域，在每像素1~2个样本的情况下，大多数降噪技术通常对于[透明渲染](https://www.zhihu.com/search?q=%E9%80%8F%E6%98%8E%E6%B8%B2%E6%9F%93&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":1606439715}" \t "https://www.zhihu.com/question/_blank)、粒子渲染、体积渲染的渲染效果都并不算好。当下业界需要研究出更好的降噪技术或者相关方案，以在较低的每像素采样数下带来更佳的透明渲染品质。这即是业界目前未攻克的难题之一。

2、部分覆盖渲染问题

一般情况下，我们可以在命中着色器中进行[alpha测试](https://www.zhihu.com/search?q=alpha%E6%B5%8B%E8%AF%95&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":1606439715}" \t "https://www.zhihu.com/question/_blank)，但镜头一旦移动，就会遇到性能和表现上的问题。主要原因依然在于目前的降噪技术并不能很好的处理这种实时的部分覆盖在每像素1~2样本下的可见性判定。这也是业界目前未攻克的难题之一。

粒子系统的渲染问题

使用实时光线追踪渲染粒子系统的主要问题在于，粒子一般为朝向相机的公告板，如果进行反射，则会出视觉失真现象,对于光线追踪粒子系统的此类渲染问题，一般的方案是维护两个顶层加速结构（Top Level Acceleration Structures, TLAS）另外，由于固有的动态特性，一些粒子系统必须在加速结构中每帧更新。在少许大型粒子或许多小型粒子上都可能会发生这种更新，因此对于这些粒子系统来说，优化空间加速度结构以快速在整个世界中进行光线追踪则是另一项挑战。所以，实时光线追踪中对粒子系统更真实的渲染，同样是当前未攻克的难题之一。

4、全局光照领域的渲染难题

首先，使用实时光线追踪进行全局光照，会遇到即使在离线渲染中也存在的各类开放性问题。比如离线渲染中暂未解决的过高方差，小孔全局光照（Pinhole GI）等问题，这些在实时光线追踪领域目前同样需要解决。

而且离线渲染方案中的许多解决方案，不一定都可以运用到实时渲染中。对于实时光线追踪，目前而言必须借助缓存技术摊销着色成本，来达以交互速率进行渲染的性能要求。所以基于实时光线追踪技术的全局光照，目前仍然有大量的难题待解决，需要业界共同努力去攻克。

5、一致性与非一致性难题

一般来说，更好的处理一致性问题是提升实时光线追踪性能的关键。相邻的光线可以执行类似的操作和内存访问，从而带来较好的性能表现，而有些光线可能会破坏缓存的一致性，从而影响整体性能。

但从反过来说，光线间的不一致性却可以提高蒙特卡洛方法的收敛速度。

可以将这个棘手而两难的问题归纳为理论与系统，或者数学与工程间的碰撞。而实时渲染中最重要的问题，正是协调、权衡与处理这种脱节。

6、其他

另外，也存在一些目前已有一定进展，且仍然需完善的通用领域，可以将它们分为调度问题（Scheduling）、解耦问题（Decoupling）、整体性渲染问题（Holistic Rendering）三大类：

调度问题（Scheduling）

SIMD一致性（SIMD coherence）

数据缓存一致性（data cache coherence）

指令缓存（Instruction cache）

占用率（occupancy）

磁盘即时流式传输（just-in-time streaming from disk）

解耦问题（Decoupling）

重用跨路径与跨帧的中间结果（Reuse of intermediate results across paths and frames）

整体性渲染问题（Holistic Rendering）

准蒙特卡罗与多重重要性采样协同感知（Cooperative perceptual QMC/MIS）

多光源采样（many light sampling）

预滤波（prefiltering）

降噪（[denoising](https://www.zhihu.com/search?q=denoising&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":1606439715}" \t "_blank)）

而以上各类细分领域当中，都有大量悬而未决的问题需要业界共同解决。

目前NVIDIA可以实现光线追踪的显卡：

1、RTX20系列30系列40系列显卡都支持光追。

 2、GTX16系列目前有两款型号可以支持光追，分别是GTX1660和GTX1660Ti。  
  3、GTX10系列部分显卡可以支持光追。简单的说10系列显卡只有GTX1060 6G以上型号才支持光追，例如：GTX1060 GTX1070 GTX1070Ti GTX1080 GTX1080Ti。

