IRON: Inverse Rendering by Optimizing Neural SDFs and Materials from Photometric Images

- https://arxiv.org/abs/2204.02232
- Kai-46/IRON: Inverse rendering by optimizing neural SDF and materials from photometric images (github.com)
- [v1] Tue, 5 Apr 2022 14:14:18 UTC (26,335 KB)

> Previous work:
>
> - DVR: autonomousvision/differentiable_volumetric_rendering (github.com)
> - IDR: Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance (lioryariv.github.io)
> - NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction (lingjie0206.github.io)

1 Introduction

> *inverse rendering*: the reconstruction of shape and appearance of real-world objects from a set of 2D input images

- recently developed neural representations for shape and radiance fields entangle material and lighting, and cannot be directly used for applications like relighting or material editing
  - 这些形式（NeRF, DeepSDF, NeuS等neural implicit network）一般是晦涩的隐式表达，而且把材料的计算跟光照的计算混杂在一起，不适合用来做重光照工作或者材质编辑工作
- IRON addresses the inverse rendering problem with the objective of **embracing both the flexibility and compactness of neural representations**, and **the convenience of meshes with material textures** in downstream applications
- IRON includes 2 neural scene components: neural SDF and neural materials
- IRON performs an inverse rendering optimization starting from multi-view images captured by co-locating a flashlight with a moving camera (called *photometric images* in recent work)
  - 光度学图片，在暗室中利用移动的闪光灯+相机拍摄
- key contributions
  - RON, a neural inverse rendering pipeline for high-quality reconstruction of object shape and spatially varying materials, outperforming existing methods for photometric images
  - A hybrid optimization scheme for neural SDFs and materials that **first optimizes a volumetric radiance field** then **performs edge-aware physics-based surface rendering** for improved performance and compatibility with meshes and material textures

- An edge-aware rendering optimization featuring a novel edge sampling algorithm that generates unbiased gradient estimates for better optimizing neural SDFs.

> 原文指出NeRF's volume rendering nature leads to reduced quality of estimated surface geometry。内涵是NeRF着重于新视角的渲染，而不是通过inverse rendering还原出geometry / topology，做到这一点的是IDR，Neus等工作及其变体

3 Method

Assumptions

- opaque objects
- the input photometric images are captured using collocated flashlight illumination without ambient light
  - 即暗室中，用放在一起的闪光灯和相机拍摄
- for efficiency, ignore shadows (which are minimal in practice due to the collocated flashlight) and global illumination effects like interreflections

I&O

IRON consists of 4 compact MLPs (one neural SDF and 3 neural materials) with positional encoding:

- **neural SDF** $S_{\Theta_s}: \mathbf{x} \to (S, \mathbf{f})$
  - $\mathbf{x}$ a 3D location
  - S an SDF value
  - $\mathbf{f}$ a 256D local geometric feature descriptor, can then be fed into neural material networks
  - Θ_s weights
  - `fields.py/SDFNetwork` instantiated as `sdf_network` in `render_volume.py` and `render_surface.py`
    - an **8-layer** MLP of width **256** and a **skip connection** at the 4th layer
    - The input 3D location $\mathbf{x}$ is encoded by **positional encoding using 6 frequencies** to compensate the spectral bias of MLPs
    - the gradient is right the normal
- **neural diffuse albedo** $\beta_{\Theta_\beta}: (\mathbf{x}, \mathbf{n}, \mathbf{n}, \mathbf{f}) \to \beta$
  - $\mathbf{x}$ a 3D location
  - $\mathbf{n}$ the 3D surface normal / $-\mathbf{d}$ the 3D view direction
  - $\mathbf{f}$ a 256D local geometric feature descriptor
  - β the 3D diffuse albedo at $\mathbf{x}$ given surface normal $\mathbf{n}$ and feature descriptor
  - Θ_β weights

