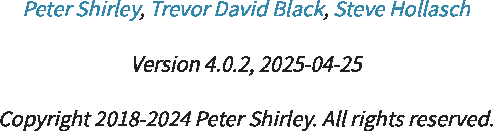
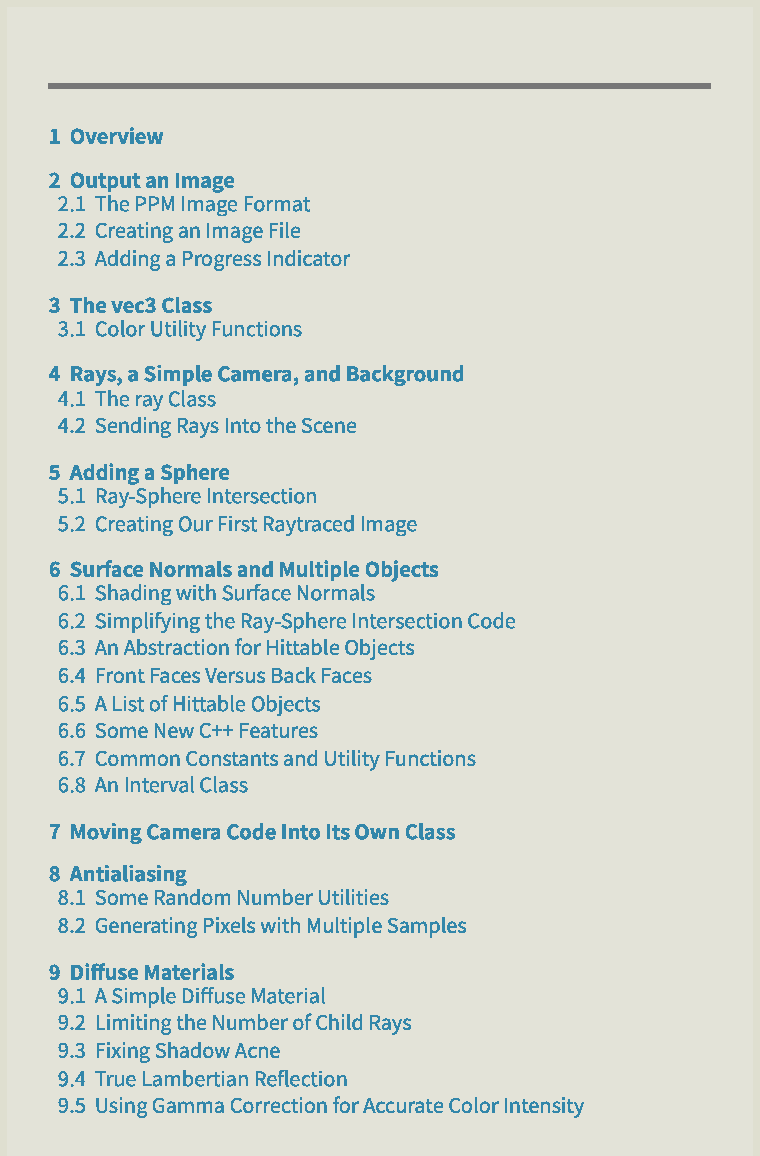
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**Ray Tracing in One Weekend**  
 一个周末的光线追踪

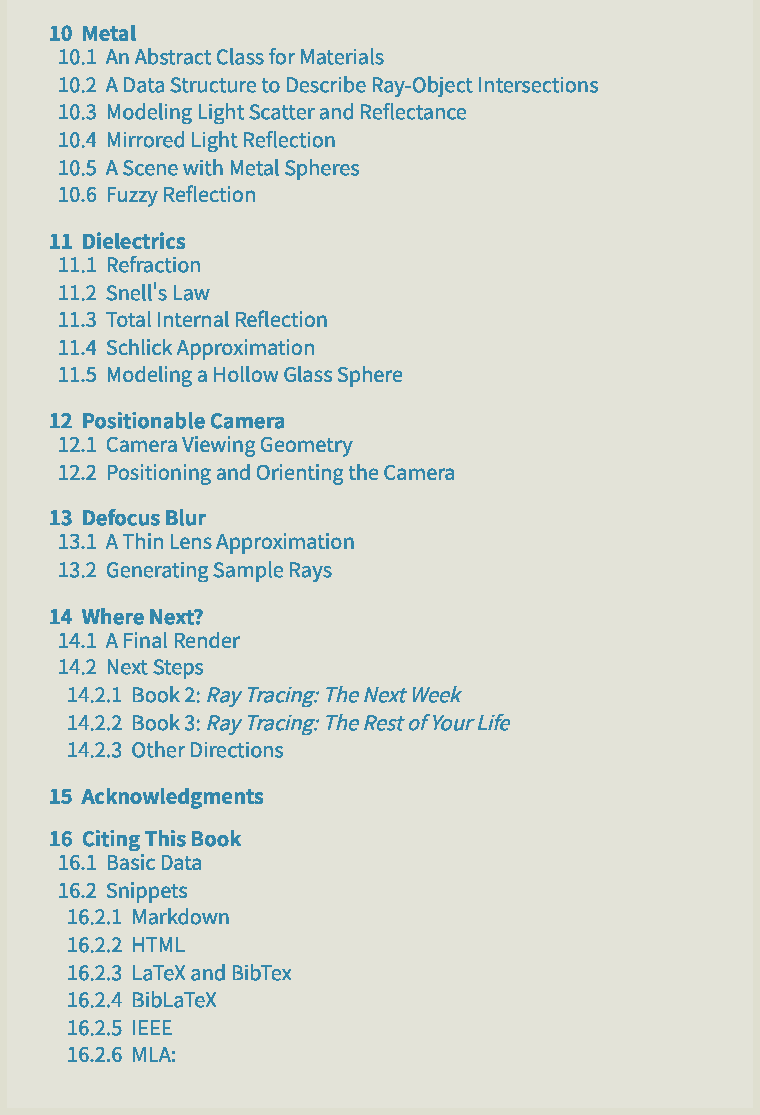


https://raytracing.github.io/books/RayTracingInOneWeekend.html 1/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 1/120

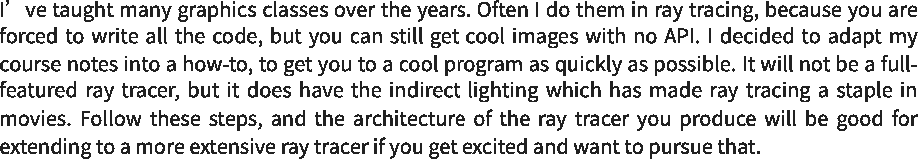


**Contents**  
 内容

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



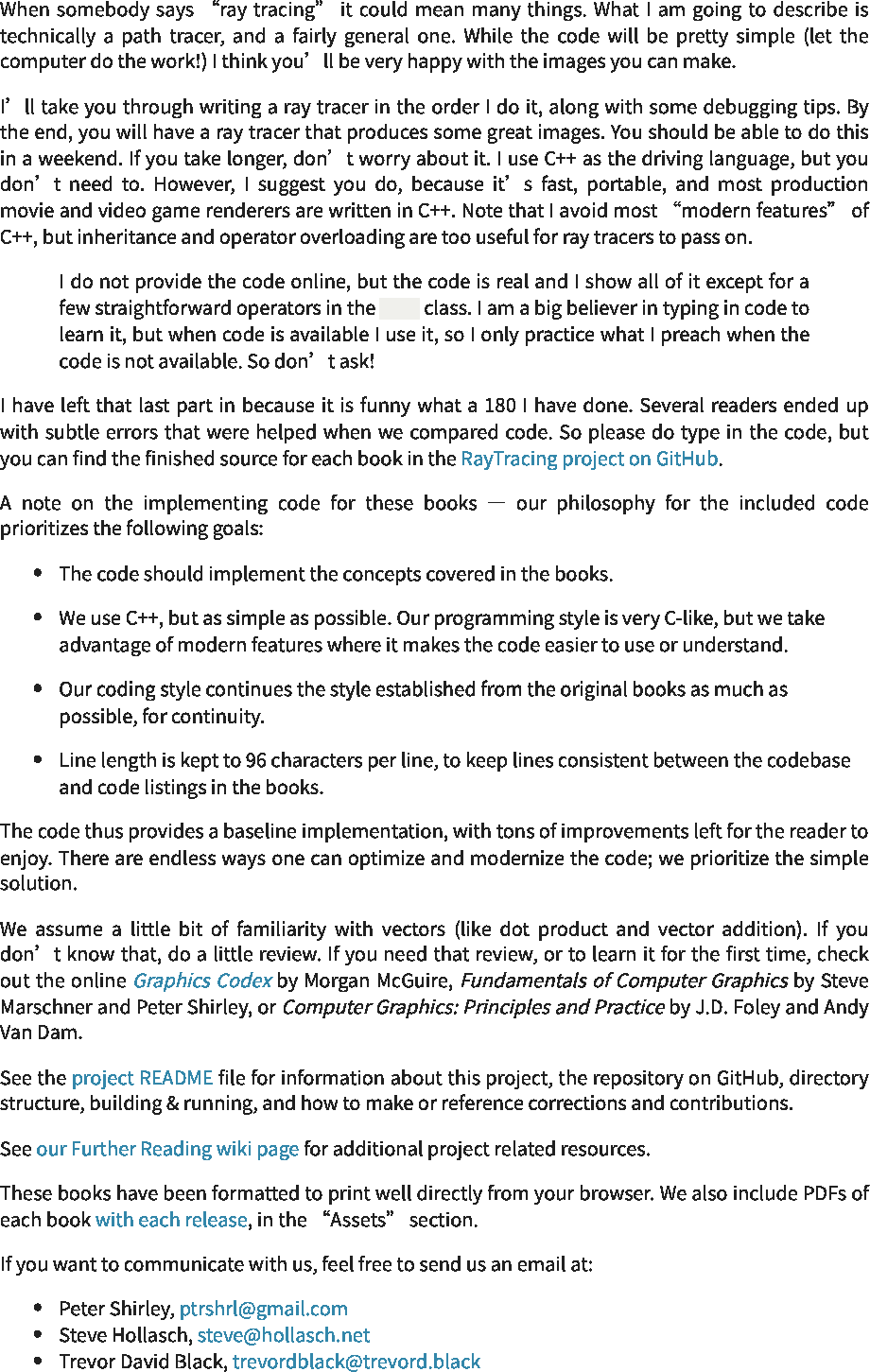
1. Overview  
 1.概述



https://raytracing.github.io/books/RayTracingInOneWeekend.html 2/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 2/120

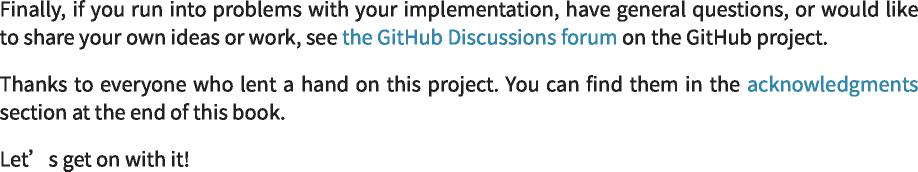
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 3/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 3/120



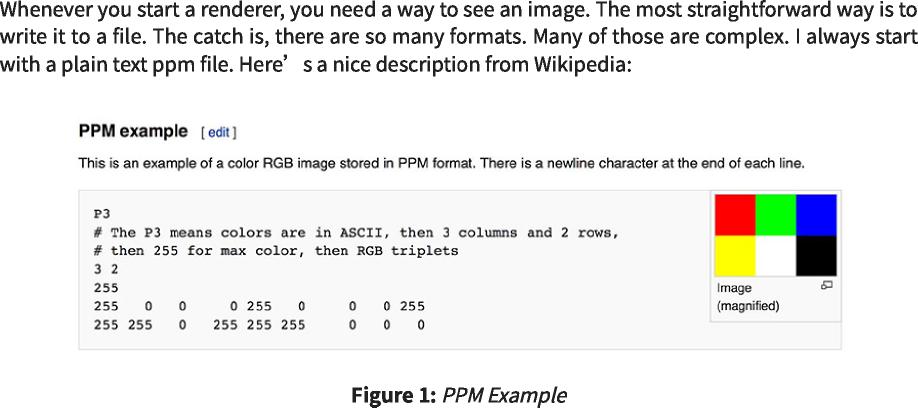
vec3  
 vec3

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**2. Output an Image**  
 2.输出图像

**2.1. The PPM Image Format**  
 2.1.PPM图像格式

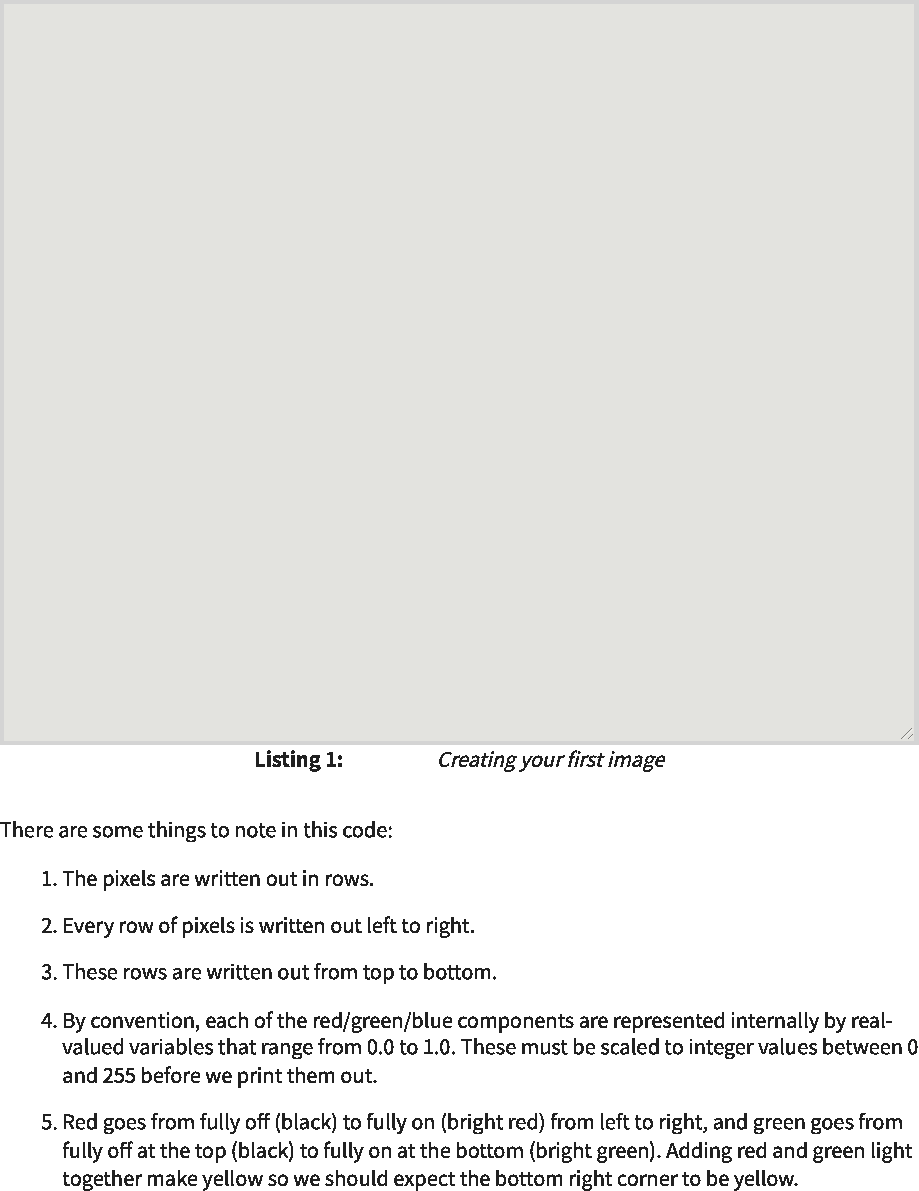


https://raytracing.github.io/books/RayTracingInOneWeekend.html 4/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 4/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**2.2. Creating an Image File**  
 2.2.创建图像文件



[main.cc]  
 [main.cc]

#include <iostream>  
 #include<iostream>

int main() {  
 int main(){

// Image  
 //图像

int image\_width = 256; int image\_height = 256;  
 int image\_width=256；int image\_height=256；

// Render  
 //渲染

std::cout << 'P3\n' << image\_width << • • << image\_height << '\n255\n';  
 std：：cout<<'P3\n'<<image\_width<<••<<image\_height<<'\n255\n';

for (int j = 0; j < image\_height; j++) { for (int i = 0; i < image\_width; i++) { auto r = double(i) / (image\_width-1); auto g = double(j) / (image\_height-1); auto b = 0.0;  
 for(int j=0;j<image\_height;j++){for(int i=0;i<image\_width;i++){auto r=double(i)/(image\_width-1);auto g=double(j)/(image\_height-1);auto b=0.0；

int ir = int(255.999 \* r);   
int ig = int(255.999 \* g);   
int ib = int(255.999 \* b);  
 int ir=int(255.999\*r)； int ig=int(255.999\*g)； int ib=int(255.999\*b)；

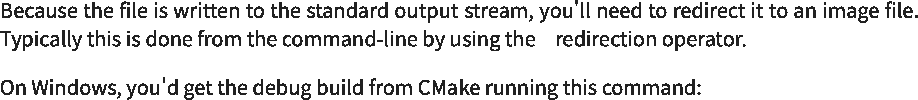
std::cout << ir << • • << ig << • • << ib << •\n•;  
 std：：cout<<ir<<••<<ig<<••<<ib<<•\n•;

}

}

}

https://raytracing.github.io/books/RayTracingInOneWeekend.html 5/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 5/120



>

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**cmake -B build**  
 CMake-B构建

**cmake --build build**  
 cmake--构建构建





**build\Debug\inOneWeekend.exe > image.ppm**  
 build\Debug\inOneWeekend.exe>image.ppm

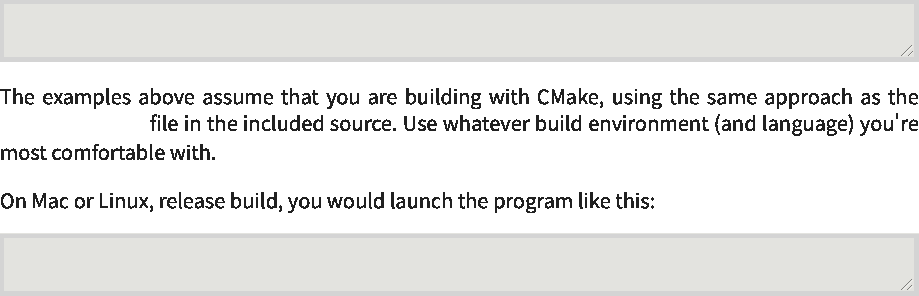


**cmake --build build --config release**  
 cmake--build build--config发布





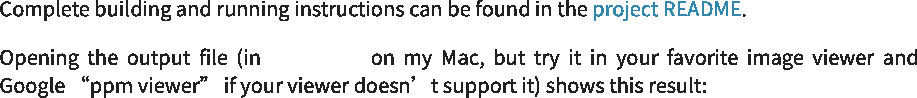




CMakeLists.txt  
 CMakeLists.txt

**build\**Release**\inOneWeekend.exe > image.ppm**  
 build\Release\inOneWeekend.exe>image.ppm

**build/inOneWeekend > image.ppm**  
 build/inOneWeekend>image.ppm

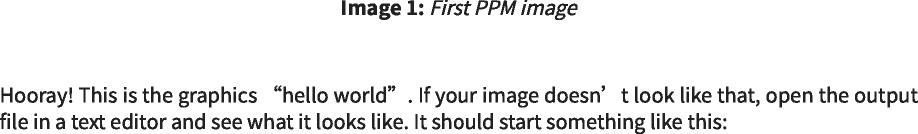


ToyViewer  
 玩具查看器

https://raytracing.github.io/books/RayTracingInOneWeekend.html 6/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 6/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





https://raytracing.github.io/books/RayTracingInOneWeekend.html 7/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 7/120

P3  
 P3

256 256

255

0 0 0

1 0 0

2 0 0

3 0 0

4 0 0

5 0 0

6 0 0

7 0 0

8 0 0

9 0 0

10 0 0

11 0 0

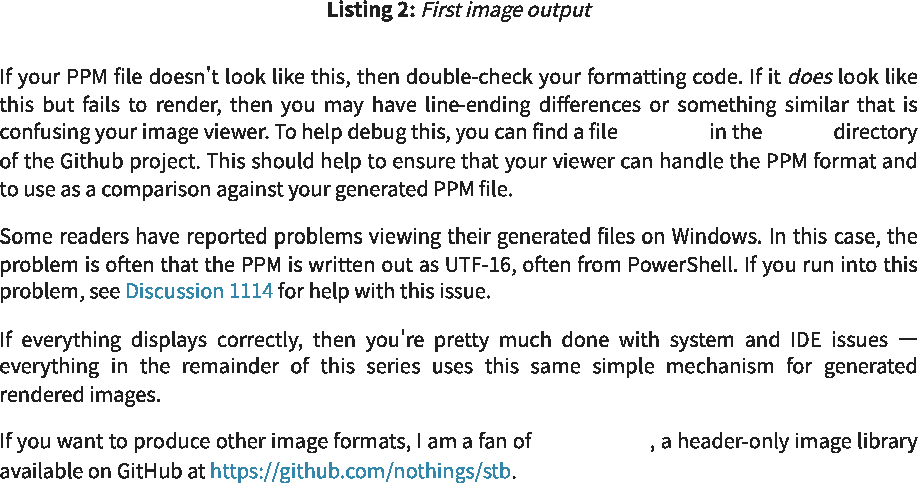
12 0 0

...