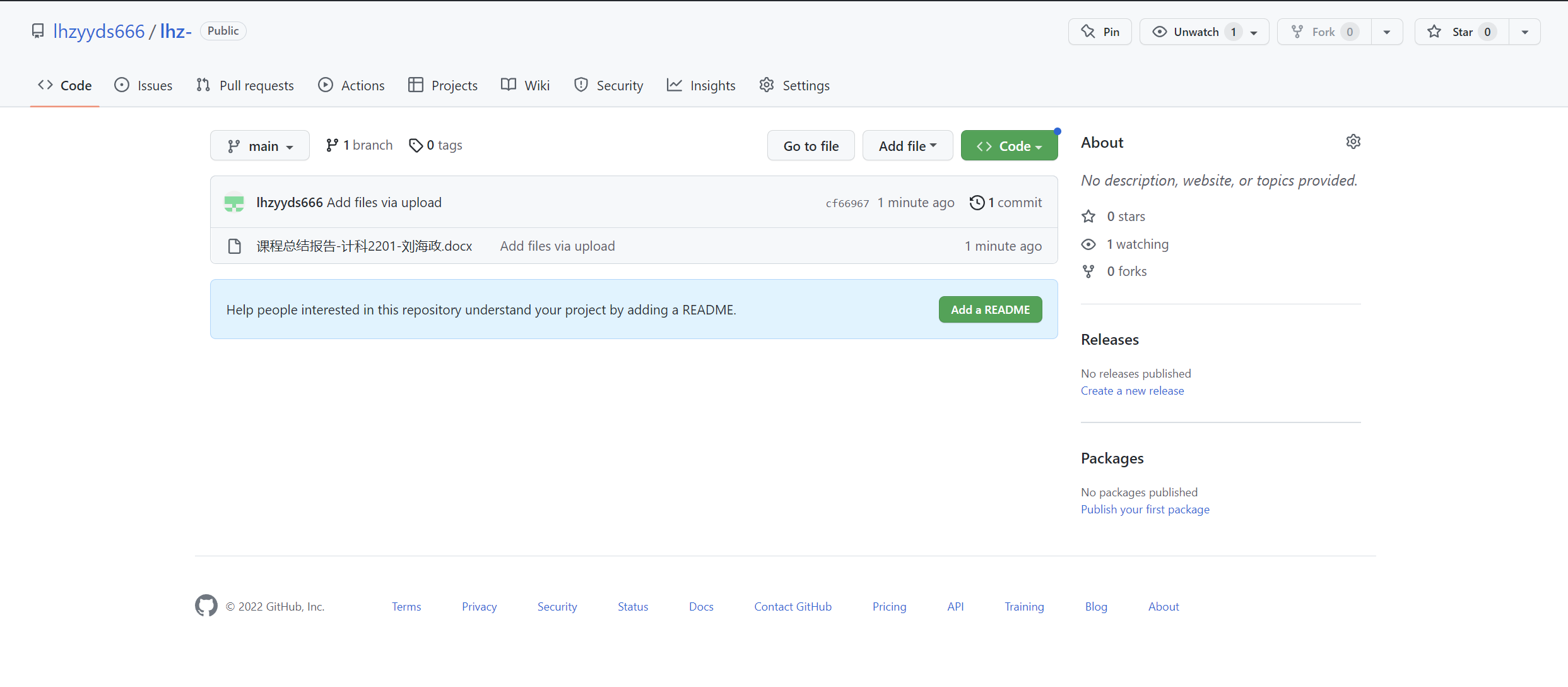