    - This function plays a dual role in optimization.
        - In first optimization phase, we treat β as a neural radiance field, and include view direction as an additional parameter (in place of the second $\mathbf{n}$).
        - In the second phase, we constrain β to represent diffuse albedo in our edge-aware physics-based surface rendering
    - `fields.py/RenderingNetwork` instantiated as `color_network` in `render_volume.py` and `color_network_dict["diffuse_albedo_network"]` in `render_surface.py`
        - an **8-layer** MLP of width **256** and a **skip connection** at the 4th layer
        - The input 3D location $\mathbf{x}$ is **positional-encoded using 10 frequencies**. The second surface normal $\mathbf{n}$ (equals the viewing direction in the first stage of volumetric radiance fields rendering) is **positional-encoded using 4 frequencies**.
- **neural specular albedo** $\mathcal{K}_{\Theta_\mathcal{K}}: (\mathbf{x}, \mathbf{n}, \mathbf{f}) \rightarrow \mathcal{K}$
    - encodes 3D spatially-varying specular albedo $\mathcal{K}$
    - $\Theta_\mathcal{K}$ weights
    - `fields.py/RenderingNetwork` instantiated as `color_network_dict["specular_albedo_network"]` in `render_surface.py`
        - an **4-layer** MLP of width **256**
        - The input 3D location $\mathbf{x}$ is **positional-encoded using 6 frequencies**
- **neural roughness** $\alpha_{\Theta_\alpha}: (\mathbf{x}, \mathbf{n}, \mathbf{f}) \rightarrow \alpha$
    - encodes the 1D spatially-varying specular roughness
    - small values indicate shiny surfaces, and large values less shiny
    - Θ_α weights
    - `fields.py/RenderingNetwork` instantiated as `color_network_dict["specular_roughness_network"]` in `render_surface.py`
        - an **4-layer** MLP of width **256**
        - The input 3D location $\mathbf{x}$ is **positional-encoded using 6 frequencies**

> 看代码和文章都可以看出，这种编排对IDR是有很大借鉴的。而IDR的一个思想就是Disentangling Geometry and Appearance，其中neural SDF做的即是geometry，neural materials做的即是appearance

two-stage optimization scheme:

- in the first stage
    - optimize neural SDF S_{Θ_s} and diffuse albedo β_{Θ_β} by treating β_{Θ_β} as a volumetric radiance field
    - this phase is designed to **recover correct object topology** and serves as an **initialization for the second phase**
- in the second stage

- **jointly** optimizing neural SDF S_{Θ_s}, neural materials β_{Θ_β}, $\mathcal{K}_{\Theta_\mathcal{K}}$, and α_{Θ_α}, and light intensity L via an edge-aware physics-based surface rendering method
- refine geometric details and factorize materials from lighting

> 其实这里也说得很清楚了，两个表示材料的神经材料场只有在stage 2也就是进行表面渲染的阶段才训练。stage 1是用来恢复拓扑结构的，与材料无关。

3.1. Volumetric radiance field rendering (stage 1)

> Target: to harness the **power of volume rendering in recovering correct object topology**, i.e., the correct number and location of holes in the geometry
>
> Code: `render_volumne.py`

- optimize $\beta_{\Theta_\beta}(\mathbf{x}, \mathbf{n}, \mathbf{n}, \mathbf{f})$ as a view-dependent neural radiance field by substituting view direction $-\mathbf{d}$ for the second $\mathbf{n}$
- perform volumetric radiance field rendering of the neural SDF S_{Θ_s} and colors $\beta_{\Theta_\beta}(\mathbf{x}, \mathbf{n}, -\mathbf{d}, \mathbf{f})$
- the volumetric nature of this stage is **inconsistent** with our goal of producing shape and materials that are compatible with the mesh-based rendering paradigm used in existing graphics pipelines. This **motivates the second optimization stage** where we perform edge-aware physics-based surface rendering

3.2. Edge-aware physics-based surface rendering (stage 2)

> Target: to perform full inverse rendering stage that jointly optimizes neural SDF S_{Θ_s}, neural materials β_{Θ_β}, $\mathcal{K}_{\Theta_\mathcal{K}}$, and α_{Θ_α}, and light intensity L from photometric images
>
> Code: `render_surface.py`

This stage has two key components: differentiable physics-based shading and edge-aware surface rendering.