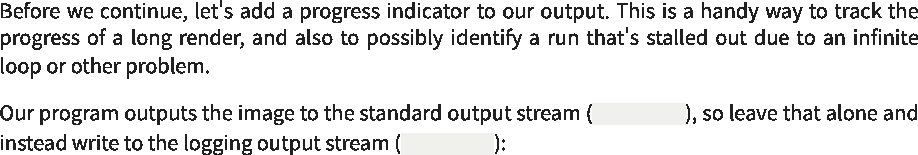
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**2.3. Adding a Progress Indicator**  
 2.3.添加进度指示器



stb\_image.h  
 stb\_image.h

test.ppm images  
 测试.ppmimages

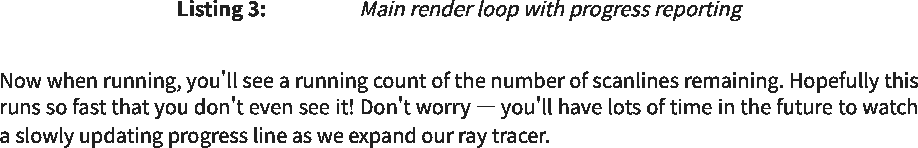


std::clog  
 标准：：堵塞

std::cout  
 std：：cout

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| for (int j = **0**; j < image\_height; -i--i-j) {  for(int j=0； j<image\_height；-i--i-j){ |  |  |  |  |  |
| std::clog << "\rScanlines remaining: " << (image\_height  std：：clog<<"\rScanlines remaining："<<(image\_height  std::flush;  std：：flush； | - j)  -j) | << | ' | ' | << |
| for (int i = **0**; i < image\_width; i-i--i-) {  为了 (int i=0； i<image\_width;i-i---i--){  auto r = double(i) / (image\_width**-1**);  auto r=double(i)/(image\_width-1)；  auto g = double(j) / (image\_height**-1**); auto b = **0.0**;  auto g=double(j)/(image\_height-1)；自动b=0.0；  int ir = int(**255.999** \* r);  int ig = int(**255.999** \* g);  int ib = int(**255.999** \* b);  int ir=int(255.999\*r)； int ig=int(255.999\*g)； int ib=int(255.999\*b)；  std::cout << ir << ' ' << ig << ' ' << ib << '\n';  std：：cout<<ir<< “ <<ig<< “ <<ib<< '\n';  }  } |  |  |  |  |  |
| std::clog << "\rDone. \n";  std：：clog<<"\rDone.\n"； |  |  |  |  |  |
|  |  |  |  |  |

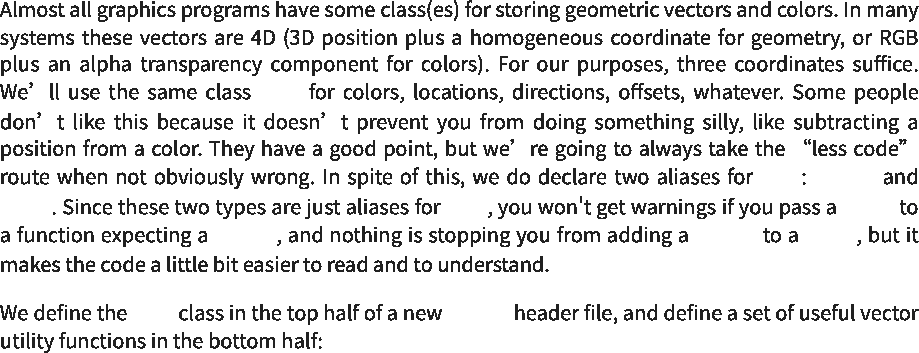
https://raytracing.github.io/books/RayTracingInOneWeekend.html 8/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 8/120



[main.cc]  
 [main.cc]

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**3. The vec3 Class**  
 3.VEC3类



vec3 point3  
 vec3点3

color vec3 color  
 colorvec3color

point3 point3 color  
 point3point3color

vec3 vec3.h  
 vec3vec3.h

vec3  
 vec3

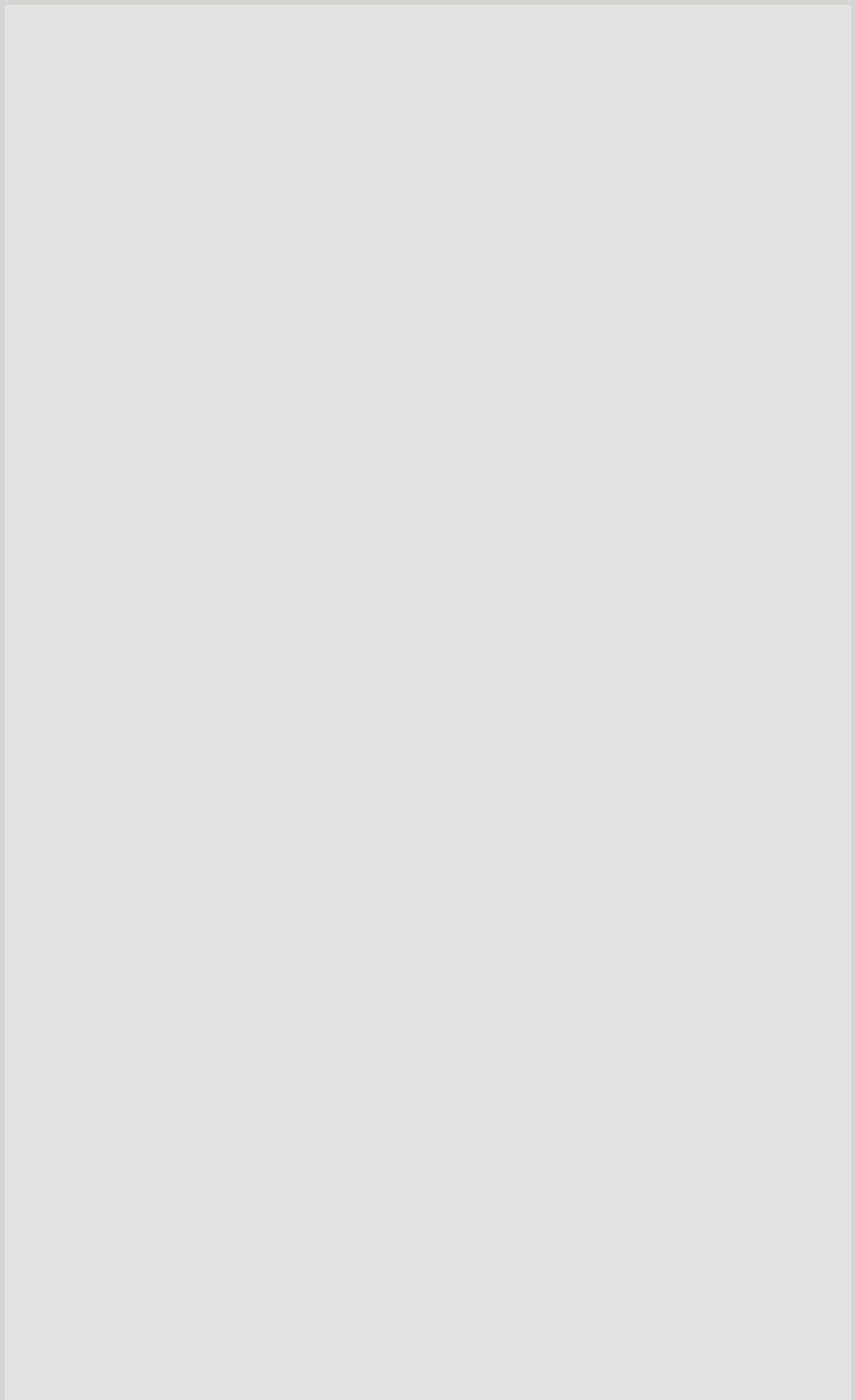
https://raytracing.github.io/books/RayTracingInOneWeekend.html 9/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 9/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 10/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 10/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 11/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 11/120



#ifndef VEC3\_H #define VEC3\_H  
 #ifndef VEC3\_H#define VEC3\_H

#include <cmath> #include <iostream>  
 #include<cmath>#include<iostream>

class vec3 {  
 vec3类{

public:  
 公众：

double e[3];  
 双e[3]；

vec3() : e{0,0,0} {}  
 vec3()：e{0,0,0}{}

vec3(double e0, double e1, double e2) : e{e0, e1, e2} {}  
 vec3(双e0，双e1，双e2)：e{e0，e1，e2}{}

|  |  |  |  |
| --- | --- | --- | --- |
| double **x**() const  double x()常量 | { | return  返回 | e[0]; }  e[0];} |
| double **y**() const  双y()常量 | { | return  返回 | 1. } |
| double **z**() const  双z()常量 | { | return  返回 | 1. } |

vec3 operator-() const { return vec3(-e[0], -e[1], -e[2]); }  
 vec3运算符-()const{return vec3(-e[0],-e[1],-e[2]);}

double operator[](int i) const { return e[i]; }  
 double operator[](int i)const{return e[i];}

double& operator[](int i) { return e[i]; }  
 double&operator[](int i){return e[i];}

vec3& operator+=(const vec3& v) { e[0] += v.e[0];  
 vec3&operator+=(const vec3&v){e[0]+=v.e[0]；

1. += v.e[1];  
    +=v.e[1]；
2. += v.e[2];   
   return \*this;  
    +=v.e[2]； 返回\*这个；

}

vec3& operator\*=(double t) { e[0] \*= t;  
 vec 3&operator\*=(double t){e[0]\*=t；

1. \*= t;  
    \*=t；
2. \*= t;   
   return \*this;  
    \*=t； 返回\*这个；

}

vec3& operator/=(double t) { return \*this \*= 1/t;  
 vec3&operator/=(double t){return\*this\*=1/t；

}

double **length**() const {  
 double length()const{

return std::sqrt(length\_squared());  
 return std：：sqrt(length\_squared())；

}

double **length\_squared**() const {  
 double length\_squared()const{

return e[0]\*e[0] + e[1]\*e[1] + e[2]\*e[2];  
 返回e[0]\*e[0]+e[1]\*e[1]+e[2]\*e[2]；

}

};

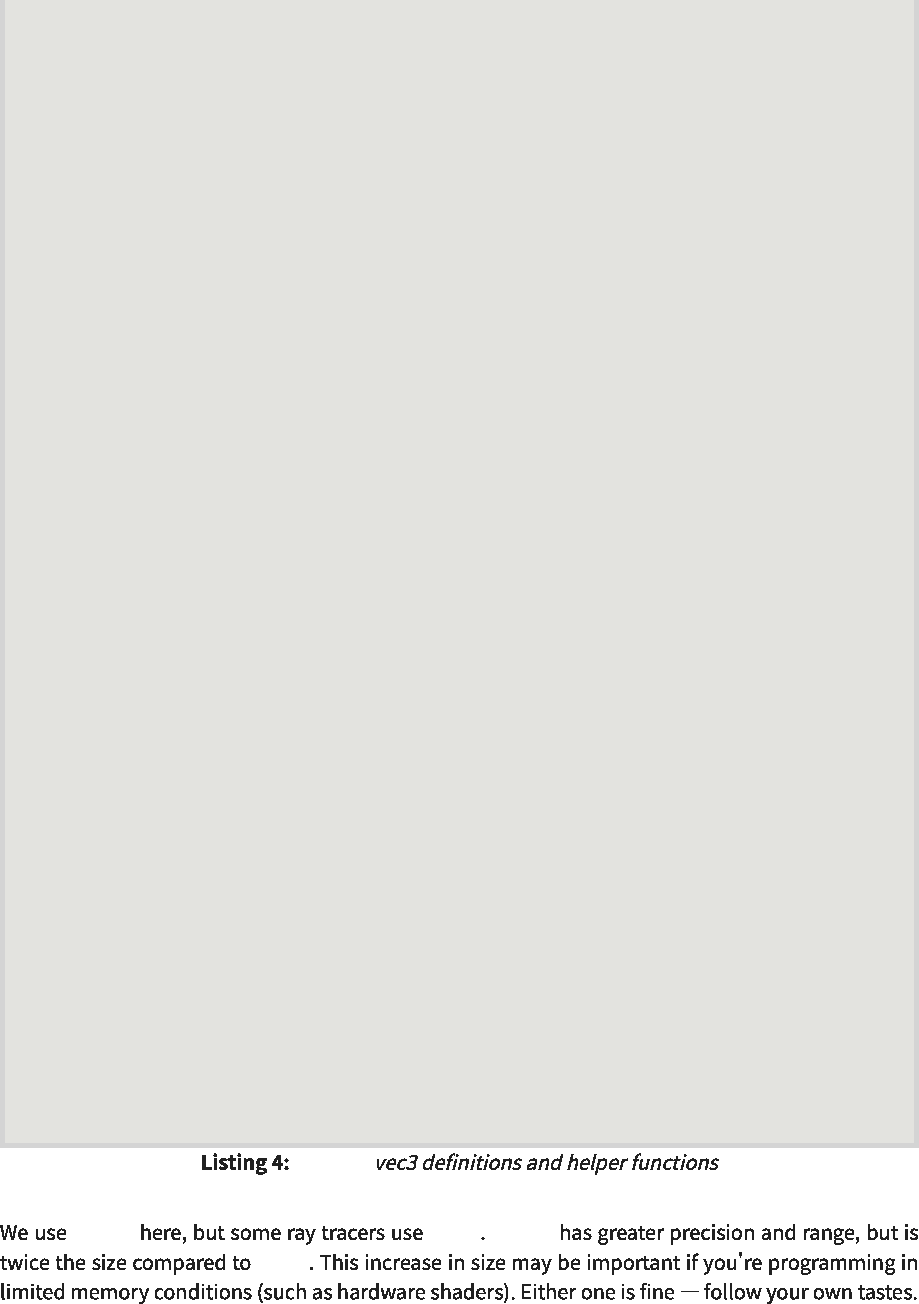
// point3 is just an alias for vec3, but useful for geometric clarity in the code. using point3 = vec3;  
 //point3只是vec3的别名，但对于代码中的几何清晰度很有用。使用point3=vec3；

// Vector Utility Functions  
 //向量效用函数

inline std::ostream& operator<<(std::ostream& out, const vec3& v) { return out << v.e[0] << • • << v.e[1] << • • << v.e[2];  
 内联std：：ostream&operator<<(std：：ostream&out，const vec3&v){return out<<v.e[0]<<••<<v.e[1]<<••<<v.e[2]；

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**3.1. Color Utility Functions**  
 3.1.颜色实用函数



[vec3.h]  
 [vec3.h]

double float double  
 双浮双

float  
 浮动

}

inline vec3 operator+(const vec3& u, const vec3& v) {  
 内联vec3运算符+(const vec3&u，const vec3&v){

return vec3(u.e[0] + v.e[0], u.e[1] + v.e[1], u.e[2] + v.e[2]);  
 返回vec3(u.e[0]+v.e[0]，u.e[1]+v.e[1]，u.e[2]+v.e[2])；

}

inline vec3 operator-(const vec3& u, const vec3& v) {  
 内联vec3运算符-(const vec3&u，const vec3&v){

return vec3(u.e[0] - v.e[0], u.e[1] - v.e[1], u.e[2] - v.e[2]);  
 返回vec3(u.e[0]-v.e[0]，u.e[1]-v.e[1]，u.e[2]-v.e[2])；

}

inline vec3 operator\*(const vec3& u, const vec3& v) {  
 内联vec3运算符\*(const vec3&u，const vec3&v){

return vec3(u.e[0] \* v.e[0], u.e[1] \* v.e[1], u.e[2] \* v.e[2]);  
 返回vec3(u.e[0]\*v.e[0]，u.e[1]\*v.e[1]，u.e[2]\*v.e[2])；

}

inline vec3 operator\*(double t, const vec3& v) { return vec3(t\*v.e[0], t\*v.e[1], t\*v.e[2]);  
 内联vec3运算符\*(double t，const vec3&v){返回vec3(t\*v.e[0]，t\*v.e[1]，t\*v.e[2])；

}

inline vec3 operator\*(const vec3& v, double t) { return t \* v;  
 内联vec3运算符\*（常量vec3&v，双t）{返回t\*v；

}

inline vec3 operator/(const vec3& v, double t) { return (1/t) \* v;  
 内联vec3运算符/（常量vec3&v，双t）{返回（1/t）\*v；

}

inline double dot(const vec3& u, const vec3& v) {  
 内联双点(const vec3&u，const vec3&v){

return u.e[0] \* v.e[0]  
 返回u.e[0]\*v.e[0]

+ u.e[1] \* v.e[1]  
 +u.e[1]\*v.e[1]

+ u.e[2] \* v.e[2];  
 +u.e[2]\*v.e[2]；

}

inline vec3 cross(const vec3& u, const vec3& v) { return vec3(u.e[1] \* v.e[2] - u.e[2] \* v.e[1], u.e[2] \* v.e[0] - u.e[0] \* v.e[2], u.e[0] \* v.e[1] - u.e[1] \* v.e[0]);  
 内联vec3交叉(const vec3&u，const vec3&v){返回vec3(u.e[1]\*v.e[2]-u.e[2]\*v.e[1]，u.e[2]\*v.e[0]-u.e[0]\*v.e[2]，u.e[0]\*v.e[1]-u.e[1]\*v.e[0])；

}

inline vec3 unit\_vector(const vec3& v) { return v / v.length();  
 inline vec3 unit\_vector(const vec3&v){return v/v.length()；

}

#endif  
 #endif

https://raytracing.github.io/books/RayTracingInOneWeekend.html 12/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 12/120



vec3 color.h  
 vec3color.h

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

#ifndef COLOR\_H   
#define COLOR\_H  
 #ifndef COLOR\_H #定义颜色\_H

#include 'vec3.h'   
#include <iostream>   
using color = vec3;  
 #包括“vec3.h” #include<iostream> 使用color=vec3；

void **write\_color**(std::ostream& out, const color& pixel\_color) {  
 void write\_color(std：：ostream&out，const color&pixel\_color){

auto r = pixel\_color.x();  
 auto r=pixel\_color.x()；

auto g = pixel\_color.y();  
 auto g=pixel\_color.y()；

auto b = pixel\_color.z();  
 auto b=pixel\_color.z()；

// Translate the [0,1] component values to the byte range [0,255].  
 //将[0,1]分量值转换为字节范围[0,255]。

int rbyte = int(255.999 \* r);  
 int rbyte=int(255.999\*r)；

int gbyte = int(255.999 \* g);  
 int gbyte=int(255.999\*g)；

int bbyte = int(255.999 \* b);  
 int bbyte=int(255.999\*b)；

// Write out the pixel color components.  
 //写出像素颜色分量。

out << rbyte << • • << gbyte << • • << bbyte << •\n•;  
 out<<rbyte<<••<<gbyte<<••<<bbyte<<•\n•;

}

#endif  
 #endif



|  |  |  |
| --- | --- | --- |
|  | [color.h]  [颜色.h] |  |

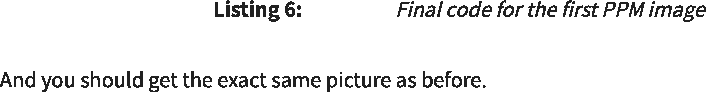
https://raytracing.github.io/books/RayTracingInOneWeekend.html 13/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 13/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



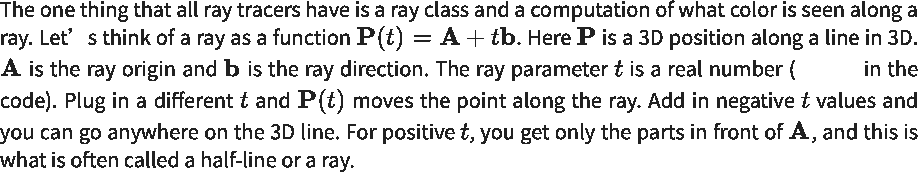
|  |  |  |
| --- | --- | --- |
|  |  |  |
| #include 'color.h'  #include 'vec3.h'  #包括“color.h” #包括“vec3.h” |  |  |
| #include <iostream>  #include<iostream>  int main() {  int main(){  // Image  //图像  int image\_width = 256;  int image\_height = 256;  int image\_width=256； int image\_height=256；  // Render  //渲染  std::cout << 'P3\n' << image\_width << • • << image\_height <<  std：：cout<<'P3\n'<<image\_width<< •• <<图像高度<<  for (int j = 0; j < image\_height; j++) {  for(int j=0;j<image\_height;j++){  std::clog << '\rScanlines remaining: ' << (image\_height  std：：clog<<'\rScanlines remaining：'<<(image\_height  std::flush;  std：：flush；  for (int i = 0; i < image\_width; i++) {  for(int i=0;i<image\_width;i++){ | '\n255\n';  '\n255\n';  - j) << • •  -j)<< •• | << |
| auto pixel\_color = color(double(i)/(image\_width-1),  auto pixel\_color=color(double(i)/(image\_width-1)，  double(j)/(image\_height-1), 0);  double(j)/(image\_height-1),0)；  write\_color(std::cout, pixel\_color);  write\_color(std：：cout，pixel\_color)； |  |  |
| }  }  std::clog << '\rDone. \n';  }  std：：clog<<'\r完成。\n'; } |  |  |

**4. Rays, a Simple Camera, and Background**  
 4.光线、简单的相机和背景



[main.cc]  
 [main.cc]

**4.1. The ray Class**  
 4.1.射线类

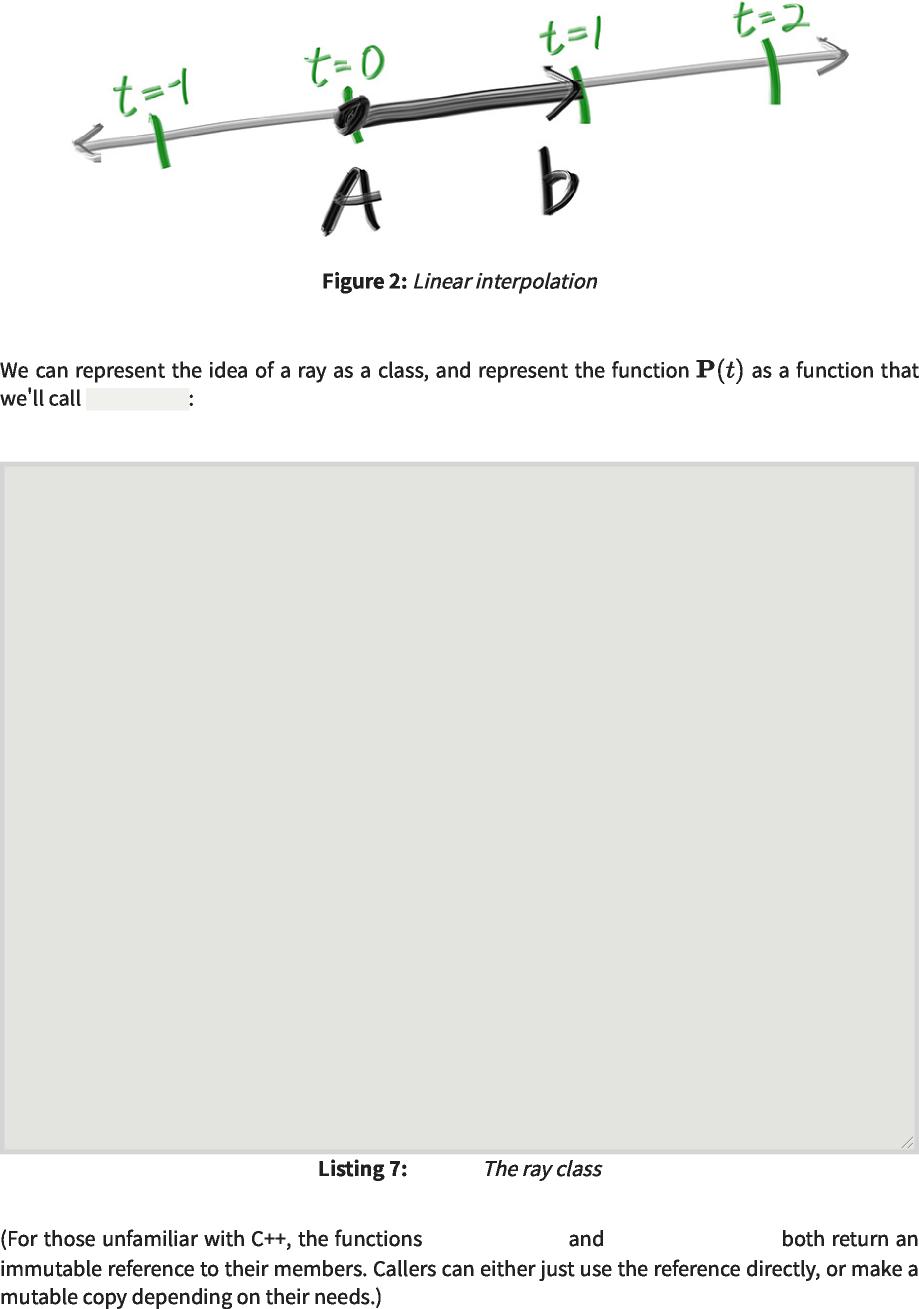


double  
 double

https://raytracing.github.io/books/RayTracingInOneWeekend.html 14/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 14/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 15/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 15/120



ray::at(t)  
 射线：：at(t)

#ifndef RAY\_H #define RAY\_H  
 #ifndef RAY\_H#define RAY\_H

#include "vec3.h"  
 #包括“vec3.h”

class ray { public: ray() {}  
 class ray{public：ray(){}

ray(const point3& origin, const vec3& direction) : orig(origin), dir(direction)  
 射线（常量点3&原点，常量vec3&方向）：orig（原点），dir（方向）

{}

const point3& origin() const { return orig; } const vec3& direction() const { return dir; }  
 const point3&origin()const{return orig;}const vec3&direction()const{return dir;}

point3 at(double t) const { return orig + t\*dir;  
 点3 at(double t)const{return orig+t\*dir；

}

private:  
 私人：

point3 orig; vec3 dir;  
 点3 orig；vec3 dir；

};

