1 Light 库说明文档

目录

1.1 数学库

数学库 (LightMath.h)

1.1.1 常数定义

`LightMath.h` 中定义了如下常数:

```
const double EPS = 1e-6;
    // 预设精度
const double PI = 3.14159265358979323846264338327950288419716939937510; // 圆周率
const double INV_PI = 1.0 / PI;
    // 圆周率倒数
```

1.1.2 isZero 函数

```
bool isZero(double d);
```

判断是否为零值，返回值为真当且仅当 `d` 的绝对值小于 `EPS`

1.1.3 Vector3 类型

`Vector3` 是三维向量结构体，支持多种基本运算。

1.1.3.1 分量访问 `Vector3` 定义如下

```
struct Vector3
{
    union
```

```
    {
        double _[3];
        struct {double x, y, z;};
    };
}
```

union 关键字可以让不同的变量共用一段内存

例如，令 v 为一个 Vector3 类型的对象，你可以用 v.x 和 v._[0] 两种等价方法访问其第一个分量；同理，v.y 等价于 v._[1]，v.z 等价于 v._[2]。

1.1.3.2 构造函数

数学库提供了四种构造 Vector3 向量的方法

为了表达方便，我们用 (x, y, z) 的格式表示一个 Vector3 对象。

```
Vector3 MakeVector3();
```

- **返回** (0, 0, 0)

```
Vector3 MakeVector3(double *__);
```

- **返回** (__[0], __[1], __[2])

```
Vector3 MakeVector3(double _x, double _y, double _z);
```

- **返回** (_x, _y, _z)

```
Vector3 MakeVector3(Vector4 v4);
```

- **返回** (v4.x, v4.y, v4.z)，相当于舍去第四维

1.1.3.3 运算符重载

已经提供的运算符重载如下：

```
Vector3 operator + (const Vector3 &b) const;
Vector3 operator - (const Vector3 &b) const;
Vector3 operator - () const;
Vector3 operator * (const Vector3 &b) const;
Vector3 operator / (const Vector3 &b) const;
```

含义如下：

- (a, b, c) + (x, y, z) **返回** (a+x, b+y, c+z)
- (a, b, c) - (x, y, z) **返回** (a-x, b-y, c-z)
- -(x, y, z) **返回** (-x, -y, -z)
- (a, b, c) * (x, y, z) **返回** (a*x, b*y, c*z)
- (a, b, c) / (x, y, z) **返回** (a/x, b/y, c/z)

1.1.3.4 成员函数

```
double sum() const;
```

- **返回**分量和 x + y + z

```
double len_sq() const;
```

- **返回**模长平方

```
double len() const;
```

- **返回**模长

```
Vector3 pow(double index) const;
```

- **返回** (pow(x, index), pow(y, index), pow(z, index))，不会用到

```
bool iszero() const;
```

- **返回**真当且仅当每个分量的绝对值都小于 EPS

1.1.4 Vector4 类型

Vector4 是四维向量结构体，支持多种基本运算。定义与 Vector3 高度相似。

1.1.4.1 分量访问 Vector4 定义如下

```
struct Vector4
{
    union
    {
        double _[4];
        struct
        {
            union
            {
                struct { double x, y, z; };
                Vector3 v;
            };
            double w;
        };
    };
}
```

你可以用 x，y，z，w 和 _[0]，_[1]，_[2]，_[3] 访问四个分量，也可以用 v 访问前三个分量组成的三维向量

1.1.4.2 构造函数

数学库提供了六种构造 Vector4 向量的方法

为了表达方便，我们用 (x, y, z, w) 的格式表示一个 Vector4 对象。

```
Vector4 MakeVector4();
```

- **返回** (0, 0, 0, 0)

```
Vector4 MakeVector4(double *__);
```

- **返回** (__[0], __[1], __[2], __[3])

```
Vector4 MakeVector4(double _x, double _y, double _z, double _w);
```

- **返回** (_x, _y, _z, _w)

```
Vector4 MakeVector4(Vector3 _v, double _w);
```

- **返回** (_v.x_, _v.y, _v.z, _w)，相当于手动加入第四维

```
Vector4 MakeVector4(Vector3 _v, double _w);
```

- **返回** (_v.x_, _v.y, _v.z, _w)，相当于手动加入第四维

```
Vector4 MakePos(double _x, double _y, double _z);
```

- **返回** (_x, _y, _z, 1)，构造齐次坐标表示法坐标

```
Vector4 MakeVec(double _x, double _y, double _z);
```

