光线追踪
简单介绍

Why BDPT

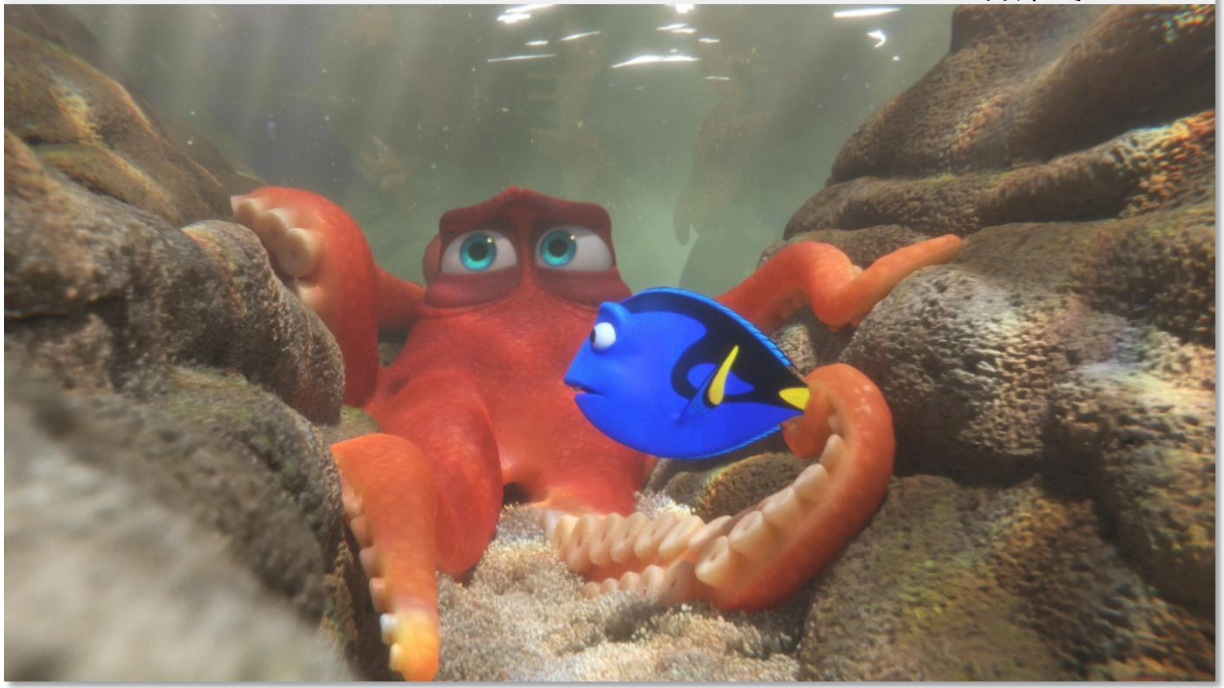
如何用taichi
实现一个BDPT

taichi语言的
开发心得

如何用taichi实现一个双向路径追踪积分器(BDPT)

taichi-con2

林耀冬



什么是路径追踪?

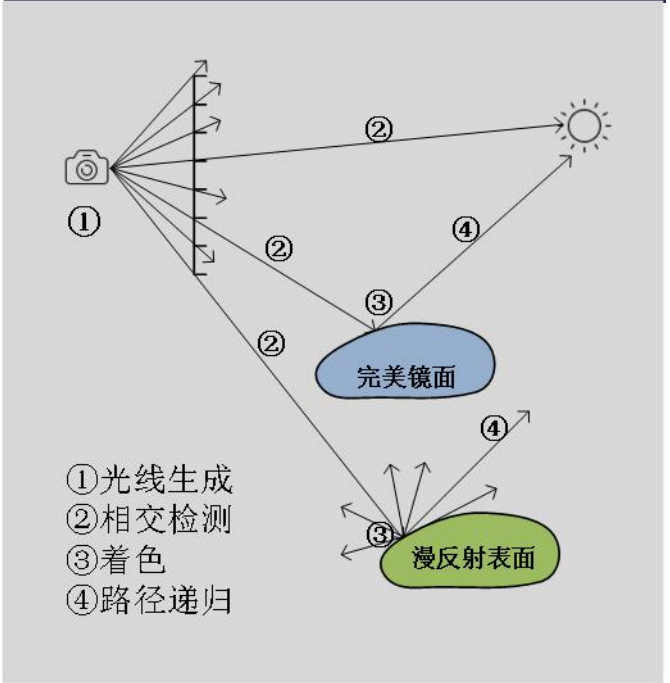
路径追踪
简单介绍

Why BDPT

如何用taichi
实现一个BDPT

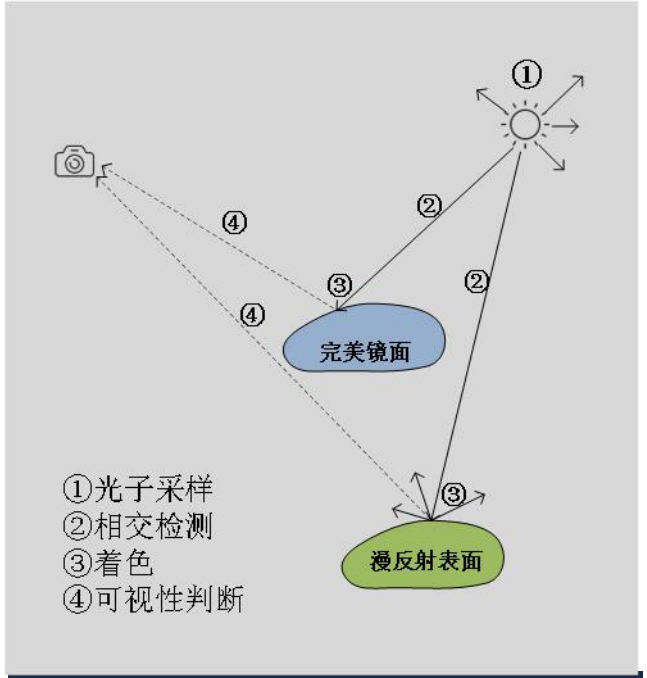
taichi语言的
开发心得

从相机发射光线 PathTrace



只计算可视范围相交，性能好

从光源出光线 LightTracing



真实光线传播规律，更丰富光学现象

参考资料：https://graphics.stanford.edu/courses/cs348b-00/lectures/lecture12/walk011.html

路径追踪积分器的实现

光线追踪
简单介绍

Why BDPT

如何用taichi
实现一个BDPT

taichi语言的
开发心得

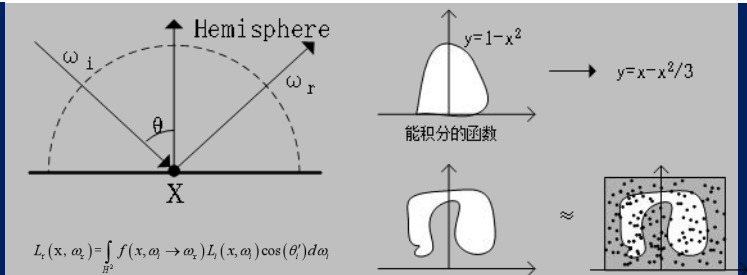
伪代码

算法 1 路径追踪算法

输入： 图像分辨率w,h,场景信息Scene
输出： 每个像素点的Color

```
function PathTrace(Scene, w, h)
    for i = 0 → w − 1 and j = 0 → h − 1 do
        ray = GetCameraRayFromPixel(i, j)                          //计算相机光线
        ThroughOut = [1.0, 1.0, 1.0, 1.0]                          //初始化光线通量
        for depth = 0 → depthmax do
            Intersection = Intersect(Scene, ray)                   //场景相交
            Color+ = DirectLighting(Intersection) * ThroughOut     //直接光照[1]
            ray = SampleRay(Intersection)                          //采样间接光照方向[1]
            Throughput* = BRDF(CameraRay, ray, Intersection)       //计算间接光照通量[1]
        end for
    end for
    return Color
end function
```

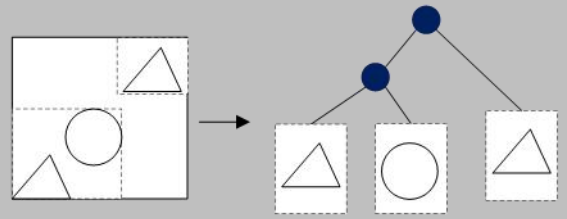
采样光线 = 半球积分



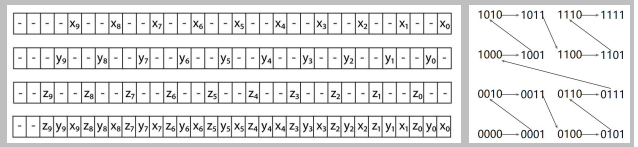
蒙特卡洛积分

相交检测

BVH包围层次盒



- 计算所有三角形的重心坐标p=[x,y,z]，并归一化
- 构建3D莫顿码[2]，莫顿码是Z-index排布，有空间意义



- 基数排序莫顿码
- 莫顿码越接近，说明空间上越相近，被分到同一个节点下

参考资料：[1]https://www.realtimerendering.com/raytracinggems/rtg2/index.html 英伟达光追精粹2
[2]Nvidia 2012: Maximizing Parallelism in the Construction of BVHs,Octrees, and k-d Trees

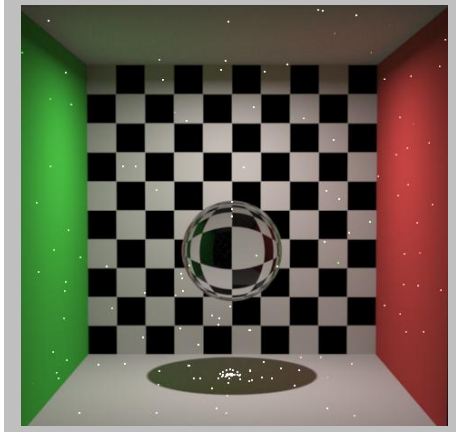
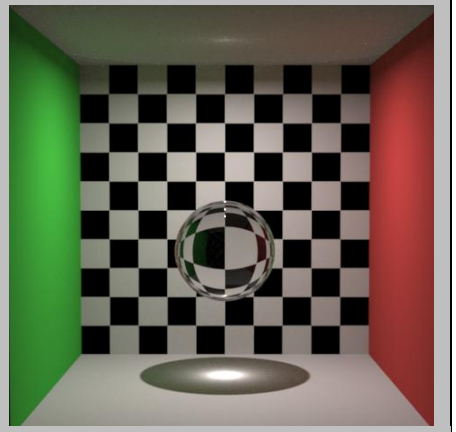
双向方法的优势

光线追踪
简单介绍

Why BDPT

如何用taichi
实现一个BDPT

taichi语言的
开发心得

单向路径追踪	双向路径追踪
	

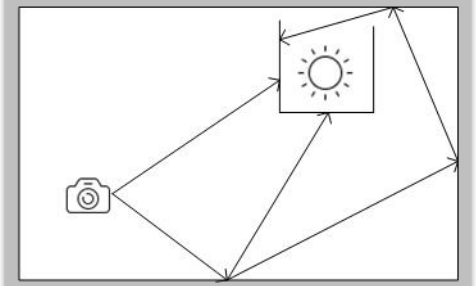
优势

漫反射区：噪点少

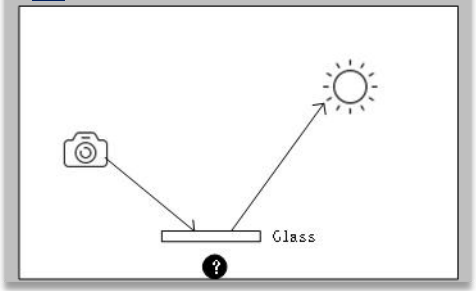
散焦区：效果好

采样效率

光源出射口狭窄



透明物体多，散焦区域大



参考资料：[1]Towards Bidirectional Path Tracing at Pixar，迪士尼关于BDPT的讲解

Overview

光线追踪
简单介绍

Why BDPT

如何用taichi
实现一个BDPT

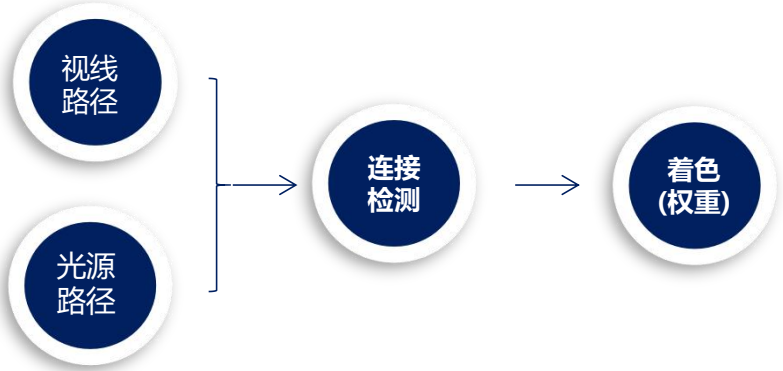
taichi语言的
开发心得

伪代码

算法 1 双向路径追踪算法

输入： 图像分辨率w,h,场景信息Scene
输出： 每个像素点的Color

```
function BDPT(Scene, w, h)
    for i = 0 → w − 1 and j = 0 → h − 1 do
        EyeRay = GetEyeRayFromPixel(i, j)
        LightRay = GetLightRayFromPixel(i, j)
        GenerateEyePath(Scene, EyeRay)
        GenerateLightPath(Scene, LightRay)
        for e = 0 → depthmax, l = 0 → depthmax do
            eyeSample, radiance = ConnectPath(Scene, e, l)
            Color[eyeSample] += radiance
        end for
    end for
    return Color
end function
```



相同时间渲染

bdpt　　pt

确定kernel

- 寻找伪代码中的For循环
- 数据无关算法
- 时间复杂度→ 工作复杂度 步骤复杂度[1]

参考资料：[1]https://www.youtube.com/watch?v=ApFfM2Hfhx0

连接光线

光线追踪
简单介绍

Why BDPT

如何用taichi
实现一个BDPT

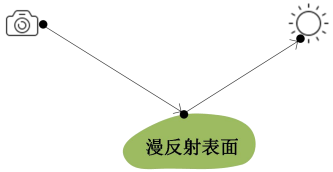
taichi语言的
开发心得

伪代码

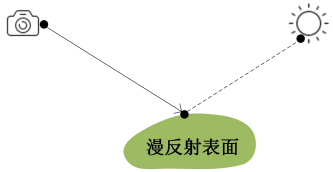
算法 1 双向路径追踪算法

```
function ConnectPath(Scene, e, l)
    radiance = [0.0, 0.0, 0.0]
    if l==0 then
        if eyeStack[e-1].hitLight then
            radiance+ = eyeStack[e − 1].radiance
        end if
    else if e==1 then
        eyeSample = GetRandomEyeSample()
        ray = lightStack[l − 1].hitPos − eyeSample.hitPos
        Intersection = Intersect(Scene, ray)
        if Intersection.hitPos = lightStack[l − 1].hitPos then
            radiance+ = DirectLighting(Intersection) ∗ lightStack[l − 1].beta
        end if
    else if l==1 then
        lightSample = GetRandomLightSample()
        ray = eyeStack[e − 1].hitPos − lightSample.hitPos
        Intersection = Intersect(Scene, ray)
        if Intersection.hitPos = eyeStack[e − 1].hitPos then
            radiance+ = DirectLighting(Intersection) ∗ eyeStack[e − 1].beta
        end if
    else
        ray = eyeStack[e − 1].hitPos − lightStack[l − 1].hitPos
        Intersection = Intersect(Scene, ray)
        if Intersection.hitPos = eyeStack[e − 1].hitPos then
            radiance+ = DirectLighting(lightStack[l − 1]) ∗ lightStack[l − 1].beta ∗
DirectLighting(eyeStack[e − 1]) ∗ eyeStack[e − 1].beta
        end if
    end if
    return eyeSample, radiance
end function
```

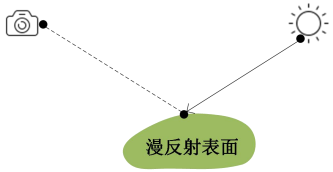
l —— 光源路径上采样点个数
e —— 相机路径上采样点个数



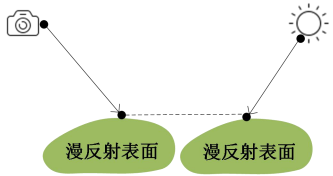
l=0 e=3
PathTrace



l=1 e=2
DirectLight



e=1 l=2
LightTracing



e>=2 l>=2
Else

参考资料：https://pbr-book.org/3ed-2018/Light_Transport_III_Bidirectional_Methods/Bidirectional_Path_Tracing，PBRT-V3中的讲解

开发思路

光线追踪
简单介绍

Why BDPT

如何用taichi
实现一个BDPT

taichi语言的
开发心得

数据结构设计

场景信息：bvh树

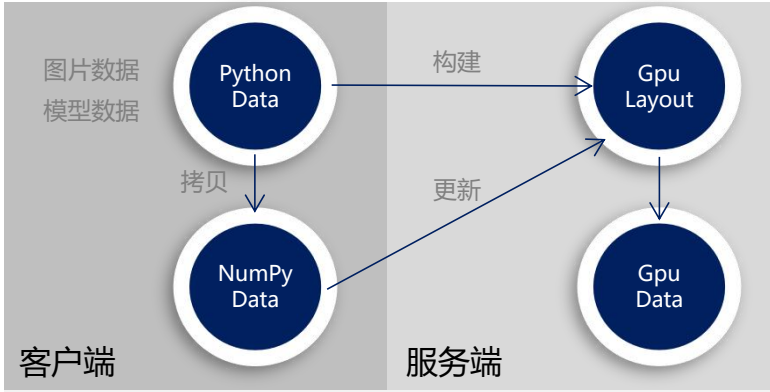
储存光线路径信息：位置，法线，通量

颜色输出信息：RGBA

材质信息：色彩，反射率

文件名	意义
Accel	加速结构
Brdf	双向反射函数(材质)
Example	构建场景代码
Integrator	积分器
Sky	天空盒
Spectrum	光谱（采样）
Texture	纹理（采样）
Camera.py	小孔相机
Scene.py	场景数据抽象层
SceneData.py	场景数据结构体定义
UtilsFunc.py	常用的函数

数据初始化



- 变量**声明**在__init__中：

  self.material = ti.Vector.field(SCD.MAT_VEC_SIZE, dtype=ti.f32)

- 变量的**布局**在set_python_data中：

  ti.root.dense(ti.i, self.material_count).place(self.material)

- 数据**上传**在set_gpu_data中：

  self.material.from_numpy(self.material_np)

多重重要性采样（MIS）

光线追踪
简单介绍

Why BDPT

如何用taichi
实现一个BDPT

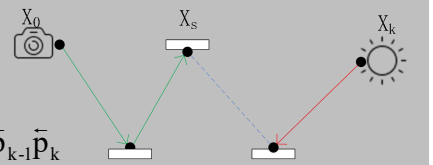
taichi语言的
开发心得

无MIS	MIS

将重要性采样化为权重

$$\omega_s(X)=\frac{p_s(X)}{\sum_{i=0}^{k}p_i(X)}$$

$$p_s(X)=\vec{p}_0\vec{p}_1\cdots\vec{p}_s\vec{p}_{s+1}\cdots\vec{p}_{k-1}\vec{p}_k$$



展开公式 $$\frac{1}{\omega_s(X)}=\frac{\sum_{i=0}^{k}p_i(X)}{p_s(X)}=\vec{p}_{s+1}\cdot d_s^E+1+\vec{p}_s\cdot d_{s+1}^L$$

相机部分 $$d_s^E=\frac{1+\vec{p}_s d_{s-1}^E}{\vec{p}_{s-1}}\longrightarrow d_1^E=\frac{1}{\vec{p}_0}$$

光源部分 $$d_{s+1}^L=\frac{1+\vec{p}_{s+1}d_{s+2}^L}{\vec{p}_{s+2}}\longrightarrow d_k^L=\frac{1}{\vec{p}_{k+1}}$$

注：p的箭头方向指的是光线到X_s点的方向，不是前向和后向pdf

参考资料：[1]https://www.slideshare.net/takahiroharada/introduction-to-bidirectional-path-tracing-bdpt-implementation-using-opencl-cedec-2015，AMD工程师关于BDPT的演讲
[2] https://graphics.stanford.edu/courses/cs348b-03/papers/veach-chapter9.pdf

taichi开发小贴士

光线追踪
简单介绍

Why BDPT

如何用taichi
实现一个BDPT

taichi语言的
开发心得

Tips

使用继承

节省大量重复代码

父类

```
@ti.data_oriented
class example:
    def __init__(self, imgSizeX, imgSizeY, sample_count):
```

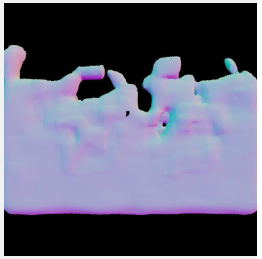
子类

```
import Example
@ti.data_oriented
class example(Example.example):
    def __init__(self, imgSizeX, imgSizeY, sample_count):
        ti.init(arch=ti.gpu)
        Example.example.__init__(self, imgSizeX, imgSizeY, sample_count)
```

Debug

taichi内部的print, 打印过多

ColorMap



Static

节省代码，却增加了编译时间　替换长变量名

```
for i in ti.static(range(3)):
    tmp = coeffs
    tmp[i] -= RGB2SPEC_EPSILON
    r0 = eval_residual(tmp, rgb)

    tmp = coeffs
    tmp[i] += RGB2SPEC_EPSILON
    r1 = eval_residual(tmp, rgb)
```

```
@ti.func
def get_viscosity_Ax(x: ti.template(), i):
    neigh_count, serachR      = ti.static(\
        particle_data.hash_grid.neighborCount, \
        particle_data.hash_grid.searchR)
    ret = ti.Vector([0.0,0.0,0.0])
    cur_neighbor      = neigh_count[i]
```

好用的第三方组件

pip install taichi-glsl

pip install pywavefront

Further

未来计划

纹理系统

BVH算法优化

光线追踪
简单介绍

Why BDPT

如何用taichi
实现一个BDPT

taichi语言的
开发心得

如何用taichi实现一个双向光线追踪积分器

Thanks!

https://github.com/lyd405121/ti-raytrace