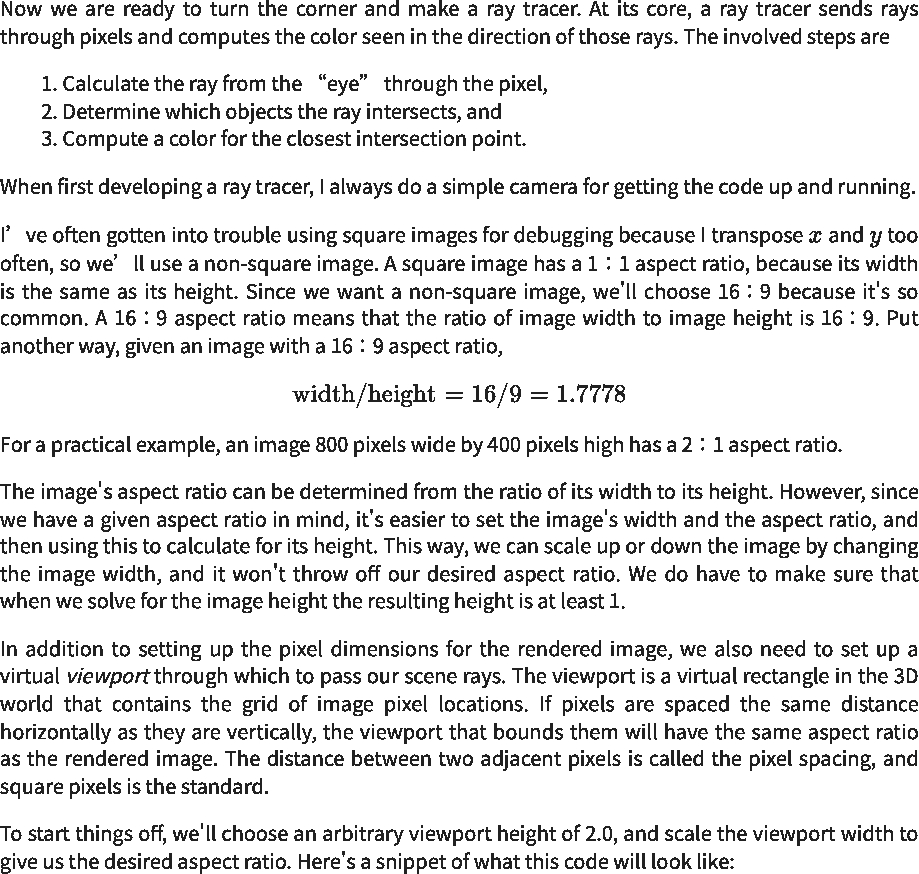
#endif  
 #endif

[ray.h]  
 [雷.h]

ray::origin() ray::direction()  
 射线：：原点()射线：：方向()

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**4.2. Sending Rays Into the Scene**  
 4.2.将光线发送到场景中



https://raytracing.github.io/books/RayTracingInOneWeekend.html 16/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 16/120

auto aspect\_ratio = 16.0 / 9.0;   
int image\_width = 400;  
 auto aspect\_ratio=16.0/9.0； int image\_width=400；

// Calculate the image height, and ensure that it's at least 1.  
 //计算图像高度，并确保它至少为1。

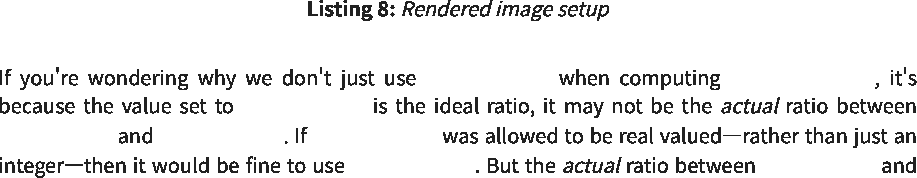
int image\_height = int(image\_width / aspect\_ratio);   
image\_height = (image\_height < 1) ? 1 : image\_height;  
 int image\_height=int(image\_width/aspect\_ratio)； image\_height=(image\_height<1)？1：image\_height；

// Viewport widths less than one are ok since they are real valued.  
 //视口宽度小于1是可以的，因为它们是实值。

auto viewport\_height = 2.0;  
 auto viewport\_height=2.0；

auto viewport\_width = viewport\_height \* (double(image\_width)/image\_height);  
 auto viewport\_width=viewport\_height\*(double(image\_width)/image\_height)；





aspect\_ratio viewport\_width  
 纵横比视口宽度

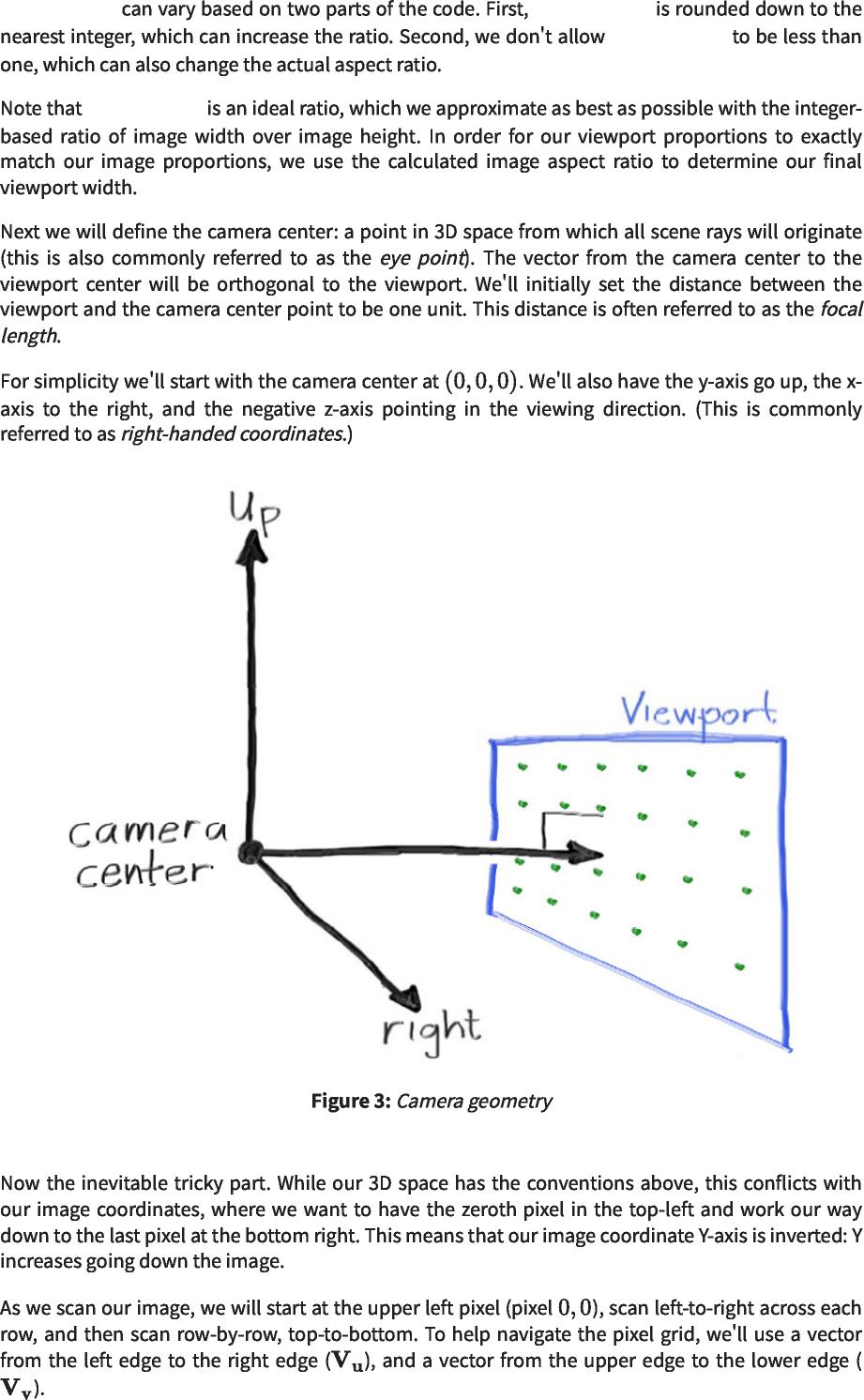
aspect\_ratio  
 纵横比

image\_width image\_height image\_height  
 image\_width image\_height image\_height

aspect\_ratio image\_width  
 纵横比图像宽度

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 17/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 17/120

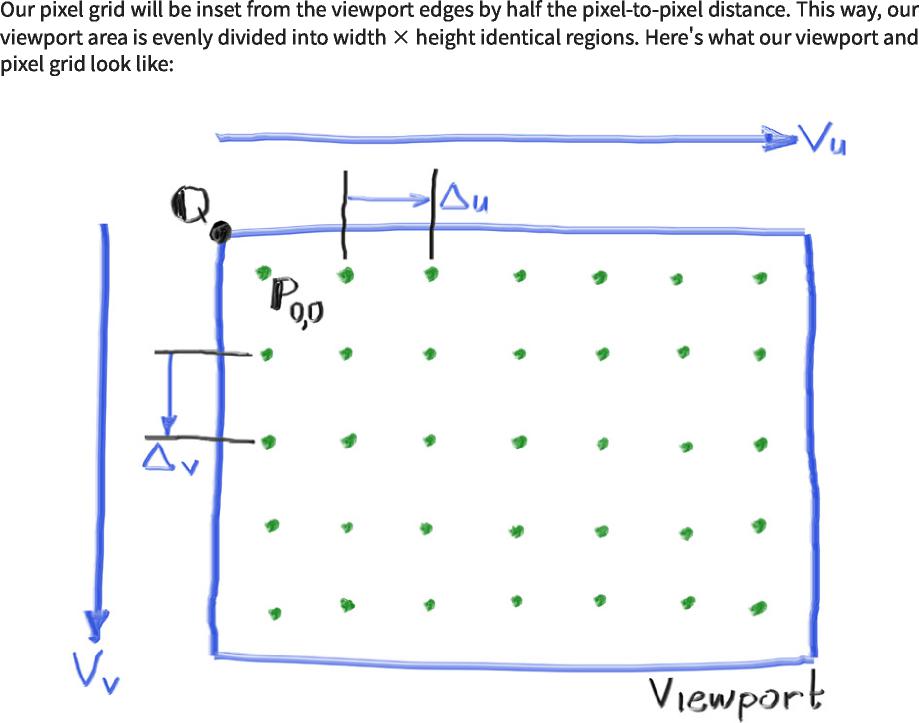


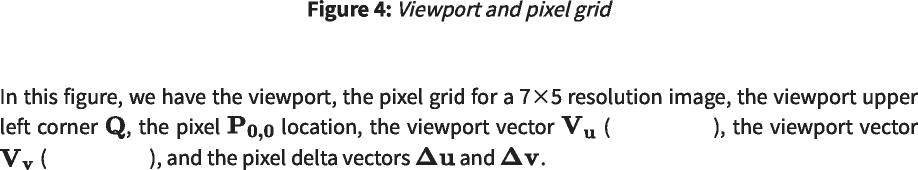
image\_height image\_height  
 image\_heightimage\_height

image\_height  
 图像高度

aspect\_ratio  
 纵横比

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





viewport\_v  
 视口\_v

viewport\_u  
 视口\_u

https://raytracing.github.io/books/RayTracingInOneWeekend.html 18/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 18/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



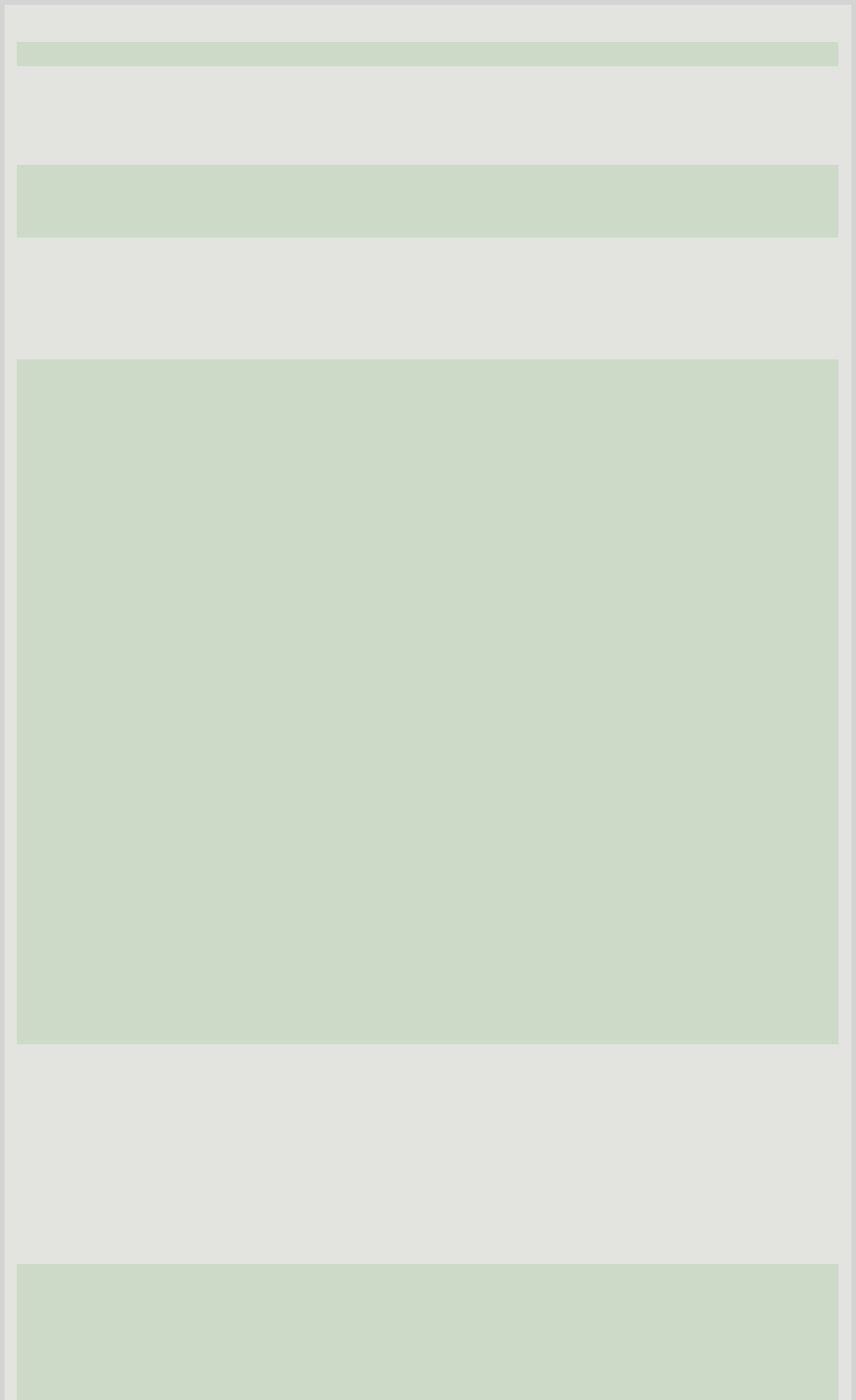
ray\_color(const ray& r)  
 ray\_color（常量光线&r）



https://raytracing.github.io/books/RayTracingInOneWeekend.html 19/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 19/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 20/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 20/120



#include 'color.h'  
 #包括“color.h”

#include 'ray.h'  
 #包括“ray.h”

#include 'vec3.h'  
 #包括“vec3.h”

#include <iostream>  
 #include<iostream>

color ray\_color(const ray& r) {  
 color ray\_color(const ray&r){

return color(0,0,0);  
 返回颜色(0,0,0)；

}

int main() {  
 int main(){

// Image  
 //图像

auto aspect\_ratio = 16.0 / 9.0;  
 auto aspect\_ratio=16.0/9.0；

int image\_width = 400;  
 int image\_width=400；

// Calculate the image height, and ensure that it's at least 1. int image\_height = int(image\_width / aspect\_ratio); image\_height = (image\_height < 1) ? 1 : image\_height;  
 //计算图像高度，并确保它至少为1。int image\_height=int(image\_width/aspect\_ratio)；image\_height=(image\_height<1)？1：image\_height；

// Camera  
 //相机

auto focal\_length = 1.0;  
 auto focal\_length=1.0；

auto viewport\_height = 2.0;  
 auto viewport\_height=2.0；

auto viewport\_width = viewport\_height \* (double(image\_width)/image\_height);  
 auto viewport\_width=viewport\_height\*(double(image\_width)/image\_height)；

auto camera\_center = point3(0, 0, 0);  
 auto camera\_center=point3(0,0,0)；

// Calculate the vectors across the horizontal and down the vertical viewport  
 //计算水平视口和垂直视口向下的向量

edges.  
 边缘。

auto viewport\_u = vec3(viewport\_width, 0, 0);  
 auto viewport\_u=vec3(viewport\_width,0,0)；

auto viewport\_v = vec3(0, -viewport\_height, 0);  
 auto viewport\_v=vec3(0,-viewport\_height,0)；

// Calculate the horizontal and vertical delta vectors from pixel to pixel.  
 //计算从像素到像素的水平和垂直增量向量。

auto pixel\_delta\_u = viewport\_u / image\_width;  
 auto pixel\_delta\_u=viewport\_u/image\_width；

auto pixel\_delta\_v = viewport\_v / image\_height;  
 auto pixel\_delta\_v=viewport\_v/image\_height；

// Calculate the location of the upper left pixel.  
 //计算左上像素的位置。

auto viewport\_upper\_left = camera\_center  
 auto viewport\_upper\_left=camera\_center

- vec3(0, 0, focal\_length) - viewport\_u/2 -  
 -vec3(0,0,focal\_length)-viewport\_u/2-

viewport\_v/2;  
 viewport\_v/2；

auto pixel00\_loc = viewport\_upper\_left + 0.5 \* (pixel\_delta\_u + pixel\_delta\_v);  
 auto pixel00\_loc=viewport\_upper\_left+0.5\*(pixel\_delta\_u+pixel\_delta\_v)；

// Render  
 //渲染

std::cout << 'P3\n' << image\_width << ' ' << image\_height << '\n255\n';  
 std：：cout<<'P3\n'<<image\_width<<''<<image\_height<<'\n255\n';

for (int j = 0; j < image\_height; j++) {  
 for(int j=0;j<image\_height;j++){

std::clog << '\rScanlines remaining: ' << (image\_height - j) << • • <<  
 std：：clog<<'\rScanlines remaining：'<<(image\_height-j)<<••<<

std::flush;  
 std：：flush；

for (int i = 0; i < image\_width; i++) {  
 for(int i=0;i<image\_width;i++){

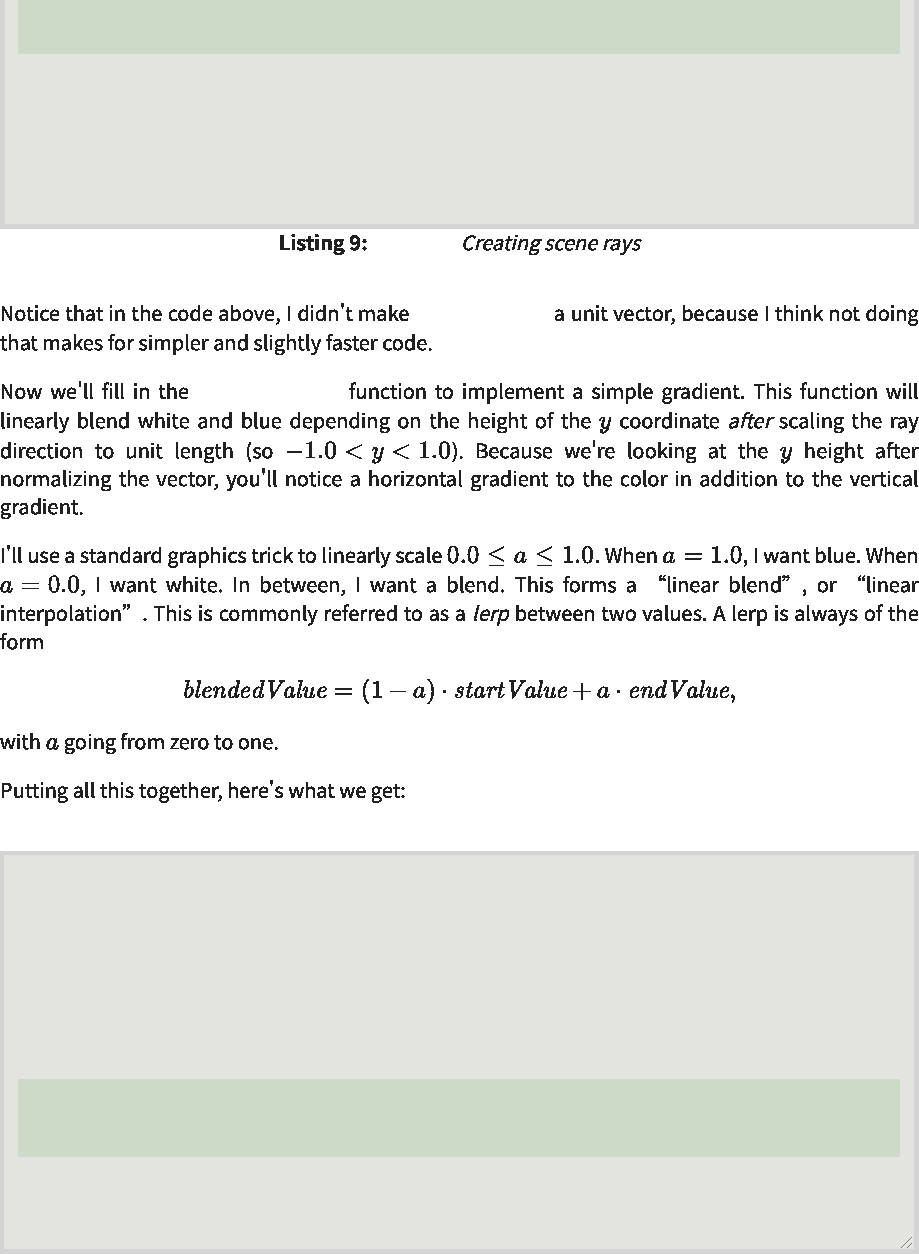
auto pixel\_center = pixel00\_loc + (i \* pixel\_delta\_u) + (j \*  
 auto pixel\_center=pixel00\_loc+(i\*pixel\_delta\_u)+(j\*

pixel\_delta\_v);  
 pixel\_delta\_v)；

auto ray\_direction = pixel\_center - camera\_center;  
 auto ray\_direction=pixel\_center-camera\_center；

ray r(camera\_center, ray\_direction);  
 光线r(相机中心，光线方向)；

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



}

}

}

[main.cc]  
 [main.cc]

ray\_direction  
 光线方向

ray\_color(ray)  
 ray\_color（光线）

std::clog << '\rDone. \n';  
 std：：clog<<'\r完成。\n';

write\_color(std::cout, pixel\_color);  
 write\_color(std：：cout，pixel\_color)；

color pixel\_color = ray\_color(r);  
 color pixel\_color=ray\_color(r)；

#include 'color.h' #include 'ray.h' #include 'vec3.h'  
 #包括“color.h”#包括“ray.h”#包括“vec3.h”

#include <iostream>  
 #include<iostream>

color ray\_color(const ray& r) {  
 color ray\_color(const ray&r){

vec3 unit\_direction = unit\_vector(r.direction());  
 vec3 unit\_direction=unit\_vector(r.direction())；

auto a = 0.5\*(unit\_direction.y() + 1.0);  
 auto a=0.5\*(unit\_direction.y()+1.0)；

return (1.0-a)\*color(1.0, 1.0, 1.0) + a\*color(0.5, 0.7, 1.0);  
 返回(1.0-a)\*颜色(1.0,1.0,1.0)+A\*颜色(0.5,0.7,1.0)；

}

...

|  |  |  |
| --- | --- | --- |
|  | [main.cc]  [main.cc] |  |

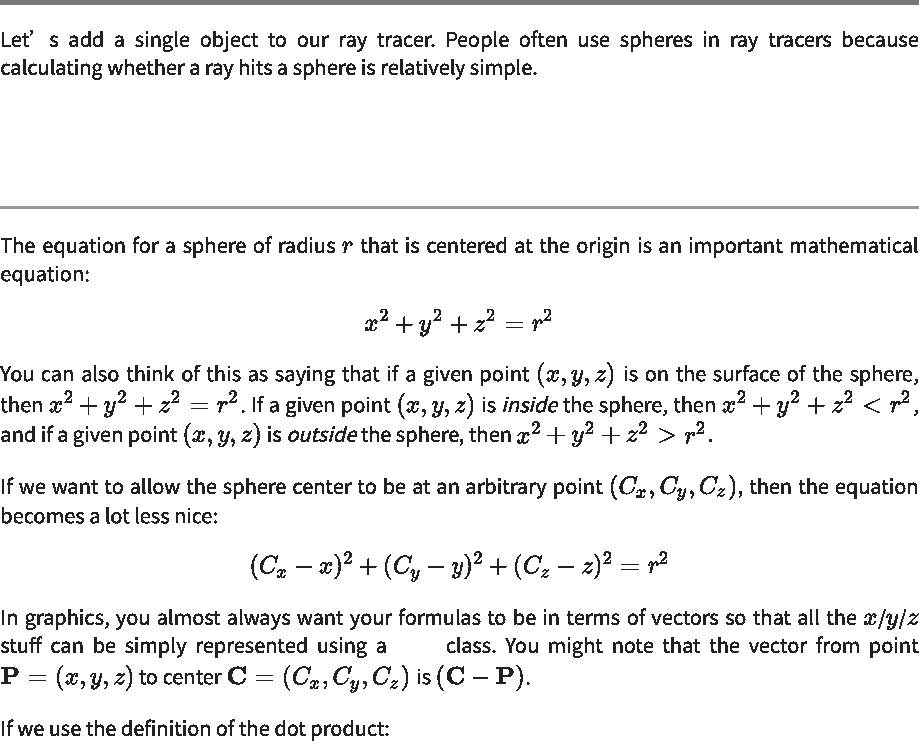
https://raytracing.github.io/books/RayTracingInOneWeekend.html 21/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 21/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**5. Adding a Sphere**  
 5.添加球体