- **返回** (_x, _y, _z, 0)，构造齐次坐标表示法向量

1.1.4.3 运算符重载

已经提供的运算符重载如下：

```
Vector4 operator + (const Vector4 &b) const;
Vector4 operator - (const Vector4 &b) const;
Vector4 operator - () const;
Vector4 operator * (const Vector4 &b) const;
Vector4 operator / (const Vector4 &b) const;
```

含义如下：

- (a, b, c, d) + (x, y, z, w) **返回** (a+x, b+y, c+z, d+w)
- (a, b, c, d) - (x, y, z, w) **返回** (a-x, b-y, c-z, d-w)
- -(x, y, z, w) **返回** (-x, -y, -z, -w)
- (a, b, c, d) * (x, y, z, w) **返回** (a*x, b*y, c*z, d*w)
- (a, b, c, d) / (x, y, z, w) **返回** (a/x, b/y, c/z, d/w)

1.1.4.4 成员函数

```
double sum() const;
```

- **返回** x + y + z + w

```
double len_sq() const;
```

- **返回**模长平方

```
double len() const;
```

- **返回**模长

```
Vector3 pow(double index) const;
```

- **返回** (pow(x, index), pow(y, index), pow(z, index))，不会用到

```
bool iszero() const;
```

- **返回**真当且仅当每个分量的绝对值都小于 EPS

```
bool iscoord() const;
```

- 判断是否为齐次坐标表示法坐标，**返回**真当且仅当 w 分量的绝对值都大于 0.5

```
Vector3 vec3() const;
```

- **返回**前三个分量组成的三维向量 (x, y, z)

1.1.5 数乘

Vector3 和 Vector4 都可以乘以或除以一个标量

```
Vector3 v3 = MakeVector3(1, 2, 3);
Vector4 v4 = MakeVector4(1, 2, 3, 4);
double c = 2.0;
std::cout << v3*c << std::endl;
std::cout << c*v3 << std::endl;
std::cout << v3/c << std::endl;
std::cout << v4*c << std::endl;
std::cout << c*v4 << std::endl;
std::cout << v4/c << std::endl;
```

1.1.6 dot 函数

```
double dot(const Vector4 &v1, const Vector4 &v2);
```

- **返回** v1, v2 的点积

1.1.7 cross 函数

```
const Vector4 cross(const Vector4 &v1, const Vector4 &v2);
```

- **返回** v1, v2 的叉积，自动忽略第四维度，并填补 0（在齐次坐标表示法下，坐标叉积无意义，故默认传入的变量都为向量）

1.1.8 Matrix33 与 Matrix34 类型

Matrix33 是一个普通的 3×3 矩阵类型，本题不会用到，不再展开。Matrix34 是表示一个仿射变换矩阵，实际大小为 4×4，省略了最后一行（原因说明请参见 math&geometry）。

两个矩阵都内置了和对应长度向量的乘法运算，和同类型矩阵的乘法运算，以及求逆函数 inv()。这两种类型几乎不会用到，只会用到 Matrix34 的乘法运算，故不再展开。

1.2 几何库

几何库 (LightGeometry.h)

几何库的所有内容可以参考**头文件内的注释**，你一般不需要修改其中的内容，只需找到其中有用的函数进行使用即可。

- **Plane** 为平面类，提供了与直线求交的函数
- **Ray** 为射线类，具体说明参见题面
- **Triangle** 为三角形类，需要选手完善求交函数
- **Sphere** 为球形类，用于内置的模型表示，选手无需关注
- **BoundingBox** 为包围盒类，用于求交加速，你可能会在 accelerate 这个点用到

1.3 随机库

随机库（LightRandom.h）

随机库是用于在多线程条件下避免随机序列耦合的，这个库定义了 Light 命名空间下的一个子命名空间 Random，你可以用 Light::Random::xxx 的方式访问其中的函数或者其他内容。

在本次比赛中，需要你用到随机变量的地方都已经提供了指向线程间独立的随机引擎对象的指针 Random::RAND_ENGINE *eng，你无需对已经提供的指针 eng 进行修改，只需要通过以下两种方式调用库函数获取随机变量：

```
Light::Random::randDouble(eng, _min, _max);
//获取 [_min, _max) 之间的均匀随机双精度浮点数

Light::Random::randInt(eng, _min, _max);
//获取 [_min, _max) 之间的均匀随机4字节有符号整数
```

注意：eng 指针是调用者提供的，无需自行生成对象，拿来即用。如果自行定义随机引擎可能会导致线程间随机数耦合，因此种情况造成结果出现偏差或不理想，后果自行承担。