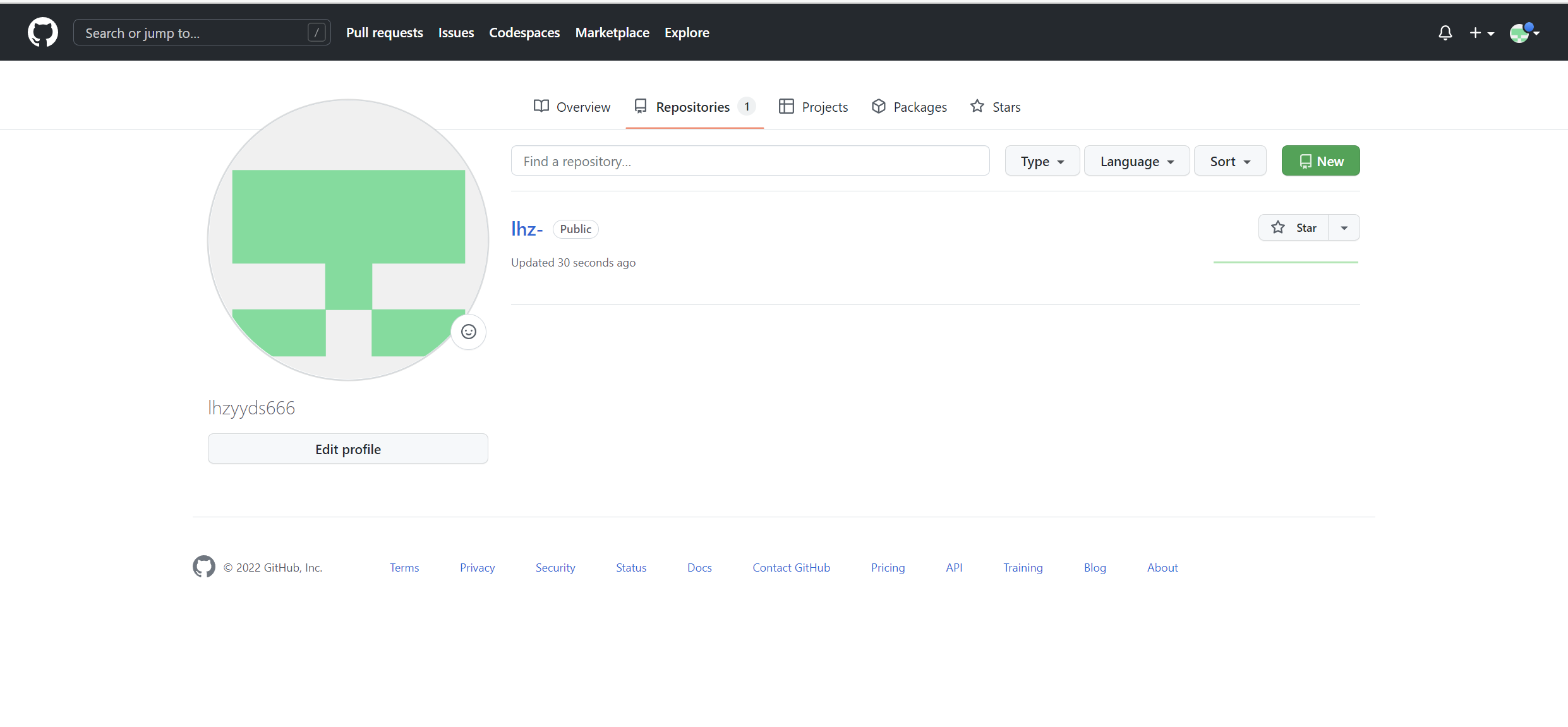
光线追踪在硬件效率排序为40系>30系>20系列>16系列>10系列.