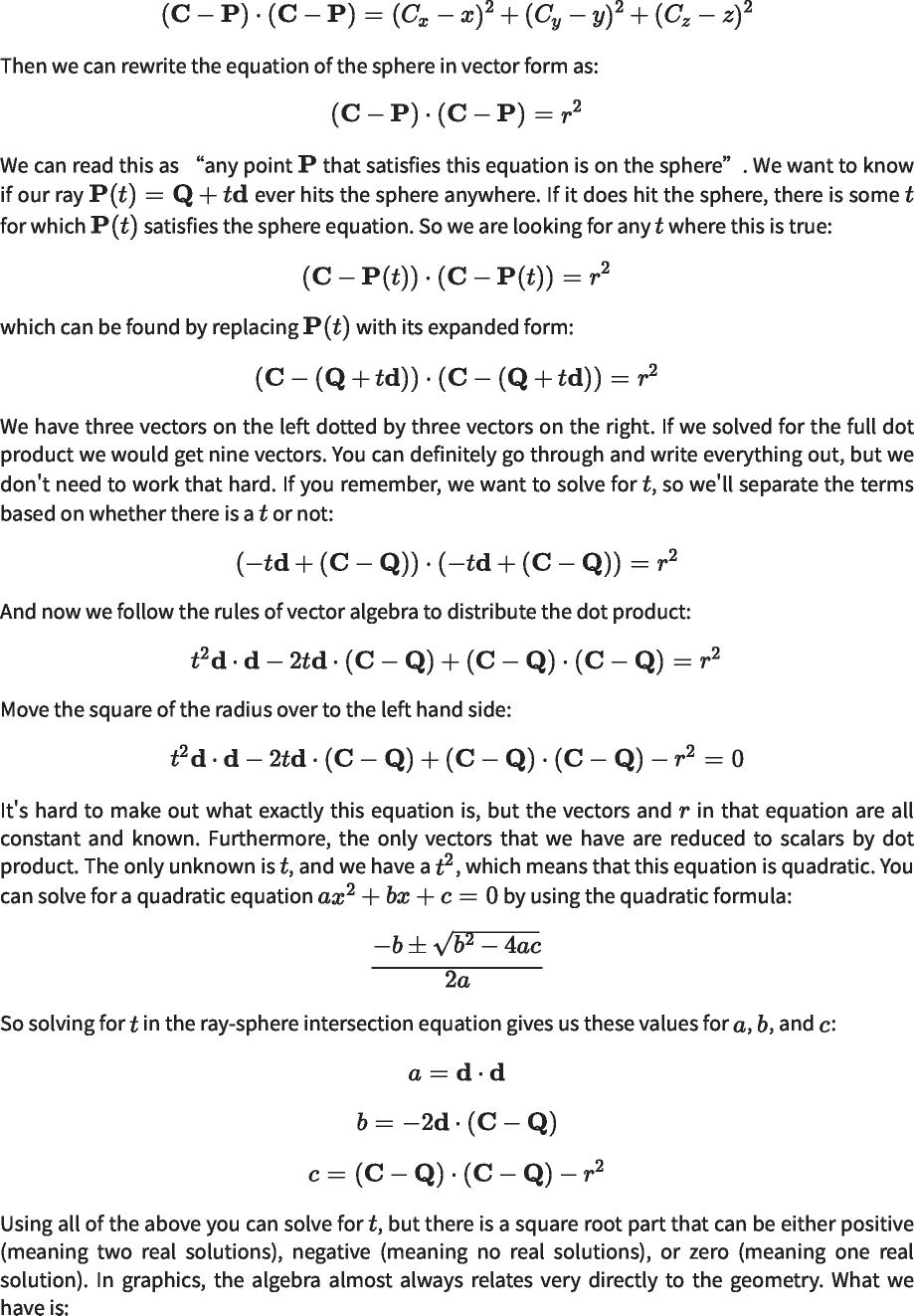
https://raytracing.github.io/books/RayTracingInOneWeekend.html 22/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 22/120



**5.1. Ray-Sphere Intersection**  
 5.1.射线球相交

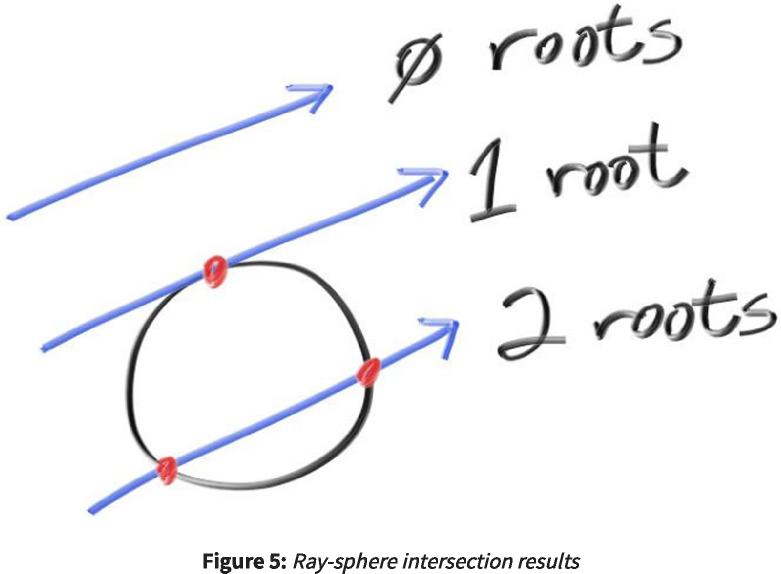
vec3  
 vec3

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



https://raytracing.github.io/books/RayTracingInOneWeekend.html 23/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 23/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**5.2. Creating Our First Raytraced Image**  
 5.2.创建我们的第一个光线跟踪图像

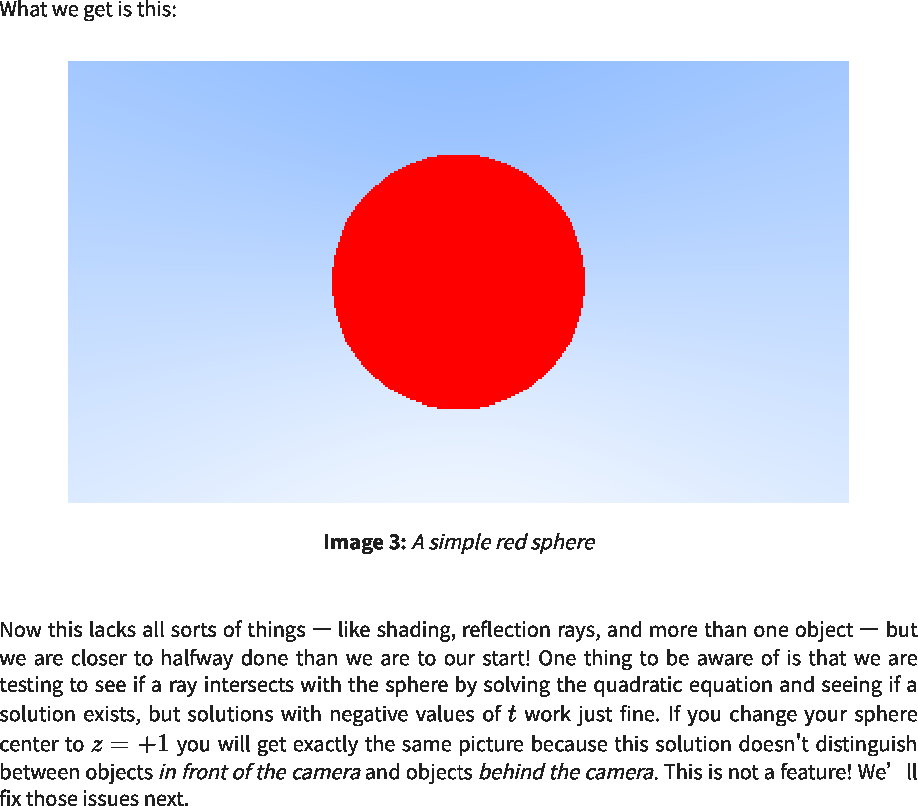


|  |  |
| --- | --- |
|  |  |
| bool **hit\_sphere**(const point3& center, double radius, const ray& r) vec3 oc = center - r.origin();  bool hit\_sphere(const point3&center，double radius，const ray&r)vec3 oc=center-r.origin()；  auto a = dot(r.direction(), r.direction());  auto a=dot(r.direction(),r.direction())；  auto b = -2.0 \* dot(r.direction(), oc);  auto b=-2.0\*dot(r.direction(),oc)；  auto c = dot(oc, oc) - radius\*radius;  自动c=点（oc，oc）-半径\*半径；  auto discriminant = b\*b - 4\*a\*c;  自动判别式=b\*b-4\*a\*c；  return (discriminant >= 0);  返回(判别式>=0)；  } | { |
| color **ray\_color**(const ray& r) {  color ray\_color(const ray&r){ |  |
| if (hit\_sphere(point3(0,0,-1), 0.5, r))  return color(1, 0, 0);  如果(hit\_sphere(point3(0,0,-1),0.5, r)) 返回颜色(1,0,0)； |  |
| vec3 unit\_direction = unit\_vector(r.direction()); auto a = 0.5\*(unit\_direction.y() + 1.0);  vec3 unit\_direction=unit\_vector(r.direction())；auto a=0.5\*(unit\_direction.y()+1.0)；  return (1.0-a)\*color(1.0, 1.0, 1.0) + a\*color(0.5, 0.7, 1.0);  }  返回(1.0-a)\*颜色(1.0, 1.0， 1.0)+A\*颜色(0.5,0.7, 1.0)； } |  |

|  |  |  |
| --- | --- | --- |
|  | [main.cc]  [main.cc] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 24/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 24/120

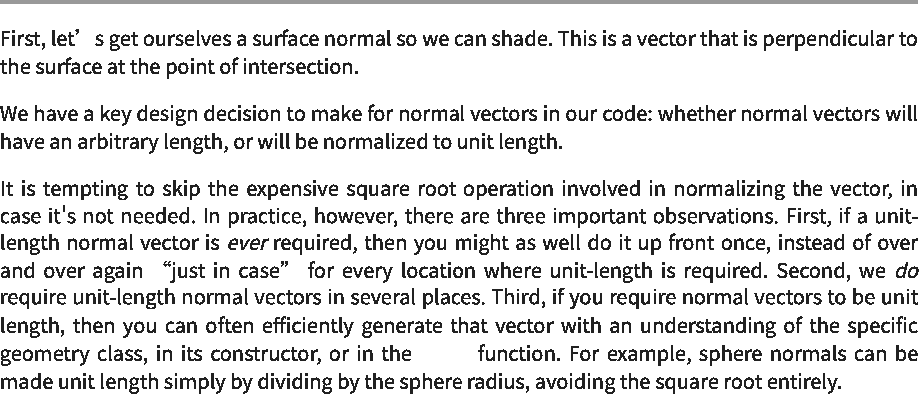
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**6. Surface Normals and Multiple Objects**  
 6.曲面法线和多个物体

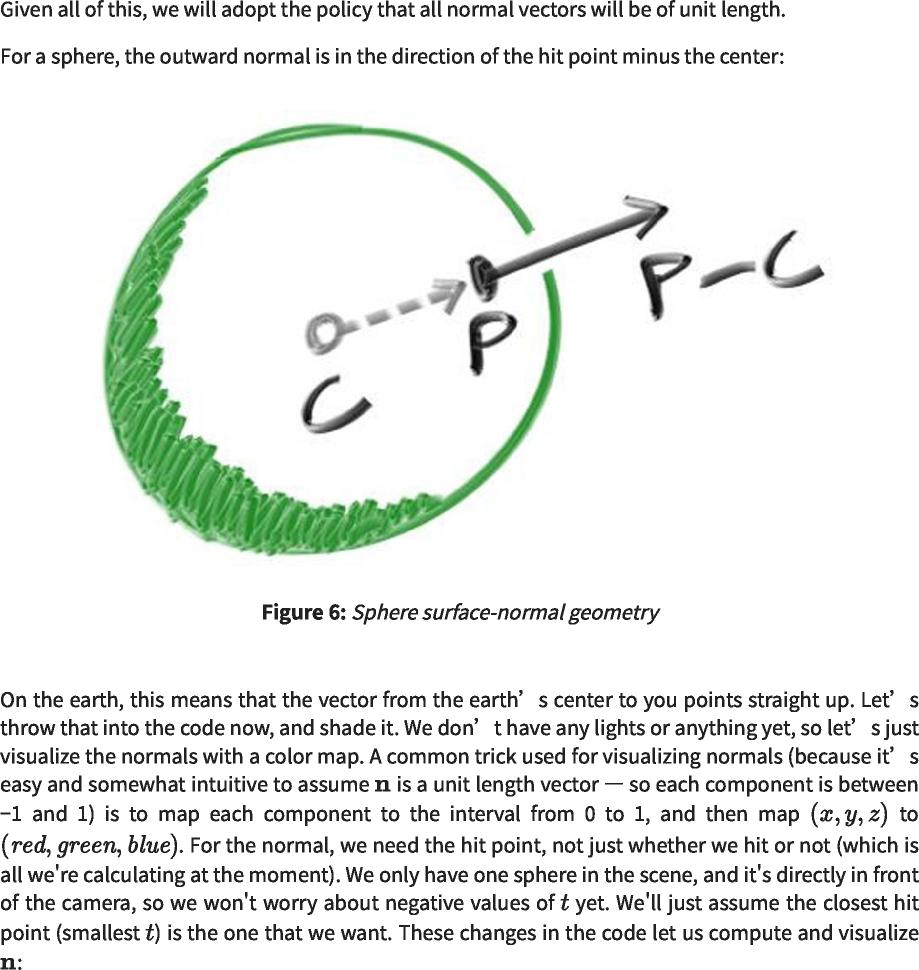
**6.1. Shading with Surface Normals**  
 6.1.使用曲面法线着色

https://raytracing.github.io/books/RayTracingInOneWeekend.html 25/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 25/120



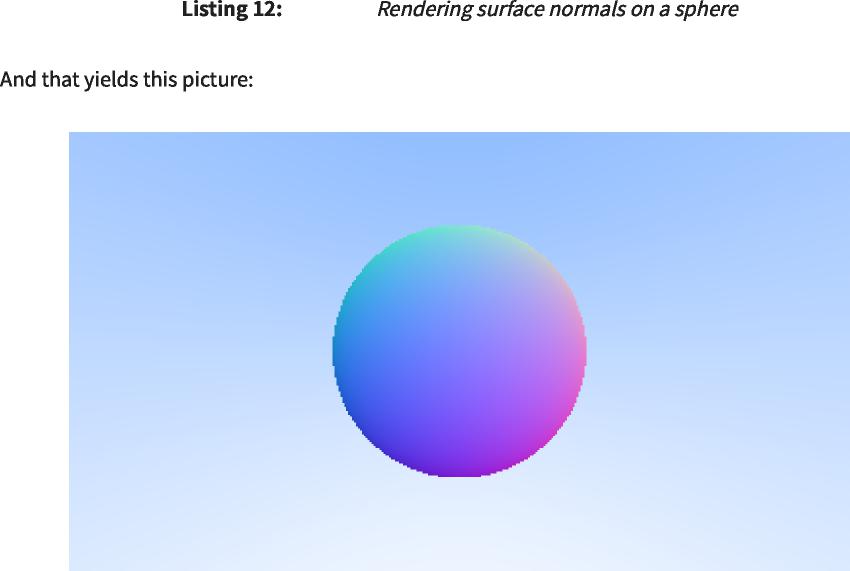
hit()  
 点击()

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



https://raytracing.github.io/books/RayTracingInOneWeekend.html 26/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 26/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



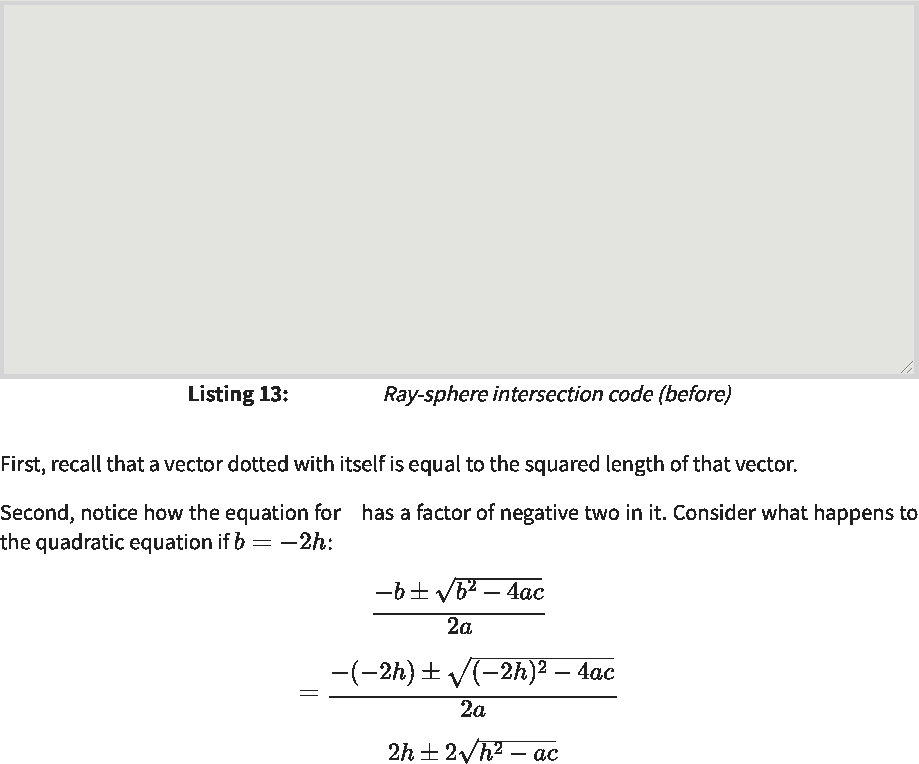
[main.cc]  
 [main.cc]

|  |  |
| --- | --- |
|  |  |
| double **hit\_sphere**(const point3& center, double radius, const ray& r)  双hit\_sphere（常量点3&中心，双半径，常量射线&r） | { |
| vec3 oc = center - r.origin();  vec 3 oc=center-r.origin()；  auto a = dot(r.direction(), r.direction());  auto b = -2.0 \* dot(r.direction(), oc);  auto c = dot(oc, oc) - radius\*radius;  auto discriminant = b\*b - 4\*a\*c;  auto a=dot(r.direction(),r.direction())； auto b=-2.0\*dot(r.direction(),oc)； 自动c=点（oc，oc）-半径\*半径； 自动判别式=b\*b-4\*a\*c； |  |
| if (discriminant < 0) {  如果(判别式<0){  return -1.0;  返回-1.0；  } else {  }else{  return (-b - std::sqrt(discriminant) ) / (2.0\*a);  返回(-b-std：：sqrt(判别式))/(2.0\*a)；  } |  |
| }  color **ray\_color**(const ray& r) {  color ray\_color(const ray&r){ |  |
| auto t = hit\_sphere(point3(0,0,-1), 0.5, r);  auto t=hit\_sphere(point3(0,0,-1),0.5,r)；  if (t > 0.0) {  如果(t>0.0){  vec3 N = unit\_vector(r.at(t) - vec3(0,0,-1));  vec3n=unit\_vector(r.at(t)-vec3(0,0,-1))；  return 0.5\*color(N.x()+1, N.y()+1, N.z()+1);  }  返回0.5\*color(N.x()+1,N.y()+1,N.z()+1)； } |  |
| vec3 unit\_direction = unit\_vector(r.direction()); auto a = 0.5\*(unit\_direction.y() + 1.0);  vec3 unit\_direction=unit\_vector(r.direction())；auto a=0.5\*(unit\_direction.y()+1.0)；  return (1.0-a)\*color(1.0, 1.0, 1.0) + a\*color(0.5, 0.7, 1.0);  }  返回(1.0-a)\*颜色(1.0,1.0,1.0)+A\*颜色(0.5,0.7,1.0)； } |  |



https://raytracing.github.io/books/RayTracingInOneWeekend.html 27/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 27/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



[main.cc]  
 [main.cc]

b  
 b

double **hit\_sphere**(const point3& center, double radius, const ray& r) {  
 double hit\_sphere(const point3&center，double radius，const ray&r){

vec3 oc = center - r.origin();  
 vec 3 oc=center-r.origin()；

auto a = dot(r.direction(), r.direction());  
 auto a=dot(r.direction(),r.direction())；

auto b = -2.0 \* dot(r.direction(), oc);  
 auto b=-2.0\*dot(r.direction(),oc)；

auto c = dot(oc, oc) - radius\*radius;  
 自动c=点（oc，oc）-半径\*半径；

auto discriminant = b\*b - 4\*a\*c;  
 自动判别式=b\*b-4\*a\*c；

if (discriminant < 0) {  
 如果(判别式<0){

return -1.0;  
 返回-1.0；

} else {  
 }else{

return (-b - std::sqrt(discriminant) ) / (2.0\*a);  
 返回(-b-std：：sqrt(判别式))/(2.0\*a)；

}

}

**6.2. Simplifying the Ray-Sphere Intersection Code**  
 6.2.射线球相交代码的简化





https://raytracing.github.io/books/RayTracingInOneWeekend.html 28/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 28/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



double **hit\_sphere**(const point3& center, double radius, const ray& r) { vec3 oc = center - r.origin();  
 double hit\_sphere(const point3&center，double radius，const ray&r){vec3 oc=center-r.origin()；

auto a = r.direction().length\_squared();   
auto h = dot(r.direction(), oc);   
auto c = oc.length\_squared() - radius\*radius;   
auto discriminant = h\*h - a\*c;  
 auto a=r.direction().length\_squared()； auto h=dot（r.direction（），oc）； auto c=oc.length\_squared()-radius\*radius； 自动判别式=h\*H-A\*c；

if (discriminant < 0) {  
 如果(判别式<0){

return -1.0;  
 返回-1.0；

} else {  
 }else{

return (h - std::sqrt(discriminant)) / a;  
 返回(h-std：：sqrt(判别式))/a；

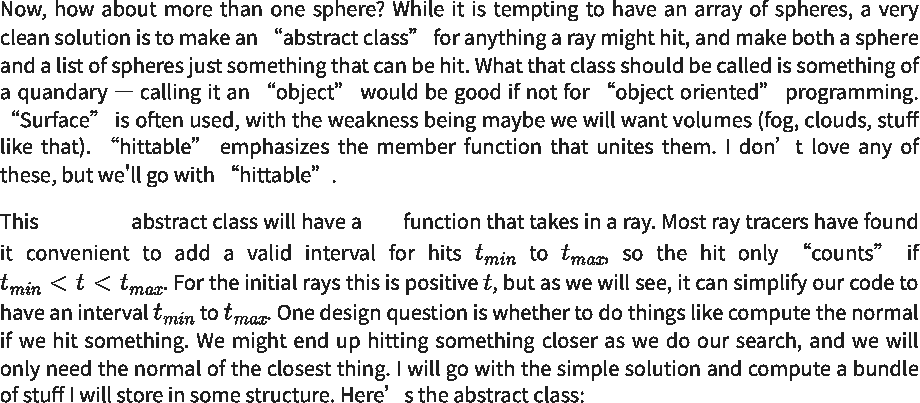
}

}



|  |  |  |
| --- | --- | --- |
|  | [main.cc]  [main.cc] |  |

**6.3. An Abstraction for Hittable Objects**  
 6.3.可命中对象的一种抽象



hittable hit  
 命中

https://raytracing.github.io/books/RayTracingInOneWeekend.html 29/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 29/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**#ifndef HITTABLE\_H   
#define HITTABLE\_H**  
 #ifndef HITTABLE\_H #define HITTABLE\_H