Physics-based shading

As the photometric image inputs have co-located flashlight and camera, the light direction is aligned with the view direction across pixel locations. The rendering equation can be simplified as:

$$L_o = \int_\Omega L_i(w_i, \mathbf{x}) f_r(w_o, w_i, \mathbf{x})(w_i \cdot \mathbf{n}) dw_i \tag{1}$$

$$\approx L_i(w_o, \mathbf{x}) f_r(w_o, w_o, \mathbf{x})(w_o \cdot \mathbf{n}) \tag{2}$$

- L_o observed light
- $\mathbf{x}$ 3D surface location

- $\mathbf{n}$ 3D surface normal
- w_i light direction (which is the same as view direction when we use collocated flash)
- w_o view direction
- L_i incident light
- f_r BRDF, *GGX* model is adopted, whose parameters are encoded by the neural materials

The white flashlight is modeled as a point light sources as :

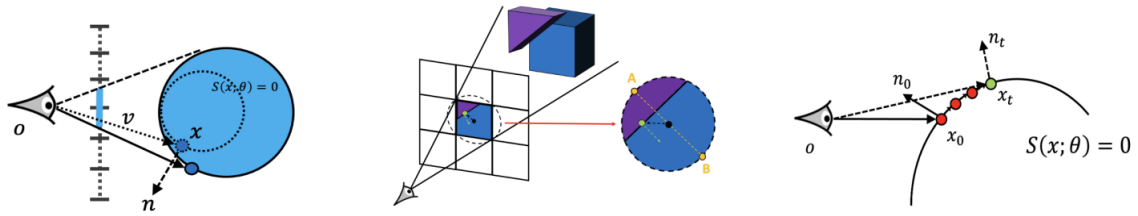
$$L_i(w_o; \mathbf{x}) = \frac{L}{||\mathbf{x} - \mathbf{o}||_2^2} \tag{3}$$

- L a scalar light intensity
- $\mathbf{o}$ the light location, same as the camera location in the collocated capture setup

Edge-aware surface rendering

- a key issue in prior neural surface rendering works such as IDR and DVR: their differentiable rendering module only works for interior pixels, due to their assumption of a smooth surface inside each pixel footprint. This assumption fails for edge pixels, where shading color is a combination of colors at disconnected surface pieces, as shown in Fig. 2(b)
- For this reason, these methods compute biased gradients w.r.t. the weights of the neural SDF that move surface points along camera rays, but not in the image plane, due to missing edge gradients reflecting how color changes w.r.t. edge location, as shown in Fig. 2(a)

> 感觉意思是，以前方法对梯度（法向量）的计算是有偏差的，对于某个图像进行优化时会导致边缘点的SDF并不在图像平面内进行移动/扩散，而是沿着camera ray / viewing direction的方向扩散（边缘点的SDF应该是沿着2D图像平面上的法向量方向扩散，而不是camera ray的方向，见算法的step 2）。
>
> 会出现这种偏差：参考The question about the edge gradient problem existed in IDR · Issue #22，作者意思应该是，IDR是采用了mask的，并且计算mask loss。而IRON是完全抛弃了mask，因此并不edge-aware的IDR是无法正确判定轮廓/edge的，会把edge当作图像内部，从而导致edge处的SDF依然沿着camera ray的方向扩散。而edge-aware surface rendering可以正确判定轮廓/edge，相当于起到了IDR中mask loss的作用



(a) Assuming known color, prior neural surface rendering methods [26, 39] fail to deform a small sphere to a target large sphere due to lack of edge handling.

(b) The shading at an edge pixel involves the combination of the shading at disconnected surface pieces identified using subpixel localization of edges.

(c) We locate edge points in 3D by walking on the zero level set of the neural SDF, then project 3D edge points into 2D for subpixel edge localization.

Figure 2. Illustration of edge-aware surface rendering for neural SDFs. (a) Existing neural surface rendering methods ignore geometric discontinuities, making it difficult to deform neural SDFs to match silhouettes even for simple objects. (b) Geometric discontinuities are introduced by edge pixels where multiple depth values are present in a single pixel, motivating our proposed edge sampling algorithm. (c) Our method can localize subpixel-accurate edges for neural SDFs enabling correct shading calculations at edge pixels.

以上是提出这个算法的意义，以下是算法的阐述：

Algorithm 1 Locate edge points

Input: ray-surface intersection $\hat{\boldsymbol{x}}$
Output: an edge point or NOT_FOUND.
Hyperparams: max # steps K, step size ϵ, threshold δ.