# 4 总结

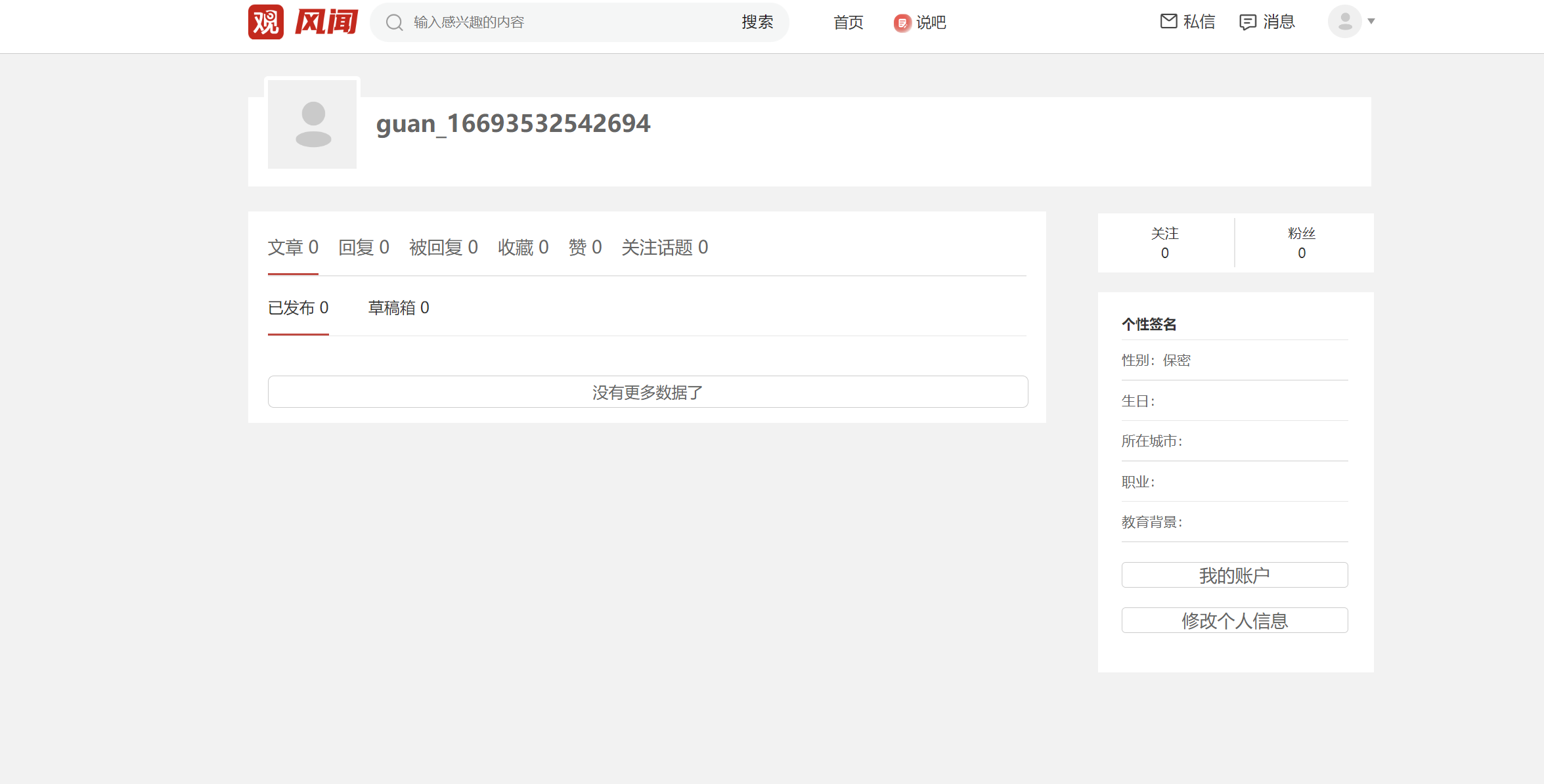
通过学习计算科学导论课程，首先我了解到了许多知识，极大地开阔了眼界，这门课程不仅有助于我正确理解相关学科中所蕴含的科学思维方法，也有助于我树立正确的思想原则，把握日后正确的研究方向。再者通过学习计算科学导论课程，我学会了主动去了解一些自己感兴趣的知识，通过查资料、做报告等方式对该部分知识有较为深刻的了解，如通过自行查阅资料对人工智能领域有了一个初步的认识，再就是通过查阅资料对实时光线追踪渲染的发展历程及实现过程和当今难题有了一个全面的了解，计算科学导论课程通过它独有的魅力引导我主动学习知识，对我现在、将来都非常有益。

# 5 附录

## Github



## 观察者



## 学习强国

## Screenshot_20221125_131720

## 哔哩哔哩

## Screenshot_20221125_131809

## CSDN



## 博客园



## 小木虫



# 参考文献

1. 王晨, 宋亮, 李少昆. 工业互联网平台:发展趋势与挑战[J]. 中国工程科学, v.20(02):23-27.
2. 赵致琢，《计算科学导论（第三版）》，科学出版社，2006。
3. 托马斯·塞缪尔·库恩，《科学革命的结构（第三版）》，北京大学出版社，1996。
4. 特伦斯谢诺夫斯基，《深度学习-智能时代的核心驱动力量》，中信出版社，2019。

[3] 李雁冰, 赵荣彩, 韩林, et al. 一种面向异构众核处理器的并行编译框架[J]. 软件学报, 30(04):121-141.

[4]Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra等.Wide & Deep Learning for Recommender Systems.

[5]Diederik P. Kingma, Jimmy Lei Ba. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION.