# Progressive Photon Mapping
## —计算机图形学大作业报告

吴克文 梁家硕

2017 年 5 月

# 目录

# 1 环境

Arch Linux x86_64 Linux 4.10.13-1-ARCH
gcc (GCC) 6.3.1 20170306
cmake version 3.8.0
GNU Make 4.2.1
OpenGL version: 3.0 Mesa 17.0.5

# 2 运行

```
$ make
```

注：

# 3 核心算法

本作业参考了三篇论文 [1][3][4]，以及书 [2]，综合效果和实现难度进行了调整，删去了繁琐的细节调整和性能优化部分。

**Algorithm 1:** Progressive Photon Mapping

**Input:** source.obj material.mtl

**Output:** output.png

```
Read model and store material info.;
Build model KD-Tree based on model info in .obj;
Perform Ray Tracing Pass to restore hitpoints;
for iterations do
    Perform Photon Tracing Pass;
    Build photon KD-Tree;
    for hitpoints do
        Find photons near the hitpoint;
        Accumulate their impact on the hitpoint;
        Update hitpoint info.;
    end
end
Generate output.png using hitpoints' info.;
```

以下为详细算法剖分：

### 3.1 BRDF

### 3.2 Ray Tracing Pass

### 3.3 Photon Tracing Pass

## 4 数理计算

除了整体性的大算法，该大作业也有一些细节上的数理计算方法也值得一提。这些计算虽然底层，但由于调用次数巨大，需要进行常数优化。

### 4.1 射线与球的求交算法

用起点 $\mathbf{s}$ 和向量 $\mathbf{v}$ 表示射线 $l$，用球心 $\mathbf{o}$ 和半径 $R$ 表示球 $S$，设点 $\mathbf{d} = \mathbf{s} + x\mathbf{v}$ 为线圆交点，则有：

$$\|\mathbf{s} + x\mathbf{v} - \mathbf{o}\| = R \tag{1}$$

令 $\mathbf{t} = \mathbf{s} - \mathbf{o}$，并展开方程 (1)，有，

$$\|\mathbf{t}\|^2 - R^2 + 2(\mathbf{t}, \mathbf{v})x + \|\mathbf{v}\|^2 x^2 = 0 \tag{2}$$

再取最小正值 $x$ 带入，即得交点。

CODE: Ray & Sphere

```
bool Ray::intersect(const Sphere &s, Point *p) const{
    Point t = bgn - s.center;
    double b = 2 * dotsProduct(t, vec),
           c = t.len2() - sqr(s.radius),
           delta = sqr(b) - 4 * c;
    if (delta < EPS) return 0;   // no intersection
    delta = sqrt(delta);
    double t1 = (-b - delta) / 2,   // calculate two
           t2 = (-b + delta) / 2,   // possible solution
           res;  // final answer
    if (t1 < EPS) if (t2 < EPS) return 0;   // tangent
                  else res = t2;
       else if (t2 < EPS) res = t1;
            else res = std::min(t1, t2);
    *p = bgn + res * vec;
```

```
    return 1;
}
```

## 4.2 射线与空间凸多边形的求交算法

多边形所在平面可用任意三点表示，又考虑到精度问题，便在初始化多边形时选取多边形上构成三角形面积最大的三点 $\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}$。求出射线与平面的交点，再带回验证是否在凸多边形内即可。

用起点 $\mathbf{s}$ 和向量 $\mathbf{v}$ 表示射线 $l$，设点 $\mathbf{d} = \mathbf{s} + x\mathbf{v}$ 为射线和多边形的交点。

因为交点在平面上，故，

$$\det(\mathbf{s} + x\mathbf{v} - \mathbf{c_1}, \mathbf{c_2} - \mathbf{c_1}, \mathbf{c_3} - \mathbf{c_1}) = 0 \tag{3}$$

令 $\mathbf{t} = \mathbf{s} - \mathbf{c_1}$，$\mathbf{c}' = \mathbf{c_2} - \mathbf{c_1}$，$\mathbf{c}'' = \mathbf{c_3} - \mathbf{c_1}$，有

$$\begin{aligned}
\det(\mathbf{t} + x\mathbf{v}, \mathbf{c}', \mathbf{c}'') =& \det(\mathbf{t}, \mathbf{c}', \mathbf{c}'') + x\det(\mathbf{v}, \mathbf{c}', \mathbf{c}'') && (4)\\
=& \mathbf{t}_x \times yz - \mathbf{t}_y \times xz + \mathbf{t}_z \times xy && (5)\\
& + x(\mathbf{v}_x \times yz - \mathbf{v}_y \times xz + \mathbf{v}_z \times xy) && (6)\\
=& 0 && (7)
\end{aligned}$$

其中，

$$\begin{aligned}
xy &= \mathbf{c}'_x\mathbf{c}''_y - \mathbf{c}'_y\mathbf{c}''_x && (8)\\
xz &= \mathbf{c}'_x\mathbf{c}''_z - \mathbf{c}'_z\mathbf{c}''_x && (9)\\
yz &= \mathbf{c}'_y\mathbf{c}''_z - \mathbf{c}'_z\mathbf{c}''_y && (10)
\end{aligned}$$

考虑如果 $\mathbf{d}$ 在凸多边形内，则有 $\mathbf{d}$ 在平面上每条边的左侧，记平面法向量为 $\mathbf{n}$，多边形逆时针点序为 $\{\mathbf{p_i}\}$，则有，

$$\begin{aligned}
(\mathbf{n}, (\mathbf{p_{i+1}} - \mathbf{p_i}) \times (\mathbf{d} - \mathbf{p_i})) &= \det(\mathbf{n}, \mathbf{p_{i+1}} - \mathbf{p_i}, \mathbf{d} - \mathbf{p_i}) && (11)\\
&= \det(\mathbf{p_i}, \mathbf{p_{i+1}}, \mathbf{n}) + \det(\mathbf{p_i} - \mathbf{p_{i+1}}, \mathbf{n}, \mathbf{d}) && (12)\\
&\geqslant 0 && (13)
\end{aligned}$$

CODE: Ray & Polygon

```
bool Ray::intersect(const Polygon &s, Point *p) const{
    Point ts = bgn - s.pList[s.c1];
```

```
    double k = vec.x * s.yz - vec.y * s.xz + vec.z * s.xy,
           b = ts.x * s.yz - ts.y * s.xz + ts.z * s.xy;
    if (fabs(k) < EPS) return 0; // parallel
    double t = -b / k;
    Point ret = bgn + t * vec;  // intersection
    // pre-calculation
    double txy = s.normvf.x * ret.y - s.normvf.y * ret.x,
           txz = s.normvf.x * ret.z - s.normvf.z * ret.x,
           tyz = s.normvf.y * ret.z - s.normvf.z * ret.y;
    for (int i = 0; i < s.num - 1; ++i)
        if (determinant(s.pList[i], s.pList[i + 1], s.normvf) +
            (s.pList[i].x - s.pList[i + 1].x) * tyz -
            (s.pList[i].y - s.pList[i + 1].y) * txz +
            (s.pList[i].z - s.pList[i + 1].z) * txy < -EPS)
            return 0; // not on the left
    if (determinant(s.pList[s.num - 1], s.pList[1], s.normvf) +
        (s.pList[s.num - 1].x - s.pList[1].x) * tyz -
        (s.pList[s.num - 1].y - s.pList[1].y) * txz +
        (s.pList[s.num - 1].z - s.pList[1].z) * txy < -EPS)
        return 0;

    *p = ret;
    return 1;
}
```

## 4.3 射线与 KD-Tree 节点立方体的判交算法

CODE: Ray & ???

```

```

# 5 效果展示

# 6 总结

本算法还有诸多提高空间，例如：

1. 利用 GPU 并行计算。

2. Stochastic Progressive Photon Mapping

限于时间、精力，无法进一步探索，实属遗憾。

# 参考文献


[1] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Transactions on Graphics (TOG)*, 27(5):130, 2008.

[2] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*, volume 364. Ak Peters Natick, 2001.

[3] Ben Spencer and Mark W Jones. Progressive photon relaxation. *ACM Transactions on Graphics (TOG)*, 32(1):7, 2013.

[4] 李睿, 陈彦云, and 刘学慧. 基于自适应光子发射的渐进式光子映射. 计算机工程与设计, 33(1):219–223, 2012.