1: $\boldsymbol{x}_t \leftarrow \hat{\boldsymbol{x}}$
2: **for** $i \leftarrow 1$ to K **do**
3: **if** $(\frac{\boldsymbol{x}_t - \boldsymbol{o}}{\|\boldsymbol{x}_t - \boldsymbol{o}\|_2})^T \boldsymbol{n}_t < \delta$ **then**
4: **return** $\boldsymbol{x}_t$
5: **else**
6: $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_t + \epsilon \cdot (\boldsymbol{n}_t - \frac{\boldsymbol{o} - \boldsymbol{x}_t}{(\boldsymbol{o} - \boldsymbol{x}_t)^T \boldsymbol{n}_t})$
7: **end if**
8: **end for**
9: **return** NOT_FOUND

> line 3：判断视线方向和法向量方向是否足够小（接近90度）
>
> line 6：利用视线方向和法向量方向往边缘点（edge point）逼近

- We **address the above issue** via a novel edge sampling algorithm tailored for neural SDFs. Our algorithm has three steps:
    1. localize subpixel-accurate 2D edges by detecting 3D edge points that are then projected to the 2D image,
    2. reparametrize edge points such that they can back-propagate gradients to the neural SDF in an auto-differentation framework,
    3. compute the shading color for edge pixels
- In step 1 - 边缘点定位 - `raytracer.py/locate_edge_points()`
    - we start from ray-surface intersections found by sphere tracing the center rays at each pixel location, and walk on the surface along the direction defined in line 6 of Algorithm 1 and illustrated in Fig. 2(c) until reaching a 3D edge point or a max number of steps
    - For the sake of efficiency, we only do the surface walk process for the ray-surface intersections at depth discontinuity pixels in order to reduce the number of evaluations of the neural SDF; we identify such depth discontinuity pixels as ones with depth Sobel gradient magnitude above a certain threshold τ .
    - We then project detected 3D edge points to image space, producing both subpixel-accurate edge locations and an edge mask marking pixels containing edges. We also obtain 2D edge normal directions by projecting the 3D surface normals at these edge points to 2D.
    - 针对the ray-surface intersections at depth discontinuity pixels，用算法1找出真正的边缘点$\mathbf{x}$在哪。然后再把边缘点投影到图像空间，获得一个精确的edge mask和2D edge normal directions
- In step 2 - 边缘点优化 - `raytracer.py/render_edge_pixels()`
    - we reparameterize the edge points' locations $\mathbf{x}$ to make them differentiable with respect to network weights of the neural SDF

- the differentiable ray-surface intersection equation (Eq. 4) for interior points in only captures how perturbations to neural SDF weights move the ray-surface intersection along the camera ray - `raytracer.py/reparam_points()`

$$\mathbf{x}_{\Theta_S} = \mathbf{x} - \frac{\mathbf{o} - \mathbf{x}}{\mathbf{n}^T(\mathbf{o} - \mathbf{x})} S_{\Theta_S}(\mathbf{x}) \tag{4}$$

- while for edge points, we care about **their movement along the surface normal**. Hence, to reparameterize edge points correctly, we propose to replace the viewing direction $\mathbf{o} - \mathbf{x}$ ($\mathbf{o}$ is the camera origin, $\mathbf{x}$ is a surface point) in Eq. 4 with the surface normal $\mathbf{n}$, and arrive at Eq. 5. - `raytracer.py/reparam_points()`

$$\mathbf{x}_{\Theta_S} = \mathbf{x} - \frac{\mathbf{n}}{\mathbf{n}^T\mathbf{n}} S_{\Theta_S}(\mathbf{x}) = \mathbf{x} - \mathbf{n} S_{\Theta_S}(\mathbf{x}) \tag{5}$$

- 直观的理解：对于一个从3D投影成2D的轮廓内部的点，SDF需要沿着视线方向（viewing direction，也就是camera rays）扩散才能形成正确的轮廓；而对于图像平面上的边缘点，SDF需要沿着法向量方向扩散才能形成正确的轮廓

- In step 3 - 边缘点着色 - `raytracer.py/render_edge_pixels()`
  - we compute the shading at each edge pixel.
  - Consider the edge pixel in Fig. 2(b), and let the green projected edge point have subpixel coordinates $[u, v]$, and the green projected surface normal be $[du, dv]$. Then the black center of this edge pixel has coordinates:

$$[u_c, v_c] = \big[\text{floor}(u), \text{floor}(v)\big] + 0.5. \tag{6}$$

  - We approximate each square pixel footprint using a circle of radius $\sqrt{2}/2$ pixels centered at $[u_c, v_c]$. We then pick 2D locations, labeled A and B, on the circle on either side of the edge. We raytrace and shade the two selected 2D locations.
  - Let us denote the shaded colors as C_A and C_B, respectively. We linearly combine C_A and C_B with weights proportional to the two segment areas separated by the edge. Suppose the fraction of segment area on the same side as A is $w_A \in [0, 1]$:

$$\alpha = 2 \cdot \arccos(\sqrt{2} \cdot [du, dv][u - u_c, v - v_c]^T) \tag{7}$$

$$w_A = 1 - \frac{1}{2\pi} \cdot \big(\alpha - \sin\alpha\big) \tag{8}$$

  - then our predicted edge pixel color is:

$$C = w_A C_A + (1 - w_A) C_B \tag{9}$$

  - 把包含边缘点的方块像素用一个圆来近似，计算edge内外（也就是物体内外）的颜色在这个圆中的权重，并根据这个权重进行混色

3.3. Training and testing

Loss:

$$L = L_2(\text{pyramid}(\hat{I}), \text{pyramid}(\hat{I})) \tag{10}$$

$$+1 - \text{SSIM}(\hat{\text{I}}, \text{I}) \tag{11}$$

$$+\lambda_1 \cdot ||\nabla_x S - 1||_2^2 \tag{12}$$

$$+ \lambda_2 \cdot \max(\text{roughness}(\text{x}) - 0.5,\ 0) \tag{13}$$

- Eq. 10 is the L2 loss on Gaussian pyramids of the predicted image $\hat{I}$ and ground-truth image I
- Eq. 11 is the SSIM loss
- Eq. 12 is the eikonal loss enforcing the validity of the SDF
- Eq. 13 is the roughness range loss encouraging the estimated roughness to stay below 0.5. λ_1, λ_2 are loss weights

Once training is complete, we convert the neural SDF and materials to a triangle mesh and texture map for deployment in the standard graphics pipeline.

1. We **first** extract a mesh from the optimized neural SDF using the marching cube algorithm.
2. We **then** use the Blender Smart UV Project tool to compute a reasonable per-vertex uv mapping.
3. **Finally**, to fill the material texture images, we densely sample points on our triangle meshes with trilinearly interpolated uv coordinates, then query the material networks at the sampled surface points, and finally splat the per-point material parameters to the texture images using their interpolated uv coordinates

4 Evaluation

4.1. Optimizing neural SDFs to fit single image

point out 2 disadvantages of the prior methods:

- the prior neural surface rendering method IDR fails to accomplish fitting single image due to a lack of edge pixel handling
- Although mesh-based differentiable rendering methods can also handle silhouettes, we observe degraded mesh quality over the course of optimization without intermediate remeshing, due to the otherwise fixed mesh topology

4.2. Inverse rendering from photometric images

- IRON outperforms state-of-the-art inverse rendering baselines by a large margin in terms of geometric accuracy and generalization to novel co-located lighting on synthetic data, and produce fewer artifacts like blurry textures on real data
- IRON results in better synthesized specular highlights and more perceptually convincing relighting
- IRON's material estimates are much more detailed than those of baseline methods

5 Conclusion

Limitations and future work

1. First, the use of photometric images leads to a more involved data capture process, although such images simplify inverse methods because of the known single point light and minimal shadows. Future work can explore extensions of our work **to the combination of flashlight and ambient illumination**.
2. Second, we do not model multiple bounces of light. This can lead to material estimation errors in concave regions that feature significant interreflection. Future directions include **devising computationally-efficient global illumination rendering algorithms for neural SDF representations**.
3. Third, our current BRDF model assumes opaque surfaces, and thus we do not expect our method to work well **on transparent and translucent objects** that feature significant refraction and subsurface scattering

Proof of edge point re-parametrization

> 见原文

- 为什么要参数化?
  - IDR里面也有相同的步骤，这是为了通过反向计算，使得SDF能够正确地扩散成轮廓
- 变量
  - $\mathbf{x}_{\Theta_{\S}}$ 网络给出的intersection point / edge
  - t_{Θ_S} 即一个要求得的未知变量，证明的目的是在讨论边缘点的情况下把它表达出来；当网络给出的intersection point就是事实上的intersection point时，$t_{\Theta_S}=0\ \rightarrow \mathbf{x}_{\Theta_S}=\mathbf{x}+0\cdot\mathbf{n}=\mathbf{x}$（这也就是eq15和eq16的意思）
  - $\Theta^{(0)_S}$ 意思不知道咋表述.....ground truth吧大概，或者收敛?
- eq14即参照IDR的写法，写出了包含t_{Θ_S}的表达式
- eq15-eq16即在ground truth（收敛?）时，eq14需要满足的性质
- eq17即SDF要能正确地扩散成轮廓，因此边缘点为输入时，SDF应该返回0
- eq18-eq22即各种求导和代入
- eq23即对eq22积分，再通过eq16排除常数项可得

Comparison with PhySG

- PhySG: Inverse Rendering with Spherical Gaussians for Physics-based Material Editing and Relighting (kai-46.github.io)是同一个作者在CVPR 2021的工作
- PhySG requires input object segmentation masks（就是黑底白shape，一个图对应一个mask），而IRON不需要

- Image Segmentation | Types Of Image Segmentation (analyticsvidhya.com)

  > Object detection builds a bounding box corresponding to each class in the image. But it tells us nothing about the shape of the object. We only get the set of bounding box coordinates. We want to get more information – this is too vague for our purposes.
  >
  > Image segmentation creates a pixel-wise mask for each object in the image. This technique gives us a far more granular understanding of the object(s) in the image.

- What is a mask in image segmentation? - Quora

  > A mask is a binary image that is used to separate an image into two or more regions. In image segmentation, a mask is typically used to identify the region of an image that contains the object of interest.

- Generating Image Segmentation Masks — The Easy Way | by Abhiroop Talasila | Towards Data Science

- Our method can also handle **spatially-varying specular roughness**, whereas PhySG assumes a constant and uniform specular lobe shape
- PhySG assumes static environmental lighting, while IRON requires a collocated flash
- **Conclusion**: IRON recovers much more accurate geometry details than PhySG

Code

- stage 1对应的文件是`render_volume.py`
  - 体渲染使用NeuSRenderer
  - 定义了一个nerf对象用于渲染背景，最后所有的网络都被封装到了一个NeuSRenderer（实例化为`renderer`）
    - 为什么要用nerf渲染背景?
      - 文章可以说是在idr与NeuS的基础上开展的，其中idr是2020年的，数据集是需要有mask的，区别于NeRF及其变体引入了SDF；而NeuS对于mask的要求是可有可无；再到了今天的IRON，已经完全摒弃了对mask的要求（代码中还有一部分相关代码）
      - 也就是说，对于四个核心MLP都借鉴了IDR的IRON来说，其神经网络本身是没有处理背景的能力的（因为背景都被IDR中的mask屏蔽掉了）
      - 这三个工作的监督方式与NeRF是类似的，从隐式场中渲染出图像与数据集图像进行比较得到loss
      - IRON的文件组织与NeuS是相似的，其中`models/fields.py`应该完全一样
    - `render()`用来渲染
      - 调用`render_core()`渲染主体部分（`render_core()`包括与NeRF渲染的背景部分加权计算）
  - **neural SDF**: `sdf_network`
  - **neural diffuse albedo**: `color_network`

- stage 2对应的文件是`render_surface.py`
  - 表面渲染使用`models/raytracer.py`，其中完整的表面渲染过程在`render_camera()`
  - `color_network_dict`里面的`"color_network"`好像是没用的

Other Resources

1. SDF(signed distance field)基础理论和计算 - 知乎 (zhihu.com)
2. 机器视觉中的前景和背景是什么意思_Blazer！的博客-CSDN博客
3. 光的能量与颜色——辐射度量学，光度学，色度学 - 知乎 (zhihu.com)
4. 7.2. Surface Versus Volume Rendering — MPHY0026 documentation
5. Marching Cubes算法理解*@左左@右右的博客-CSDN博客*marchingcubes
6. 常见的光照模型*Aimy7707的博客-CSDN博客*光照模型
7. 微表面模型 - PBR渲染管线的材质 - UWA问答 | 博客 | 游戏及VR应用性能优化记录分享 | 佑虎科技 (uwa4d.com)
8. 基于物理的渲染—更精确的微表面分布函数GGX - 知乎 (zhihu.com)
9. 图像金字塔之高斯金字塔 - 简书 (jianshu.com)
10. 图像质量评价指标之 PSNR 和 SSIM - 知乎 (zhihu.com)

    > SSIM简化漏了一个条件：一般取$c_3 = c_2/2$

11. mitsuba基本使用教程 - 简书 (jianshu.com)
12. lioryariv/idr (github.com)
13. Totoro97/NeuS: Code release for NeuS (github.com)
14. [论文笔记]NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction - 知乎 (zhihu.com)
15. Neus学习笔记_ACxz的博客-CSDN博客