**#include 'ray.h'**  
 #包括“ray.h”

class hit\_record {  
 class hit\_record{

public:  
 公众：

point3 p;  
 点3 p；

vec3 normal;  
 vec3正常；

double t;  
 双t；

};

class hittable {  
 类hittable{

public:  
 公众：

virtual ~hittable() = default;  
 virtual~hittable()=default；

virtual bool **hit**(const ray& r, double ray\_tmin, double ray\_tmax, hit\_record& rec)  
 虚拟布尔命中（const ray&r、double ray\_tmin、double ray\_tmax、hit\_record&rec）

const = **0**;  
 常量=0；

};

**#endif**  
 #endif



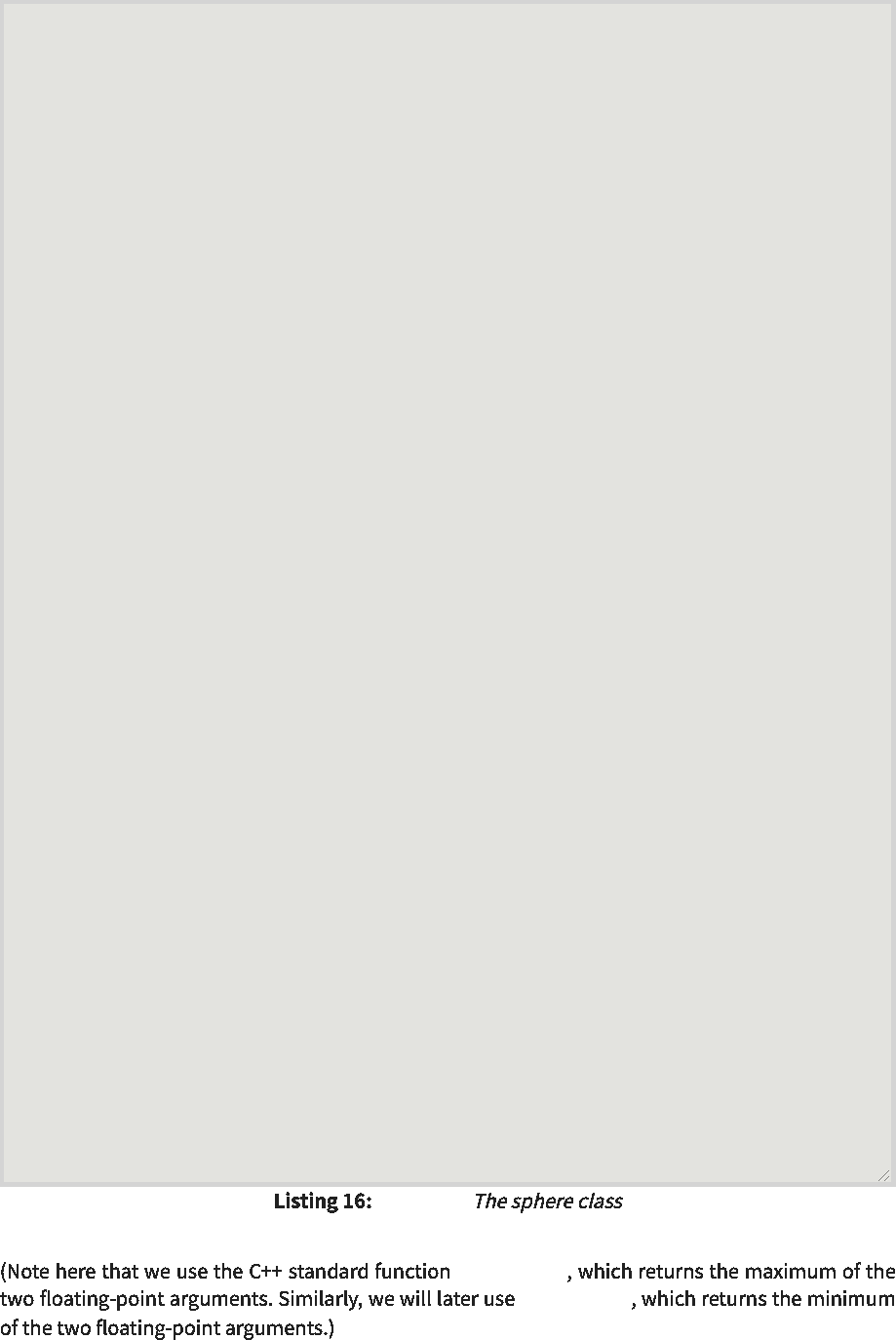
|  |  |  |
| --- | --- | --- |
|  | [hittable.h]  [hittable.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 30/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 30/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



https://raytracing.github.io/books/RayTracingInOneWeekend.html 31/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 31/120



[sphere.h]  
 [球体.h]

std::fmax()  
 std：：fmax()

std::fmin()  
 std：：fmin()

#ifndef SPHERE\_H #define SPHERE\_H  
 #ifndef SPHERE\_H#define SPHERE\_H

#include 'hittable.h' #include 'vec3.h'  
 #包括“hittable.h”#包括“vec3.h”

class sphere : public hittable {  
 类范围：public hittable{

public:  
 公众：

sphere(const point3& center, double radius) : center(center),  
 球体（常量点3&中心，双半径）：中心（中心），

radius(std::fmax(0,radius)) {}  
 radius(std：：fmax(0,radius)){}

bool **hit**(const ray& r, double ray\_tmin, double ray\_tmax, hit\_record& rec) const  
 布尔命中（const ray&r、double ray\_tmin、double ray\_tmax、hit\_record&rec）const

override {  
 覆盖{

vec3 oc = center - r.origin();  
 vec 3 oc=center-r.origin()；

auto a = r.direction().length\_squared();  
 auto a=r.direction().length\_squared()；

auto h = dot(r.direction(), oc);  
 auto h=dot（r.direction（），oc）；

auto c = oc.length\_squared() - radius\*radius;  
 auto c=oc.length\_squared()-radius\*radius；

auto discriminant = h\*h - a\*c;  
 自动判别式=h\*H-A\*c；

if (discriminant < 0)  
 如果（判别式<0）

return false;  
 返回false；

auto sqrtd = std::sqrt(discriminant);  
 auto sqrtd=std：：sqrt(判别式)；

// Find the nearest root that lies in the acceptable range.  
 //找到位于可接受范围内的最近根。

auto root = (h - sqrtd) / a;  
 auto root=(h-sqrtd)/a；

if (root <= ray\_tmin || ray\_tmax <= root) {  
 如果(root<=ray\_tmin ray\_tmax<=root){

root = (h + sqrtd) / a;  
 根=(h+sqrtd)/A；

if (root <= ray\_tmin || ray\_tmax <= root)  
 如果(root<=ray\_tmin ray\_tmax<=root)

return false;  
 返回false；

}

rec.t = root;  
 rec.t=根；

rec.p = r.at(rec.t);  
 rec.p=r.at(rec.t)；

rec.normal = (rec.p - center) / radius;  
 rec.normal=(rec.p-中心)/半径；

return true;  
 返回true；

}

private:  
 私人：

point3 center;  
 点3中心；

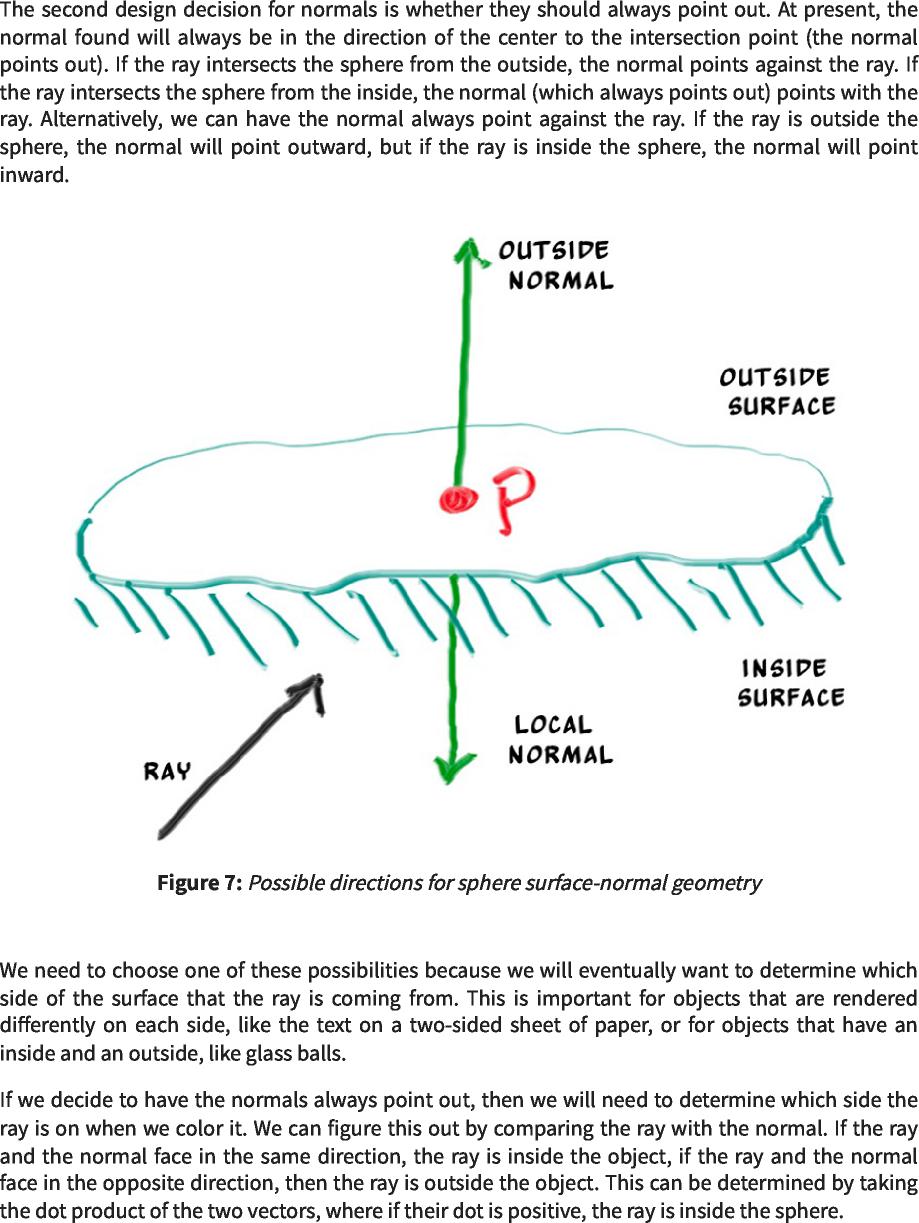
double radius;  
 双半径；

};

#endif  
 #endif

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**6.4. Front Faces Versus Back Faces**  
 6.4.前表面与后表面



https://raytracing.github.io/books/RayTracingInOneWeekend.html 32/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 32/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

if (dot(ray\_direction, outward\_normal) > 0.0) {  
 如果(dot(ray\_direction,outward\_normal)>0.0){

// ray is inside the sphere  
 //射线在球体内部

...

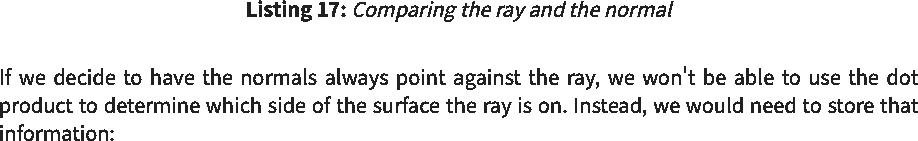
} else {  
 }else{

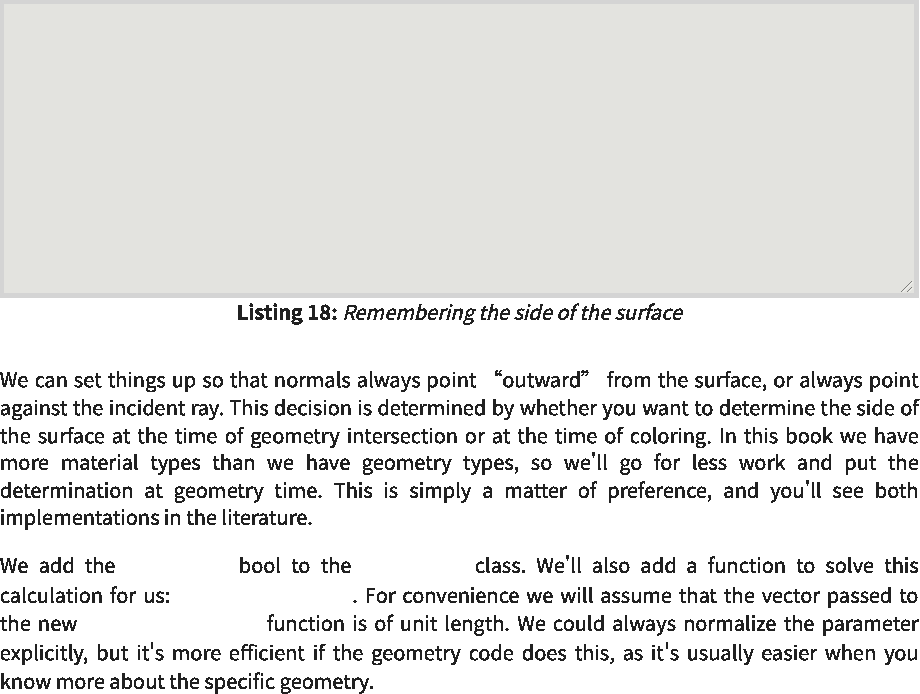
// ray is outside the sphere  
 //射线在球体之外

...

}







front\_face hit\_record  
 front\_facehit\_record

set\_face\_normal() set\_face\_normal()  
 set\_face\_normal()set\_face\_normal()

bool front\_face;  
 bool front\_face；

if (dot(ray\_direction, outward\_normal) > 0.0) {  
 如果(dot(ray\_direction,outward\_normal)>0.0){

// ray is inside the sphere  
 //射线在球体内部

normal = -outward\_normal;  
 normal=-outward\_normal；

front\_face = false;  
 front\_face=false；

} else {  
 }else{

// ray is outside the sphere  
 //射线在球体之外

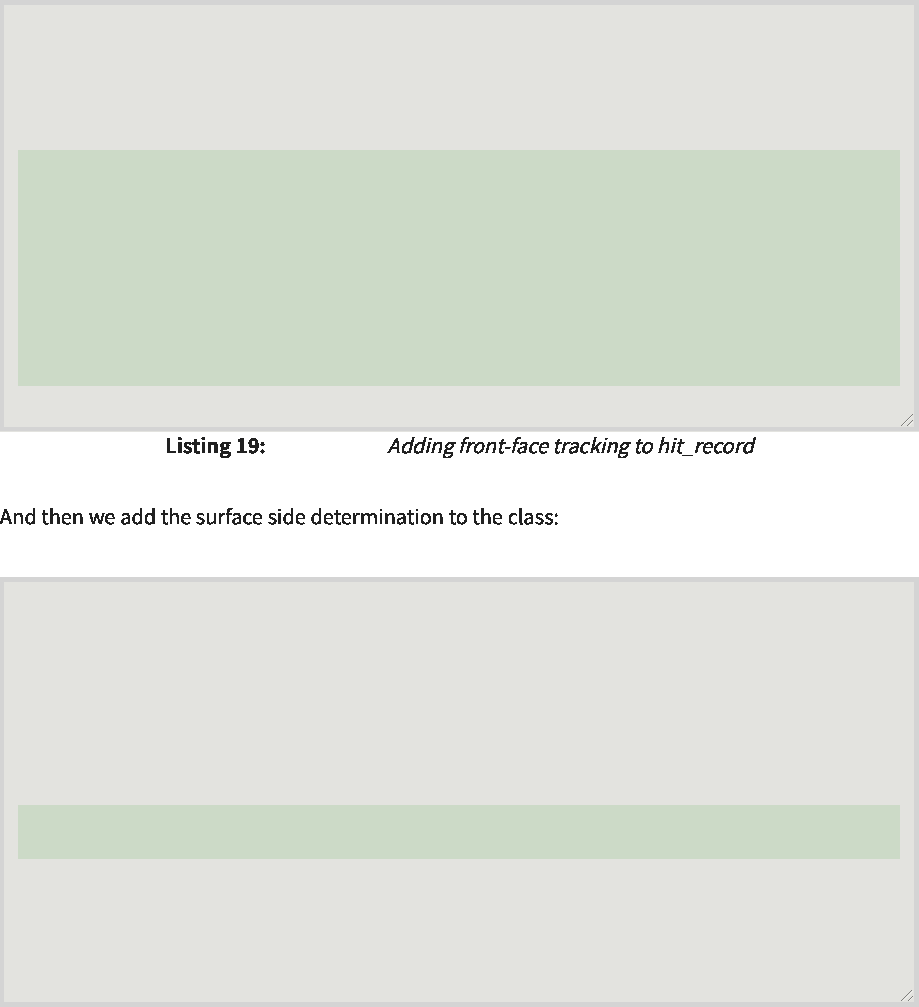
normal = outward\_normal;  
 正常=向外\_正常；

front\_face = true;  
 front\_face=true；

}

https://raytracing.github.io/books/RayTracingInOneWeekend.html 33/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 33/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



[hittable.h]  
 [hittable.h]

class hit\_record {  
 class hit\_record{

public:  
 公众：

point3 p;  
 点3 p；

vec3 normal;  
 vec3正常；

double t;  
 双t；

bool front\_face;  
 bool front\_face；

void set\_face\_normal(const ray& r, const vec3& outward\_normal) {  
 void set\_face\_normal(const ray&r，const vec3&outward\_normal){

// Sets the hit record normal vector.  
 //设置命中记录法向量。

// NOTE: the parameter `outward\_normal` is assumed to have unit length.  
 //注意：假定参数`outward\_normal`具有单位长度。

front\_face = dot(r.direction(), outward\_normal) < 0; normal = front\_face ? outward\_normal : -outward\_normal;  
 front\_face=dot(r.direction(),outward\_normal)<0；normal=front\_face？outward\_normal：-outward\_normal；

}

};

class sphere : public hittable {  
 类范围：public hittable{

public:  
 公众：

...

bool hit(const ray& r, double ray\_tmin, double ray\_tmax, hit\_record& rec) const {  
 bool hit(const ray&r、double ray\_tmin、double ray\_tmax、hit\_record&rec)const{

...

rec.t = root;  
 rec.t=根；

rec.p = r.at(rec.t);  
 rec.p=r.at(rec.t)；

vec3 outward\_normal = (rec.p - center) / radius;   
rec.set\_face\_normal(r, outward\_normal);  
 vec3 outward\_normal=(rec.p-中心)/半径； rec.set\_face\_normal(r，outward\_normal)；

return true;  
 返回true；

}

...

};

|  |  |  |
| --- | --- | --- |
|  | [sphere.h]  [球体.h] |  |

**6.5. A List of Hittable Objects**  
 6.5.可击中对象列表



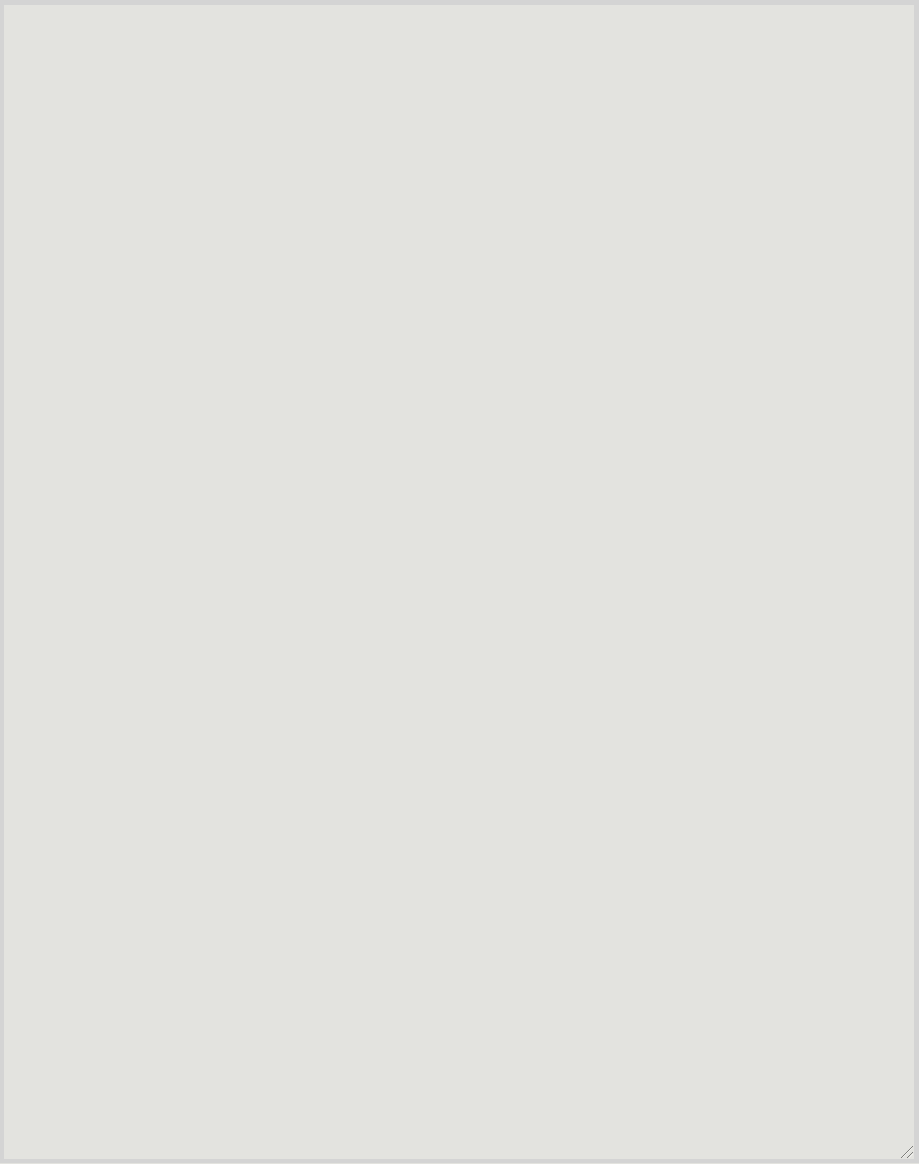
hittable  
 hittable



hittable  
 hittable

https://raytracing.github.io/books/RayTracingInOneWeekend.html 34/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 34/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



#ifndef HITTABLE\_LIST\_H #define HITTABLE\_LIST\_H  
 #ifndef HITTABLE\_LIST\_H#define HITTABLE\_LIST\_H

#include **"hittable.h"**  
 #包括“hittable.h”

#include **<memory>** #include **<vector>**  
 #include<memory>#include<vector>

**using** std::make\_shared; **using** std::shared\_ptr;  
 使用std：：make\_shared；使用std：：shared\_ptr；

**class** hittable\_list : **public** hittable {  
 class hittable\_list：public hittable{

**public**:  
 公众：

std::vector<shared\_ptr<hittable>> objects;  
 std：：vector<shared\_ptr<hittable>>对象；

hittable\_list() {}  
 hittable\_list(){}

hittable\_list(shared\_ptr<hittable> object) { add(object); }  
 hittable\_list(shared\_ptr<hittable>object){add(object);}

void clear() { objects.clear(); }  
 void clear(){objects.clear();}

void add(shared\_ptr<hittable> object) {   
objects.push\_back(object);  
 void add(shared\_ptr<hittable>对象){ objects.push\_back(对象)；

}

bool hit(const ray& r, double ray\_tmin, double ray\_tmax, hit\_record& rec) const  
 布尔命中（const ray&r、double ray\_tmin、double ray\_tmax、hit\_record&rec）const

**override** {  
 覆盖{

hit\_record temp\_rec;  
 hit\_record temp\_rec；

bool hit\_anything = false;  
 bool hit\_anything=false；

**auto** closest\_so\_far = ray\_tmax;  
 auto closest\_so\_far=ray\_tmax；

**for** (const **auto**& object : objects) {  
 for(const auto&object：objects){

**if** (object->hit(r, ray\_tmin, closest\_so\_far, temp\_rec)) {  
 if(object->hit(r，ray\_tmin，closest\_so\_far，temp\_rec)){

hit\_anything = true;  
 hit\_anything=true；

closest\_so\_far = temp\_rec.t;  
 closest\_so\_far=temp\_rec.t；

rec = temp\_rec;  
 rec=temp\_rec；

}

}

**return** hit\_anything;  
 返回hit\_anything；

}

};

#endif  
 #endif

|  |  |  |
| --- | --- | --- |
|  | [hittable\_list.h]  [hittable\_list.h] |  |

**6.6. Some New C++ Features**  
 6.6.一些新的C++特性

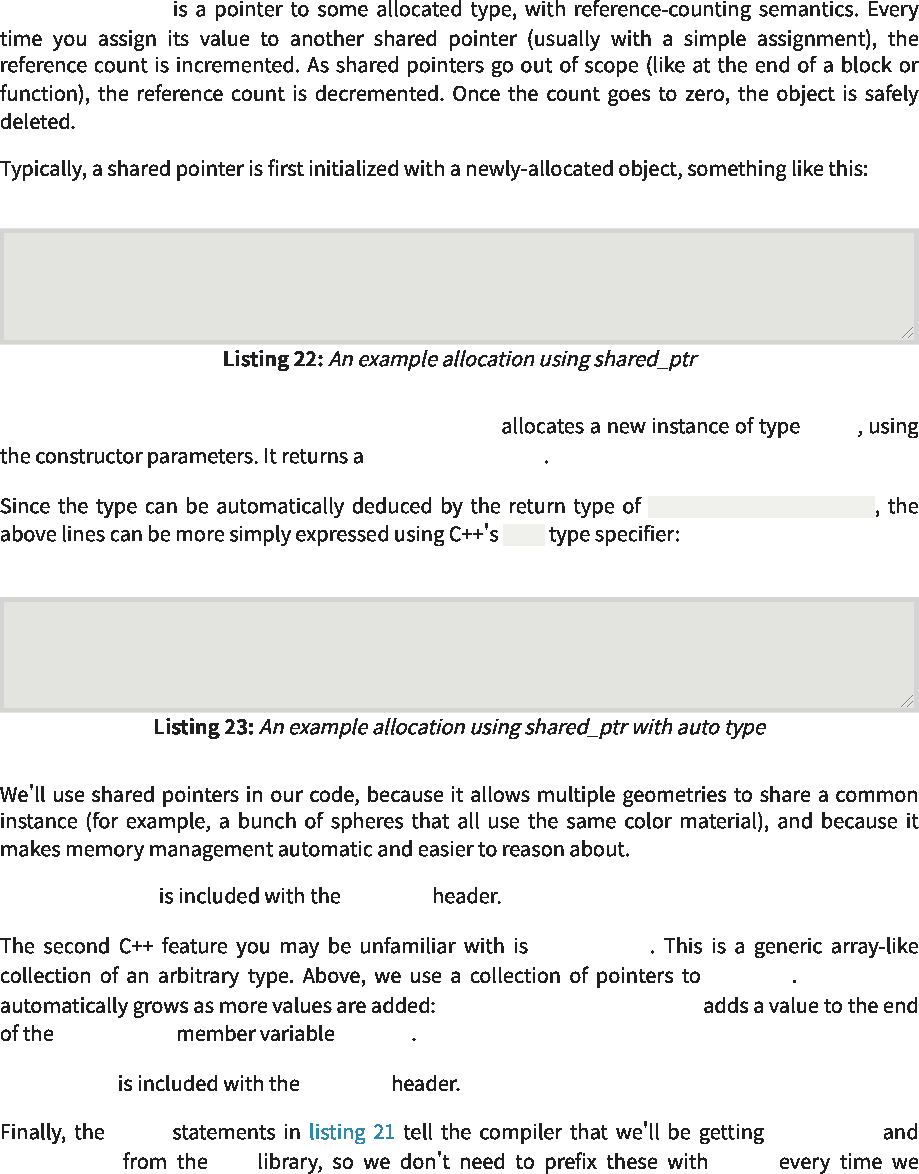


**hittable\_list**  
 hittable\_list

**vector shared\_ptr make\_shared**  
 向量shared\_ptrmake\_shared

https://raytracing.github.io/books/RayTracingInOneWeekend.html 35/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 35/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



shared\_ptr<type>  
 shared\_ptr<type>

shared\_ptr<double> double\_ptr = make\_shared<double>(0.37);  
 shared\_ptr<double>double\_ptr=make\_shared<double>(0.37)；

shared\_ptr<vec3> vec3\_ptr = make\_shared<vec3>(1.414214, 2.718281, 1.618034);   
shared\_ptr<sphere> sphere\_ptr = make\_shared<sphere>(point3(0,0,0), 1.0);  
 shared\_ptr<vec3>vec3\_ptr=make\_shared<vec3>(1.414214,2.718281,1.618034)； shared\_ptr<sphere>sphere\_ptr=make\_shared<sphere>(point3(0,0,0),1.0)；

shared\_ptr<thing>  
 shared\_ptr<thing>

make\_shared<type>(...)  
 make\_shared<type>(...)

auto  
 汽车

make\_shared<thing>(thing\_constructor\_params ...) thing  
 make\_shared<thing>(thing\_constructor\_params...)thing

std::vector  
 std：：vector

using shared\_ptr  
 使用shared\_ptr

make\_shared std std::  
 make\_sharedstdstd：：

std::shared\_ptr <memory>  
 std：：shared\_ptr<内存>

std::vector <vector>  
 std：：vector<vector>

std::vector objects  
 std：：vectorobjects

objects.push\_back(object)  
 objects.push\_back(对象)

hittable std::vector  
 hittable std：：vector

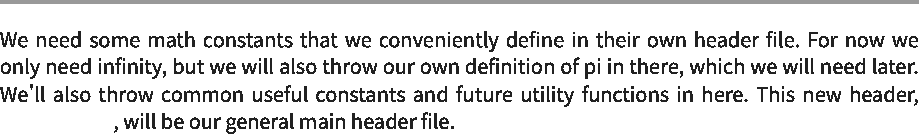
auto double\_ptr = make\_shared<double>(0.37);  
 auto double\_ptr=make\_shared<double>(0.37)；

auto vec3\_ptr = make\_shared<vec3>(1.414214, 2.718281, 1.618034);   
auto sphere\_ptr = make\_shared<sphere>(point3(0,0,0), 1.0);  
 auto vec3\_ptr=make\_shared<vec3>(1.414214,2.718281,1.618034)； auto sphere\_ptr=make\_shared<sphere>(point3(0,0,0),1.0)；



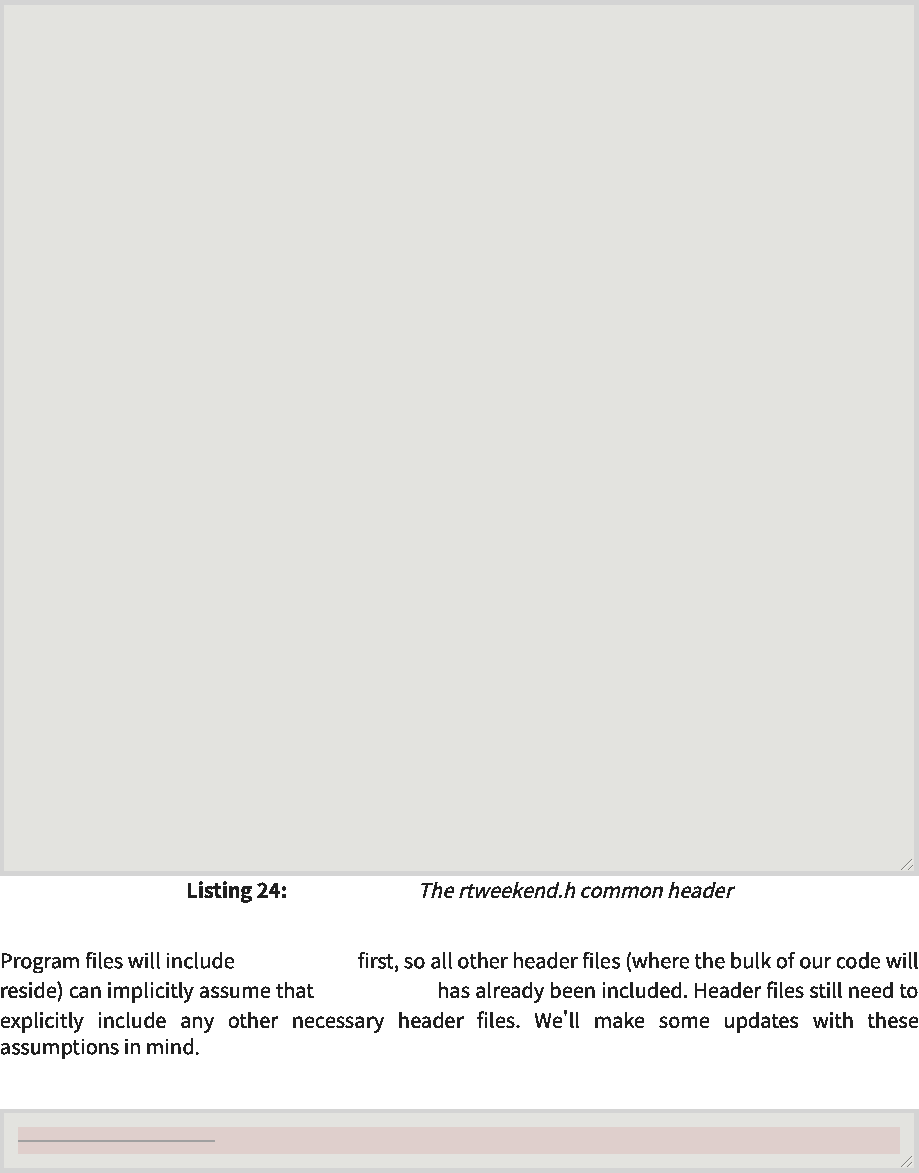
**6.7. Common Constants and Utility Functions**  
 6.7.常用常量和效用函数

https://raytracing.github.io/books/RayTracingInOneWeekend.html 36/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 36/120



rtweekend.h  
 rtweekend.h

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



[rtweekend.h]  
 [rtweekend.h]

rtweekend.h  
 rtweekend.h

rtweekend.h  
 rtweekend.h

#ifndef RTWEEKEND\_H #define RTWEEKEND\_H  
 #ifndef RTWEEKEND\_H#define RTWEEKEND\_H

#include <cmath> #include <iostream> #include <limits> #include <memory>  
 #include<cmath>#include<iostream>#include<limits>#include<memory>

// C-i--i- Std Usings  
 //C-I-I-STD使用

using std::make\_shared; using std::shared\_ptr;  
 使用std：：make\_shared；使用std：：shared\_ptr；

// Constants  
 //常量

const double infinity = std::numeric\_limits<double>::infinity(); const double pi = **3.1415926535897932385**;  
 const double infinity=std：：numeric\_limits<double>：：infinity()；常量双π=3.1415926535897932385；

// Utility Functions  
 //实用函数

inline double **degrees\_to\_radians**(double degrees) { return degrees \* pi / **180.0**;  
 inline double degrees\_to\_radians(double degrees){return degrees\*pi/180.0；

}

// Common Headers  
 //公共标头

#include 'color.h' #include 'ray.h' #include 'vec3.h'  
 #包括“color.h”#包括“ray.h”#包括“vec3.h”

#endif  
 #endif

#include <iostream>  
 #包括 <iostream>

|  |  |  |
| --- | --- | --- |
|  | [color.h]  [颜色.h] |  |

#include 'ray.h'  
 #包括 “雷.h”



|  |  |  |
| --- | --- | --- |
|  | [hittable.h]  [hittable.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 37/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 37/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |
| --- |
|  |
| **#include**   #包括 |
| **<memory>**  <存储器> |
| **#include <vector>**  #include<vector> |
|  |
| **using std::make\_shared;**  使用std：：make\_shared； |
| **using std::shared\_ptr;**  使用std：：shared\_ptr； |
|  |

|  |  |  |
| --- | --- | --- |
|  | [hittable\_list.h]  [hittable\_list.h] |  |

**#include 'vec3.h'**  
 #包括 'vec3.h'



|  |  |  |
| --- | --- | --- |
|  | [sphere.h]  [球体.h] |  |

**#include <cmath>**  
 #包括 <cmath>

**#include <iostream>**  
 #包括 <iostream>



|  |  |  |
| --- | --- | --- |
|  | [vec3.h]  [vec3.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 38/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 38/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



https://raytracing.github.io/books/RayTracingInOneWeekend.html 39/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 39/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |
| --- |
|  |
| #include 'rtweekend.h'  #包括“rtweekend.h” |
| #include 'color.h'  #包括“color.h” |
|  |
| #include 'ray.h'  #include 'vec3.h'  #包括“ray.h” #包括“vec3.h” |
|  |
| #include 'hittable.h'  #包括“hittable.h”  #include 'hittable\_list.h'  #include 'sphere.h'  #include'hittable\_list.h' #包括“球体.h” |
|  |
| #include  #包括  <iostream>  <iostream> |
|  |
|  |
|  |
| double hit\_sphere(const double  double hit\_sphere(constdouble  point3& center, radius, const ray& r) {  点3&中心，半径，常量射线&r){  ... |
| } |
|  |
| color ray\_color(const ray& r, const hittable& world) {  color ray\_color(const ray&r，const hittable&world){  hit\_record rec;  hit\_record rec；  if (world.hit(r, 0, infinity, rec)) {  if(world.hit(r,0,infinity,rec)){  return 0.5 \* (rec.normal + color(1,1,1));  返回0.5\*(rec.normal+color(1,1,1))；  } |
| vec3 unit\_direction = unit\_vector(r.direction());  auto a = 0.5\*(unit\_direction.y() + 1.0);  vec3 unit\_direction=unit\_vector(r.direction())； auto a=0.5\*(unit\_direction.y()+1.0)；  return (1.0-a)\*color(1.0, 1.0, 1.0) + a\*color(0.5, 0.7, 1.0);  }  返回(1.0-a)\*颜色(1.0, 1.0，1.0)+A\*颜色(0.5，0.7，1.0)； }  int main() {  // Image  int main(){ //图像  auto aspect\_ratio = 16.0 / 9.0;  int image\_width = 400;  auto aspect\_ratio=16.0/9.0； int image\_width=400；  // Calculate the image height, and ensure that it's at least 1. int image\_height = int(image\_width / aspect\_ratio); image\_height = (image\_height < 1) ? 1 : image\_height;  //计算图像高度，并确保它至少为1。int image\_height=int(image\_width/aspect\_ratio)；image\_height=(image\_height<1)？1：image\_height； |
| // World  //世界  hittable\_list world;  hittable\_list world；  world.add(make\_shared<sphere>(point3(0,0,-1), 0.5));  world.add(make\_shared<sphere>(point3(0,-100.5,-1), 100));  world.add(make\_shared<sphere>(point3(0,0,-1),0.5))； world.add(make\_shared<sphere>(point3(0,-100.5,-1),100))； |
| // Camera  //相机  auto focal\_length = 1.0;  auto focal\_length=1.0；  auto viewport\_height = 2.0;  auto viewport\_height=2.0；  auto viewport\_width = viewport\_height \* (double(image\_width)/image\_height);  auto viewport\_width=viewport\_height\*(double(image\_width)/image\_height)；  auto camera\_center = point3(0, 0, 0);  auto camera\_center=point3(0,0,0)；  // Calculate the vectors across the horizontal and down the vertical viewport  //计算水平视口和垂直视口向下的向量  edges.  边缘。  auto viewport\_u = vec3(viewport\_width, 0, 0);  auto viewport\_u=vec3(viewport\_width,0,0)；  auto viewport\_v = vec3(0, -viewport\_height, 0);  auto viewport\_v=vec3(0,-viewport\_height,0)； |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 40/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 40/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

// Calculate the horizontal and vertical delta vectors from pixel to pixel.  
 //计算从像素到像素的水平和垂直增量向量。

auto pixel\_delta\_u = viewport\_u / image\_width;  
 auto pixel\_delta\_u=viewport\_u/image\_width；

auto pixel\_delta\_v = viewport\_v / image\_height;  
 auto pixel\_delta\_v=viewport\_v/image\_height；

// Calculate the location of the upper left pixel.  
 //计算左上像素的位置。

auto viewport\_upper\_left = camera\_center  
 auto viewport\_upper\_left=camera\_center

- vec3(0, 0, focal\_length) - viewport\_u/2 -  
 -vec3(0,0,focal\_length)-viewport\_u/2-

viewport\_v/2;  
 viewport\_v/2；

auto pixel00\_loc = viewport\_upper\_left + 0.5 \* (pixel\_delta\_u + pixel\_delta\_v);  
 auto pixel00\_loc=viewport\_upper\_left+0.5\*(pixel\_delta\_u+pixel\_delta\_v)；

// Render  
 //渲染

std::cout << 'P3\n' << image\_width << • • << image\_height << '\n255\n';  
 std：：cout<<'P3\n'<<image\_width<<••<<image\_height<<'\n255\n';

for (int j = 0; j < image\_height; j++) {  
 for(int j=0;j<image\_height;j++){

std::clog << '\rScanlines remaining: ' << (image\_height - j) << • • <<  
 std：：clog<<'\rScanlines remaining：'<<(image\_height-j)<<••<<

std::flush;  
 std：：flush；

for (int i = 0; i < image\_width; i++) {  
 for(int i=0;i<image\_width;i++){

auto pixel\_center = pixel00\_loc + (i \* pixel\_delta\_u) + (j \*  
 auto pixel\_center=pixel00\_loc+(i\*pixel\_delta\_u)+(j\*

pixel\_delta\_v);  
 pixel\_delta\_v)；

auto ray\_direction = pixel\_center - camera\_center;  
 auto ray\_direction=pixel\_center-camera\_center；

ray r(camera\_center, ray\_direction);  
 光线r(相机中心，光线方向)；

color pixel\_color = ray\_color(r, world);  
 color pixel\_color=ray\_color(r，world)；

write\_color(std::cout, pixel\_color);  
 write\_color(std：：cout，pixel\_color)；

}

}

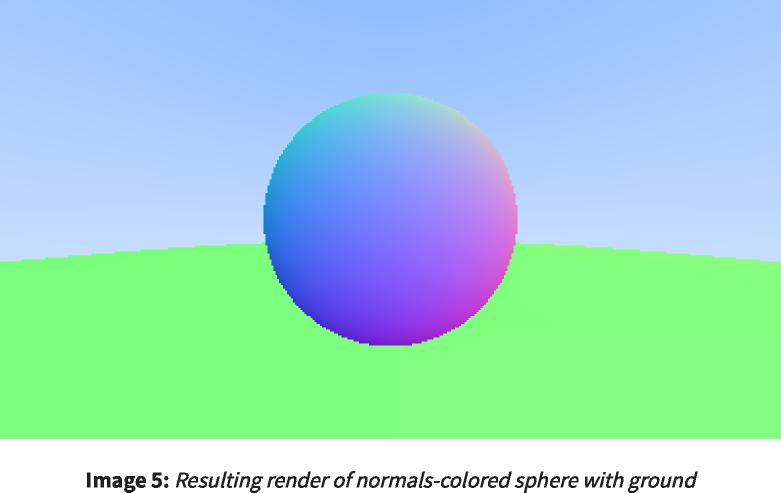
std::clog << '\rDone. \n';   
}  
 std：：clog<<'\r完成。\n'; }



[main.cc]  
 [main.cc]

https://raytracing.github.io/books/RayTracingInOneWeekend.html 41/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 41/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

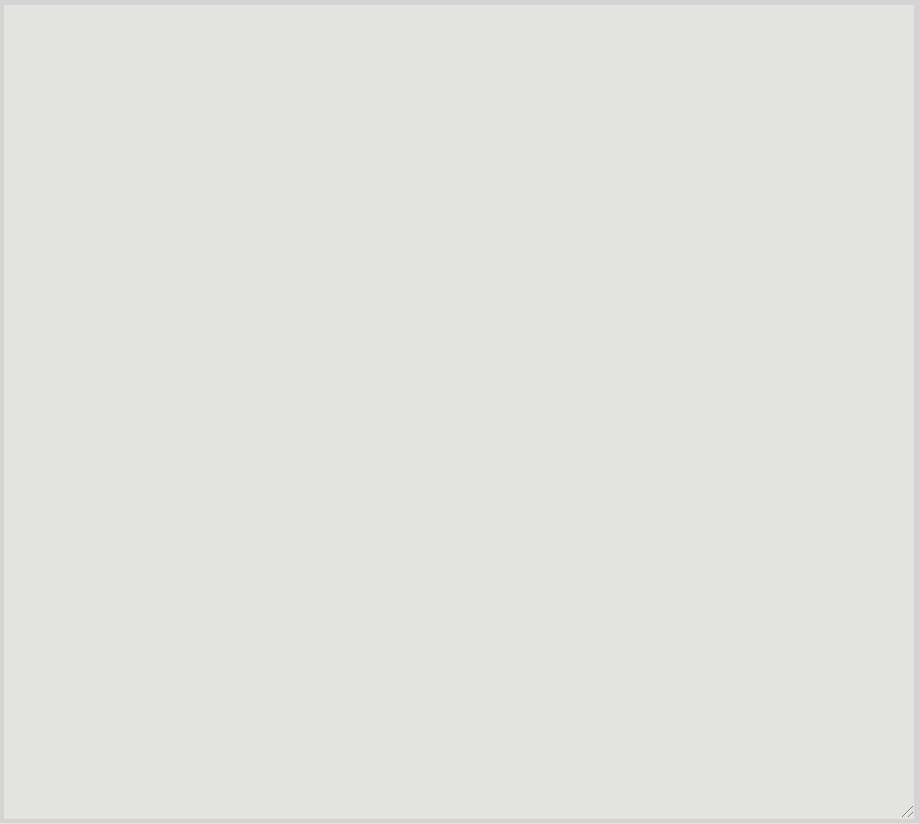


**6.8. An Interval Class**  
 6.8.区间类



https://raytracing.github.io/books/RayTracingInOneWeekend.html 42/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 42/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



#ifndef INTERVAL\_H #define INTERVAL\_H  
 #ifndef INTERVAL\_H#define INTERVAL\_H

class interval {  
 类间隔{

public:  
 公众：

double min, max;  
 双倍最小值，最大值；

interval() : min(+infinity), max(-infinity) {} // Default interval is empty interval(double min, double max) : min(min), max(max) {}  
 区间()：min(+infinity),max(-infinity){}//默认区间为空区间(double min,double max)：min(min),max(max){}

double **size**() const {   
return max - min;  
 double size()const{ 返回最大值-最小值；

}

bool **contains**(double x) const { return min <= x && x <= max;  
 bool contains(double x)const{return min<=x&&x<=max；

}

bool **surrounds**(double x) const { return min < x && x < max;  
 bool环绕（double x）const{return min<x&&x<max；

}

static const interval empty, universe;  
 静态常量区间空，宇宙；

};

const interval interval::empty = interval(+infinity, -infinity);  
 常量区间区间：：empty=区间（+无穷大，-无穷大）；

const interval interval::universe = interval(-infinity, +infinity);  
 常量区间区间：：universe=区间（-无穷大，+无穷大）；

#endif  
 #endif

|  |  |  |
| --- | --- | --- |
|  | [interval.h]  [间隔h] |  |

// Common Headers   
#include 'color.h'  
 //公共标头 #包括“color.h”

#include 'interval.h'  
 #包括“间隔.h”

#include 'ray.h'   
#include 'vec3.h'  
 #包括“ray.h” #包括“vec3.h”



|  |  |  |
| --- | --- | --- |
|  | [rtweekend.h]  [rtweekend.h] |  |

class hittable {  
 类hittable{

public:  
 公众：

...

virtual bool **hit**(const ray& r, interval ray\_t, hit\_record& rec) const = **0**;  
 虚拟布尔命中(const ray&r，interval ray\_t，hit\_record&rec)const=0；

};



|  |  |  |
| --- | --- | --- |
|  | [hittable.h]  [hittable.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 43/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 43/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |  |  |  |
| --- | --- | --- | --- |
| **class hittable\_list : public hittable {**  class hittable\_list：public hittable{  **public:**  公众：  **...** | | |  |
|  | **bool hit(const ray& r, interval ray\_t, hit\_record& rec) const override {**  bool hit(const ray&r，interval ray\_t，hit\_record&rec)const override{ | |  |
|  |  | **hit\_record temp\_rec;**  hit\_record temp\_rec；  **bool hit\_anything = false;**  bool hit\_anything=false； |  |
|  |  | **auto closest\_so\_far = ray\_t.max;**  auto closest\_so\_far=ray\_t.max； |  |
|  |  | **for (const auto& object : objects) {**  for(const auto&object：objects){ |  |
|  |  | **if (object->hit(r, interval(ray\_t.min, closest\_so\_far), temp\_rec))**  if(object->hit(r，interval(ray\_t.min，closest\_so\_far)，temp\_rec)) | **{** |
| **};** | **}**  **...** | **hit\_anything = true;**  hit\_anything=true；  **closest\_so\_far = temp\_rec.t;**  closest\_so\_far=temp\_rec.t；  **rec = temp\_rec;**  rec=temp\_rec；  **}**  **}**  **return hit\_anything;**  返回hit\_anything； |  |

|  |  |  |
| --- | --- | --- |
|  | [hittable\_list.h]  [hittable\_list.h] |  |

|  |  |  |
| --- | --- | --- |
| **class sphere : public hittable {**  类范围：public hittable{  **public:**  公众：  **...** | | |
|  | **bool hit(const ray& r, interval ray\_t, hit\_record& rec) const override {**  bool hit(const ray&r，interval ray\_t，hit\_record&rec)const override{ | |
|  |  | **...**  **// Find the nearest root that lies in the acceptable range.  auto root = (h - sqrtd) / a;**  //找到位于可接受范围内的最近根。 auto root=(h-sqrtd)/a； |
|  |  | **if (!ray\_t.surrounds(root)) {**  if(！ray\_t.surrounds(root)){ |
|  |  | **root = (h + sqrtd) / a;**  根=(h+sqrtd)/A； |
|  |  | **if (!ray\_t.surrounds(root))**  if(！ray\_t.surrounds(root)) |
| **};** | **}**  **...** | **return false;**  返回false；  **}**  **...** |

|  |  |  |
| --- | --- | --- |
|  | [sphere.h]  [球体.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 44/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 44/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

color ray\_color(const ray& r, const hittable& world) {   
hit\_record rec;  
 color ray\_color(const ray&r，const hittable&world){ hit\_record rec；

if (world.hit(r, interval(0, infinity), rec)) {  
 if(world.hit(r，interval(0，infinity)，rec)){

return 0.5 \* (rec.normal + color(1,1,1));   
}  
 返回0.5\*(rec.normal+color(1,1,1))； }

vec3 unit\_direction = unit\_vector(r.direction());  
 vec3 unit\_direction=unit\_vector(r.direction())；

auto a = 0.5\*(unit\_direction.y() + 1.0);  
 auto a=0.5\*(unit\_direction.y()+1.0)；

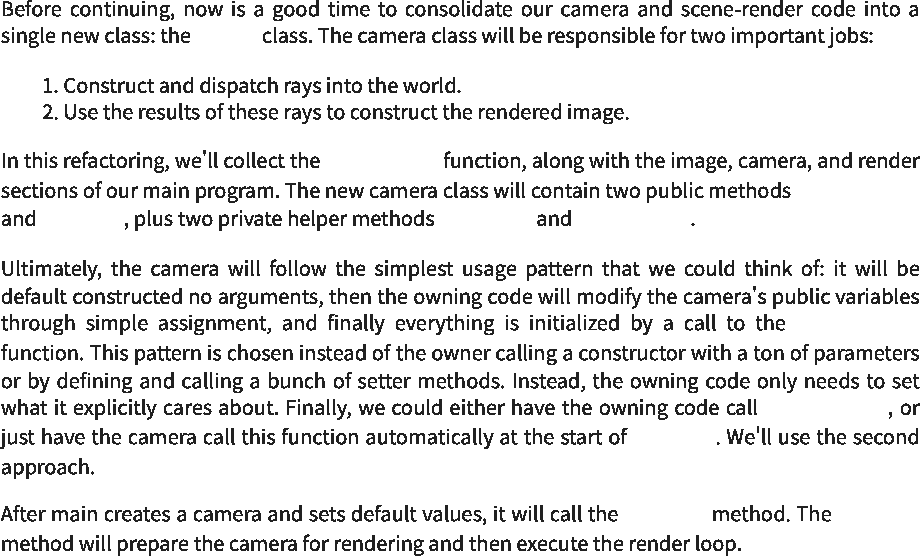
return (1.0-a)\*color(1.0, 1.0, 1.0) + a\*color(0.5, 0.7, 1.0);  
 返回(1.0-a)\*颜色(1.0,1.0,1.0)+A\*颜色(0.5,0.7,1.0)；

}



|  |  |  |
| --- | --- | --- |
|  | [main.cc]  [main.cc] |  |

**7. Moving Camera Code Into Its Own Class**  
 7.将相机代码移到自己的类中



camera  
 照相机

ray\_color()  
 ray\_color()

initialize()  
 initialize()

render() get\_ray() ray\_color()  
 render()get\_ray()ray\_color()

initialize()  
 initialize()

initialize()  
 initialize()

render()  
 渲染()

render() render()  
 渲染()渲染()

https://raytracing.github.io/books/RayTracingInOneWeekend.html 45/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 45/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |  |  |
| --- | --- | --- |
|  | camera  照相机 |  |

#ifndef CAMERA\_H   
#define CAMERA\_H  
 #ifndef CAMERA\_H #define CAMERA\_H

#include "hittable.h"  
 #包括“hittable.h”

class camera {  
 类相机{

public:  
 公众：

/\* Public Camera Parameters Here \*/  
 /\*此处为公共摄像机参数\*/

void render(const hittable& world) {  
 void render(const hittable&world){

...

}

private:  
 私人：

/\* Private Camera Variables Here \*/  
 /\*此处为私有相机变量\*/

void initialize() {  
 void initialize(){

...

}

color ray\_color(const ray& r, const hittable& world) const {  
 color ray\_color(const ray&r，const hittable&world)const{

...

}

};

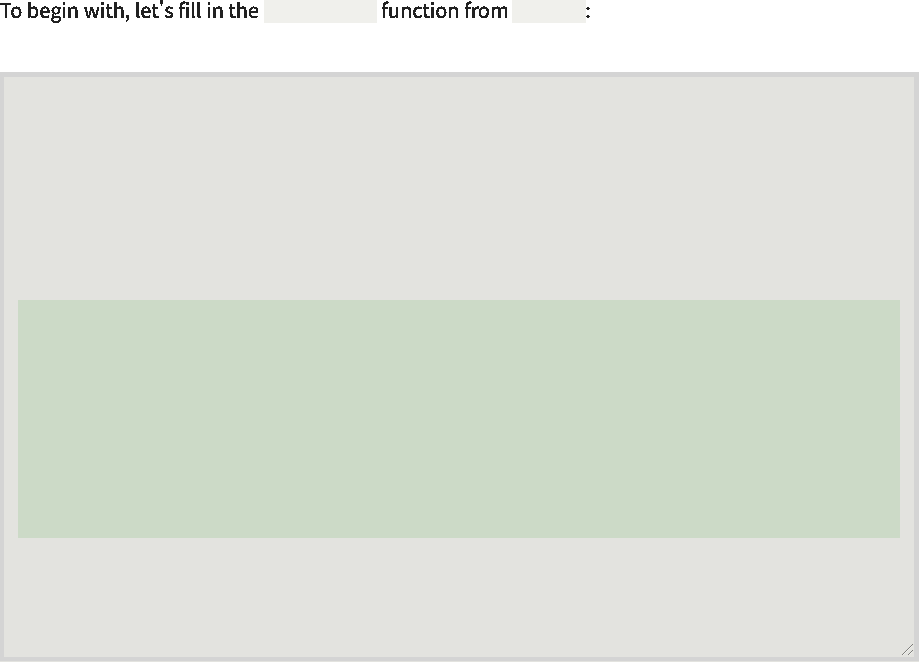
#endif  
 #endif



|  |  |  |
| --- | --- | --- |
|  | [camera.h]  [摄像机.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 46/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 46/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



main.cc   
 main.cc

ray\_color()  
 ray\_color()

class camera {  
 类相机{

...

private:  
 私人：

...

color ray\_color(const ray& r, const hittable& world) const {  
 color ray\_color(const ray&r，const hittable&world)const{

hit\_record rec;  
 hit\_record rec；

if (world.hit(r, interval(**0**, infinity), rec)) { return **0.5** \* (rec.normal + color(**1**,**1**,**1**));  
 if(world.hit(r，interval(0，infinity)，rec)){return 0.5\*(rec.normal+color(1,1,1))；

}

vec3 unit\_direction = unit\_vector(r.direction());  
 vec3 unit\_direction=unit\_vector(r.direction())；

auto a = **0.5**\*(unit\_direction.y() + **1.0**);  
 auto a=0.5\*(unit\_direction.y()+1.0)；

return (**1.0**-a)\*color(**1.0**, **1.0**, **1.0**) + a\*color(**0.5**, **0.7**, **1.0**);  
 返回(1.0-a)\*颜色(1.0,1.0,1.0)+A\*颜色(0.5,0.7,1.0)；

}

};

#endif  
 #endif

|  |  |  |
| --- | --- | --- |
|  | [camera.h]  [摄像机.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 47/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 47/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

main()  
 主()

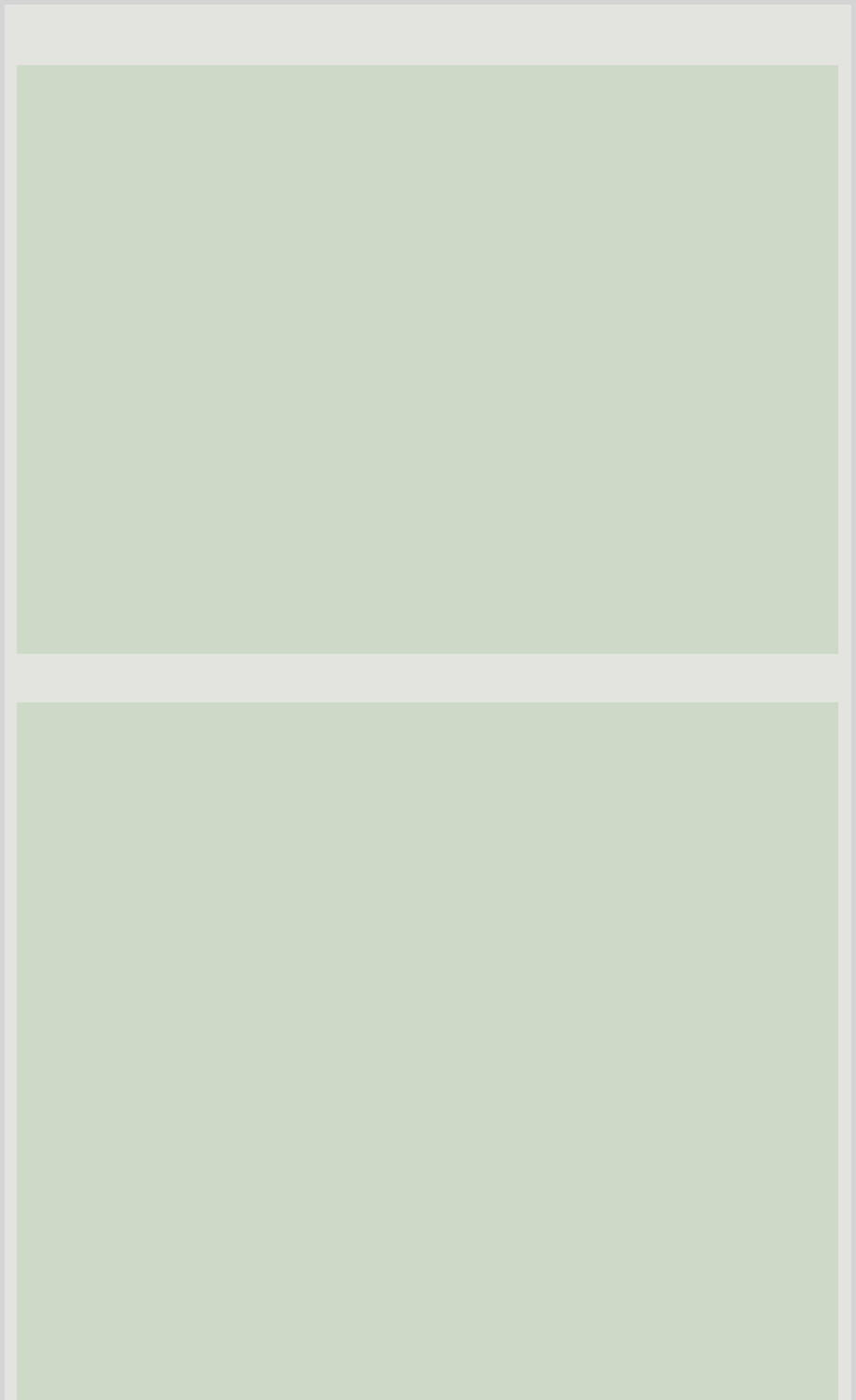
main()  
 主()



https://raytracing.github.io/books/RayTracingInOneWeekend.html 48/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 48/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 49/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 49/120



class camera { public:  
 class camera{public：

double aspect\_ratio = 1.0; // Ratio of image width over height  
 双纵横比=1.0；//图像宽高比

int image\_width = 100; // Rendered image width in pixel count  
 intimage\_width=100；//以像素计数表示的渲染图像宽度

void render(const hittable& world) {   
initialize();  
 void render(const hittable&world){ initialize()；

std::cout << 'P3\n' << image\_width << • • << image\_height << '\n255\n';  
 std：：cout<<'P3\n'<<image\_width<<••<<image\_height<<'\n255\n';

for (int j = 0; j < image\_height; j++) {  
 for(int j=0;j<image\_height;j++){

std::clog << '\rScanlines remaining: ' << (image\_height - j) << • • <<  
 std：：clog<<'\rScanlines remaining：'<<(image\_height-j)<<••<<

std::flush;  
 std：：flush；

for (int i = 0; i < image\_width; i++) {  
 for(int i=0;i<image\_width;i++){

auto pixel\_center = pixel00\_loc + (i \* pixel\_delta\_u) + (j \*  
 auto pixel\_center=pixel00\_loc+(i\*pixel\_delta\_u)+(j\*

pixel\_delta\_v);  
 pixel\_delta\_v)；

auto ray\_direction = pixel\_center - center; ray r(center, ray\_direction);  
 auto ray\_direction=pixel\_center-center；射线r(中心，射线方向)；

color pixel\_color = ray\_color(r, world); write\_color(std::cout, pixel\_color);  
 color pixel\_color=ray\_color(r，world)；write\_color(std：：cout，pixel\_color)；

}

}

std::clog << '\rDone. \n';  
 std：：clog<<'\r完成。\n';

}

private:  
 私人：

int image\_height; // Rendered image height  
 intimage\_height； //渲染图像高度

point3 center; // Camera center  
 点3中心； //相机中心

point3 pixel00\_loc; // Location of pixel 0, 0  
 点3 pixel00\_loc； //像素0,0的位置

vec3 pixel\_delta\_u; // Offset to pixel to the right vec3 pixel\_delta\_v; // Offset to pixel below  
 vec3pixel\_delta\_u；//向右像素的偏移量vec3 pixel\_delta\_v；//偏移量到下面的像素

void initialize() {  
 void initialize(){

image\_height = int(image\_width / aspect\_ratio);  
 image\_height=int(image\_width/aspect\_ratio)；

image\_height = (image\_height < 1) ? 1 : image\_height;  
 image\_height=(image\_height<1)？1：image\_height；

center = point3(0, 0, 0);  
 中心=点3(0,0,0)；

// Determine viewport dimensions.  
 //确定视口尺寸。

auto focal\_length = 1.0;  
 auto focal\_length=1.0；

auto viewport\_height = 2.0;  
 auto viewport\_height=2.0；

auto viewport\_width = viewport\_height \* (double(image\_width)/image\_height);  
 auto viewport\_width=viewport\_height\*(double(image\_width)/image\_height)；

// Calculate the vectors across the horizontal and down the vertical viewport  
 //计算水平视口和垂直视口向下的向量

edges.  
 边缘。

auto viewport\_u = vec3(viewport\_width, 0, 0); auto viewport\_v = vec3(0, -viewport\_height, 0);  
 auto viewport\_u=vec3(viewport\_width,0,0)；auto viewport\_v=vec3(0,-viewport\_height,0)；

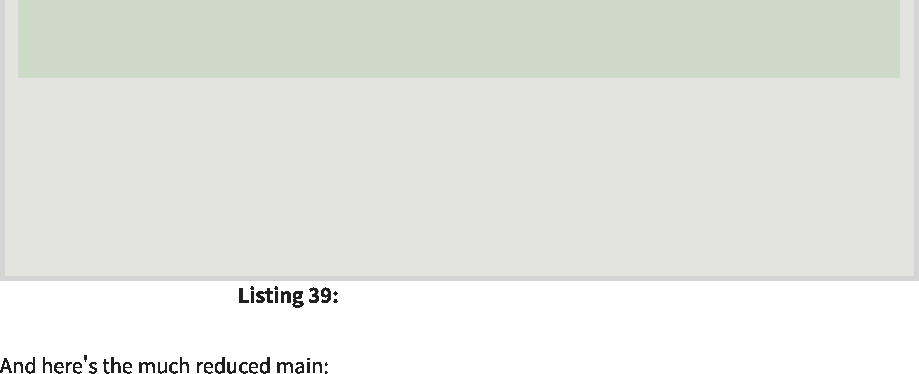
// Calculate the horizontal and vertical delta vectors from pixel to pixel.  
 //计算从像素到像素的水平和垂直增量向量。

pixel\_delta\_u = viewport\_u / image\_width;  
 pixel\_delta\_u=viewport\_u/image\_width；

pixel\_delta\_v = viewport\_v / image\_height;  
 pixel\_delta\_v=viewport\_v/image\_height；

// Calculate the location of the upper left pixel. auto viewport\_upper\_left =  
 //计算左上像素的位置。auto viewport\_upper\_left=

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



center - vec3(0, 0, focal\_length) - viewport\_u/2 - viewport\_v/2; pixel00\_loc = viewport\_upper\_left + 0.5 \* (pixel\_delta\_u + pixel\_delta\_v);  
 center-vec3(0,0,focal\_length)-viewport\_u/2-viewport\_v/2；pixel00\_loc=viewport\_upper\_left+0.5\*(pixel\_delta\_u+pixel\_delta\_v)；

}

color **ray\_color**(const ray& r, const hittable& world) const {  
 color ray\_color(const ray&r，const hittable&world)const{

...

}

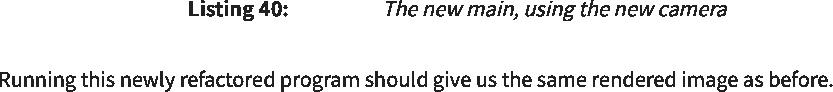
};

#endif  
 #endif

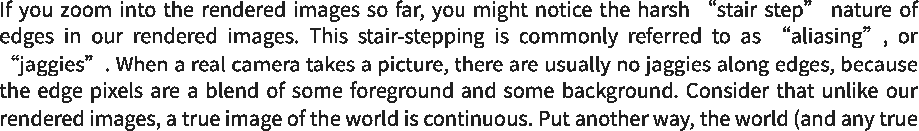
|  |  |
| --- | --- |
| [camera.h]  [摄像机.h] |  |

|  |
| --- |
| #include 'rtweekend.h'  #包括“rtweekend.h” |
| #include 'camera.h'  #包括“camera.h” |
| #include 'hittable.h'  #包括“hittable.h”  #include 'hittable\_list.h'  #include 'sphere.h'  #include'hittable\_list.h' #包括“球体.h” |
| hittable&  hittable& |
| color **ray\_color**(const ray& r, const world) {  ...  color ray\_color(const ray&r，constworld){ ... |
| } |
| int **main**() {  int main(){ |
| hittable\_list world;  hittable\_list world；  world.add(make\_shared<sphere>(point3(0,0,-1), 0.5));  world.add(make\_shared<sphere>(point3(0,-100.5,-1), 100));  world.add(make\_shared<sphere>(point3(0,0,-1),0.5))； world.add(make\_shared<sphere>(point3(0,-100.5,-1),100))；  camera cam;  照相机凸轮；  cam.aspect\_ratio = 16.0 / 9.0;  cam.aspect\_ratio=16.0/9.0；  cam.image\_width = 400;  cam.image\_width=400；  cam.render(world);  凸轮。渲染（世界）； |
| } |

**8. Antialiasing**  
 8.抗锯齿

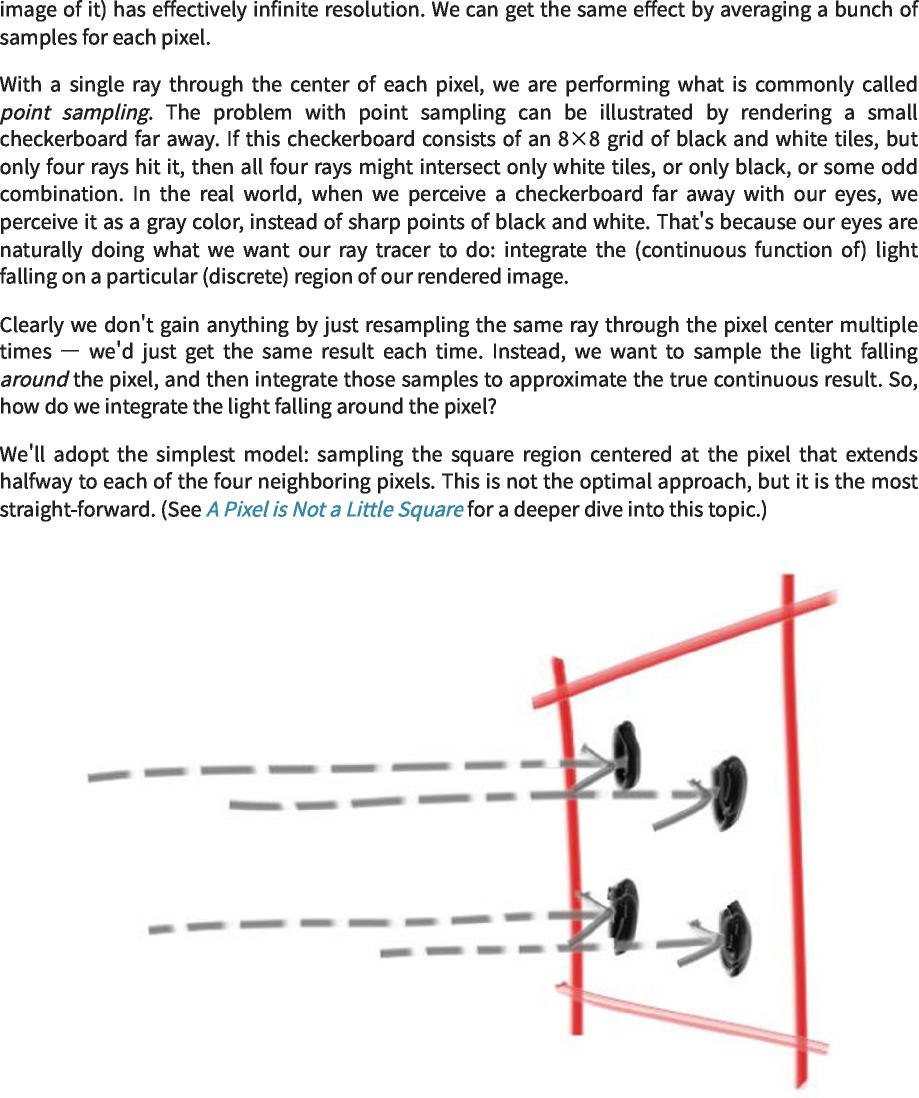


[main.cc]  
 [main.cc]



https://raytracing.github.io/books/RayTracingInOneWeekend.html 50/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 50/120

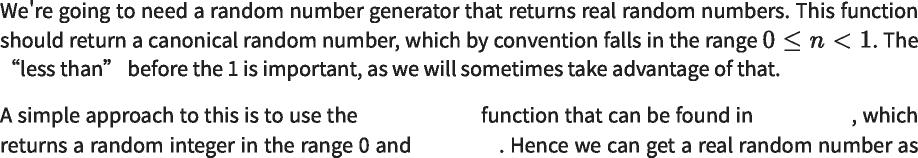
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





**8.1. Some Random Number Utilities**  
 8.1.一些随机数实用程序

https://raytracing.github.io/books/RayTracingInOneWeekend.html 51/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 51/120



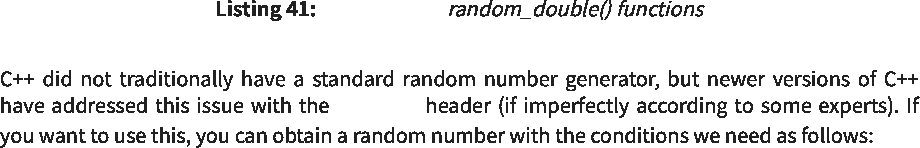
std::rand() <cstdlib>  
 std：：rand()<cstdlib>

RAND\_MAX  
 RAND\_MAX

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |  |  |
| --- | --- | --- |
|  | rtweekend.h  rtweekend.h |  |

|  |  |
| --- | --- |
| #include <cmath>  #include<cmath> |  |
| #include <cstdlib>  #include<cstdlib> |  |
| #include <iostream>  #include <limits>  #include <memory>  ...  #include<iostream> #包括<限制> #include<memory> ...  // Utility Functions  //实用函数  inline double **degrees\_to\_radians**(double degrees) {  inline double degrees\_to\_radians(double degrees){  return degrees \* pi / 180.0;  返回度\*pi/180.0；  } |  |
| inline double **random\_double**() {  inline double random\_double(){  // Returns a random real in [0,1). return std::rand() / (RAND\_MAX + 1.0);  //在[0,1)中返回一个随机实数.return std：：rand()/(RAND\_MAX+1.0);  }  inline double **random\_double**(double min, double max)  内联double random\_double（double min，double max）  // Returns a random real in [min,max).  //在[min，max)中返回一个随机实数。  return min + (max-min)\*random\_double();  return min+(max-min)\*random\_double()；  } | { |
|  |  |



[rtweekend.h]  
 [rtweekend.h]

<random>  
 <随机>

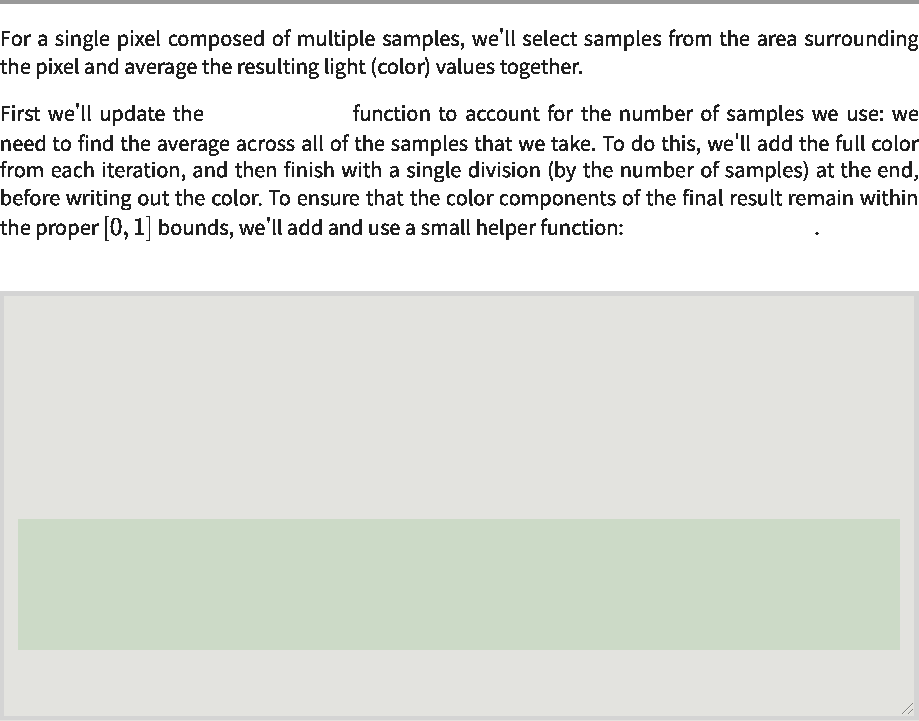
|  |  |
| --- | --- |
| ... |  |
| #include <random>  #include<random> |  |
| ... |  |
| inline double **random\_double**() {  inline double random\_double(){  static std::uniform\_real\_distribution<double> **distribution**(0.0,  静态std：：uniform\_real\_distribution<double>分布(0.0,  static std::mt19937 generator;  静态std：：mt19937发生器；  return distribution(generator);  返回分布（发电机）；  } | 1.0); |
| inline double **random\_double**(double min, double max) {  inline double random\_double(double min,double max){  // Returns a random real in [min,max).  //在[min，max)中返回一个随机实数。  return min + (max-min)\*random\_double();  return min+(max-min)\*random\_double()；  }  ... |  |

|  |  |  |
| --- | --- | --- |
|  | [rtweekend.h]  [rtweekend.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 52/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 52/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**8.2. Generating Pixels with Multiple Samples**  
 8.2.生成具有多个样本的像素



write\_color()  
 write\_color()

interval::clamp(x)  
 间隔：：钳夹（x）

class interval {  
 类间隔{

public:  
 公众：

...

bool **surrounds**(double x) const { return min < x && x < max;  
 bool环绕（double x）const{return min<x&&x<max；

}

double **clamp**(double x) const { if (x < min) return min; if (x > max) return max; return x;  
 双钳（双x）常量{if（x<min）返回min；if（x>max）返回max；返回x；

}

...

};

|  |  |  |
| --- | --- | --- |
|  | [interval.h]  [间隔h] |  |

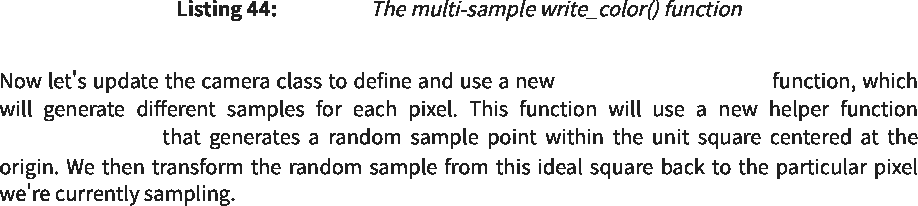


write\_color()  
 write\_color()

https://raytracing.github.io/books/RayTracingInOneWeekend.html 53/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 53/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |  |
| --- | --- |
|  |  |
| #include 'interval.h'  #包括“间隔.h” |  |
| #include 'vec3.h'  using color = vec3;  #包括“vec3.h” 使用color=vec3；  void **write\_color**(std::ostream& out, const color& pixel\_color)  void write\_color(std：：ostream&out，const color&pixel\_color)  auto r = pixel\_color.x();  auto r=pixel\_color.x()；  auto g = pixel\_color.y();  auto g=pixel\_color.y()；  auto b = pixel\_color.z();  auto b=pixel\_color.z()；  // Translate the [0,1] component values to the byte range  //将[0,1]分量值转换为字节范围 | {  [0,255]. |
| static const interval **intensity**(0.000, 0.999);  静态常数区间强度(0.000,0.999)；  int rbyte = int(256 \* intensity.clamp(r));  int rbyte=int(256\*intensity.clamp(r))；  int gbyte = int(256 \* intensity.clamp(g));  int gbyte=int(256\*intensity.clamp(g))；  int bbyte = int(256 \* intensity.clamp(b));  int bbyte=int(256\*intensity.clamp(b))； |  |
| // Write out the pixel color components.  //写出像素颜色分量。  out << rbyte << • • << gbyte << • • << bbyte << •\n•;  }  输出<<rbyte<< •• <<GB字节<< •• <<b字节<< •\n•; } |  |



sample\_square()  
 sample\_square()

[color.h]  
 [颜色.h]

camera::get\_ray(i,j)  
 相机：：get\_ray(i,j)

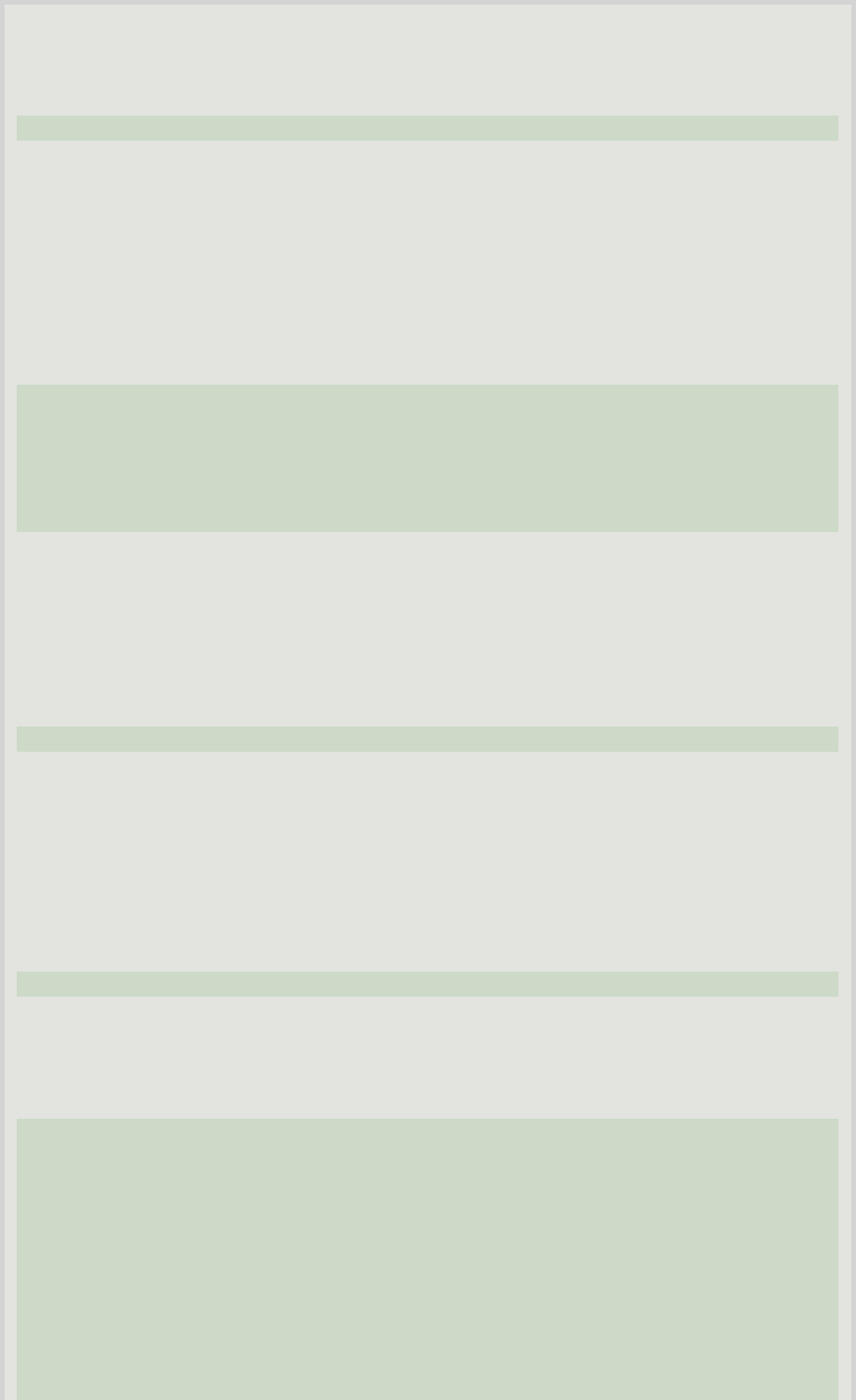
https://raytracing.github.io/books/RayTracingInOneWeekend.html 54/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 54/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 55/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 55/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 56/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 56/120



class camera {   
public:  
 类相机{ 公众：

double aspect\_ratio = 1.0; // Ratio of image width over height  
 双纵横比=1.0；//图像宽高比

int image\_width = 100; // Rendered image width in pixel count  
 intimage\_width=100；//以像素计数表示的渲染图像宽度

int samples\_per\_pixel = 10; // Count of random samples for each pixel  
 intsamples\_per\_pixel=10； //每个像素的随机样本计数

void render(const hittable& world) {   
initialize();  
 void render(const hittable&world){ initialize()；

std::cout << 'P3\n' << image\_width << • • << image\_height << '\n255\n';  
 std：：cout<<'P3\n'<<image\_width<<••<<image\_height<<'\n255\n';

for (int j = 0; j < image\_height; j++) {  
 for(int j=0;j<image\_height;j++){

std::clog << '\rScanlines remaining: ' << (image\_height - j) << • • <<  
 std：：clog<<'\rScanlines remaining：'<<(image\_height-j)<<••<<

std::flush;  
 std：：flush；

for (int i = 0; i < image\_width; i++) {  
 for(int i=0;i<image\_width;i++){

color pixel\_color(0,0,0);  
 color pixel\_color(0,0,0)；

for (int sample = 0; sample < samples\_per\_pixel; sample++) {  
 for(int sample=0;sample<samples\_per\_pixel;sample++){

ray r = get\_ray(i, j);  
 射线r=get\_ray(i，j)；

pixel\_color += ray\_color(r, world);  
 pixel\_color+=ray\_color(r，world)；

}

write\_color(std::cout, pixel\_samples\_scale \* pixel\_color);  
 write\_color(std：：cout，pixel\_samples\_scale\*pixel\_color)；

}

}

std::clog << '\rDone. \n';  
 std：：clog<<'\r完成。\n';

}

...

private:  
 私人：

int image\_height; // Rendered image height  
 intimage\_height； //渲染图像高度

double pixel\_samples\_scale; // Color scale factor for a sum of pixel samples  
 双pixel\_samples\_scale；//像素样本总和的色标因子

point3 center; // Camera center  
 点3中心； //相机中心

point3 pixel00\_loc; // Location of pixel 0, 0  
 点3 pixel00\_loc； //像素0,0的位置

vec3 pixel\_delta\_u; // Offset to pixel to the right  
 vec3pixel\_delta\_u； //向右偏移到像素

vec3 pixel\_delta\_v; // Offset to pixel below  
 vec3 pixel\_delta\_v； //偏移量到下面的像素

void initialize() {  
 void initialize(){

image\_height = int(image\_width / aspect\_ratio);  
 image\_height=int(image\_width/aspect\_ratio)；

image\_height = (image\_height < 1) ? 1 : image\_height;  
 image\_height=(image\_height<1)？1：image\_height；

pixel\_samples\_scale = 1.0 / samples\_per\_pixel;  
 pixel\_samples\_scale=1.0/samples\_per\_pixel；

center = point3(0, 0, 0);  
 中心=点3(0,0,0)；

...

}

ray get\_ray(int i, int j) const {  
 射线get\_ray(int i,int j)const{

// Construct a camera ray originating from the origin and directed at  
 //构造一条源自原点并指向

randomly sampled  
 随机抽样的

// point around the pixel location i, j.  
 //像素位置i，j周围的点。

auto offset = sample\_square();  
 auto offset=sample\_square()；

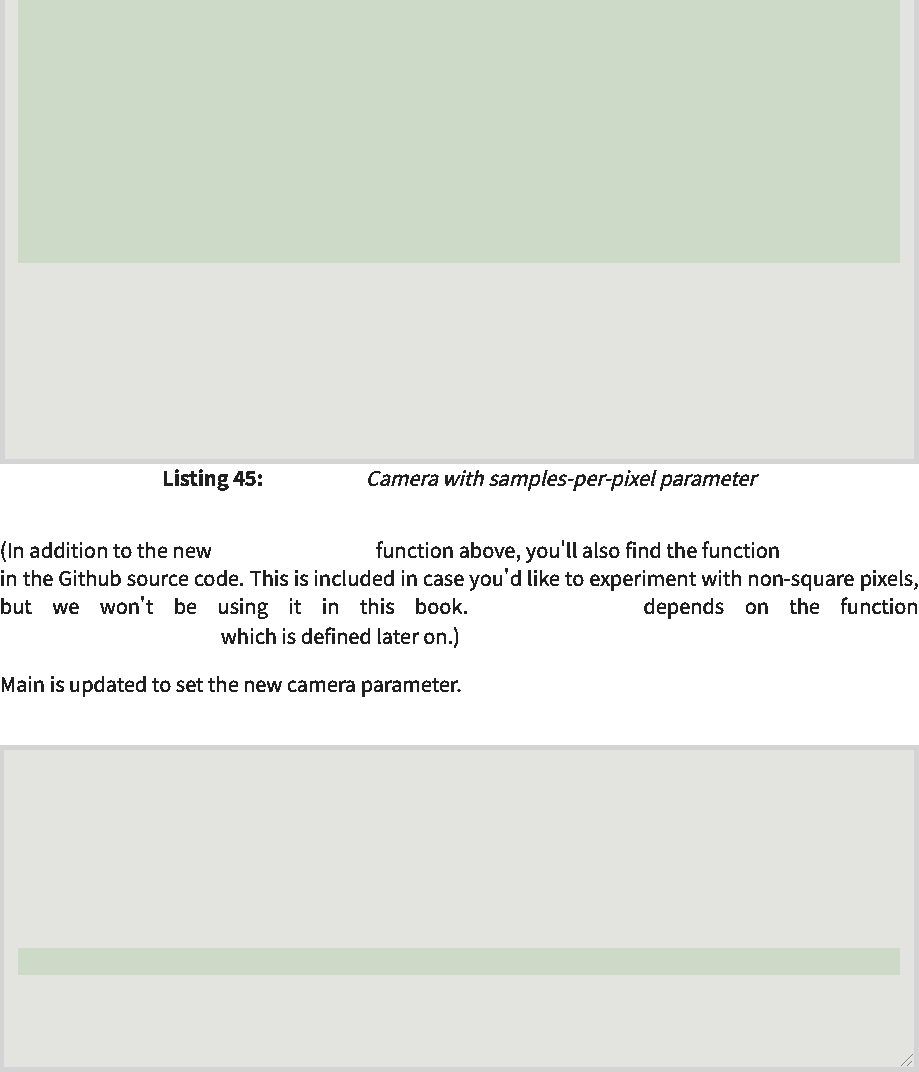
auto pixel\_sample = pixel00\_loc  
 auto pixel\_sample=pixel00\_loc

+ ((i + offset.x()) \* pixel\_delta\_u)  
 +((i+offset.x())\*pixel\_delta\_u)

+ ((j + offset.y()) \* pixel\_delta\_v);  
 +((j+offset.y())\*pixel\_delta\_v)；

auto ray\_origin = center;  
 auto ray\_origin=center；

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



}

square.  
 正方形。

}

...

}

};

#endif  
 #endif

[camera.h]  
 [摄像机.h]

sample\_disk()  
 sample\_disk()

random\_in\_unit\_disk()  
 random\_in\_unit\_disk()

vec3 **sample\_square**() const {  
 vec3 sample\_square()const{

// Returns the vector to a random point in the [-.5,-.5]-[+.5,+.5] unit  
 //将向量返回到[-.5,-.5]-[+.5,+.5]单位中的随机点

color **ray\_color**(const ray& r, const hittable& world) const {  
 color ray\_color(const ray&r，const hittable&world)const{

auto ray\_direction = pixel\_sample - ray\_origin;  
 auto ray\_direction=pixel\_sample-ray\_origin；

return ray(ray\_origin, ray\_direction);  
 返回光线(ray\_origin，ray\_direction)；

return vec3(random\_double() - 0.5, random\_double() - 0.5, 0);  
 返回vec3(random\_double()-0.5,random\_double()-0.5,0)；

sample\_square() sample\_disk()  
 sample\_square()sample\_disk()

int **main**() {  
 int main(){

...

camera cam;  
 照相机凸轮；

cam.aspect\_ratio = 16.0 / 9.0;  
 cam.aspect\_ratio=16.0/9.0；

cam.image\_width = 400;  
 cam.image\_width=400；

cam.samples\_per\_pixel = 100;  
 cam.samples\_per\_pixel=100；

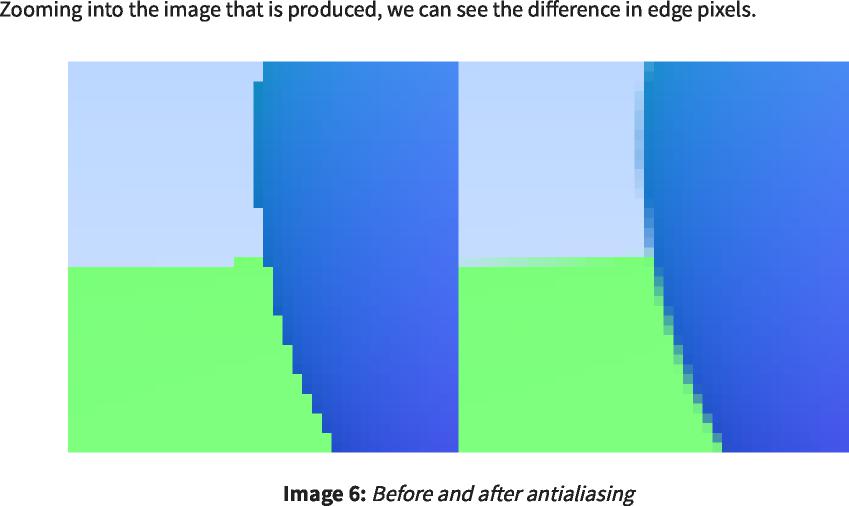
cam.render(world);  
 凸轮。渲染（世界）；

}

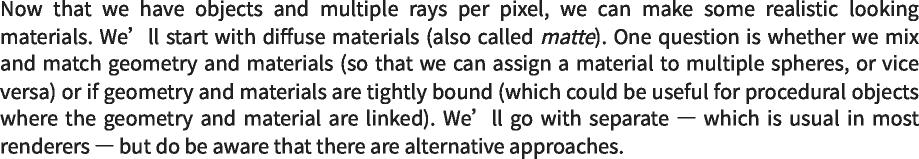
|  |  |  |
| --- | --- | --- |
|  | [main.cc]  [main.cc] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 57/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 57/120

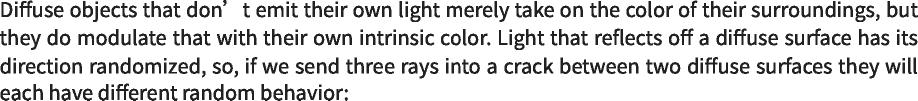
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**9. Diffuse Materials**  
 9.扩散材料

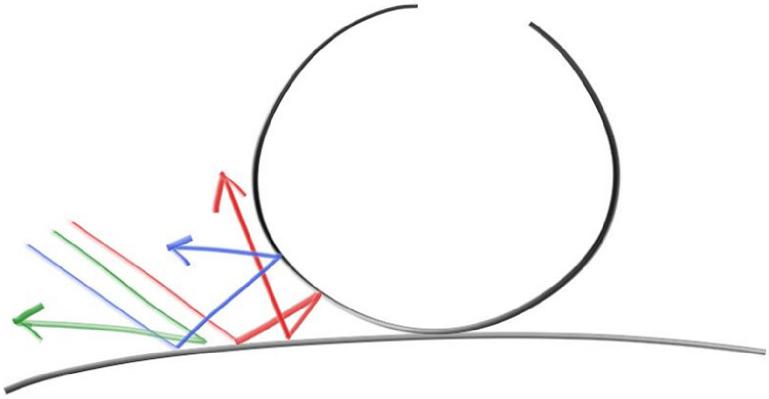


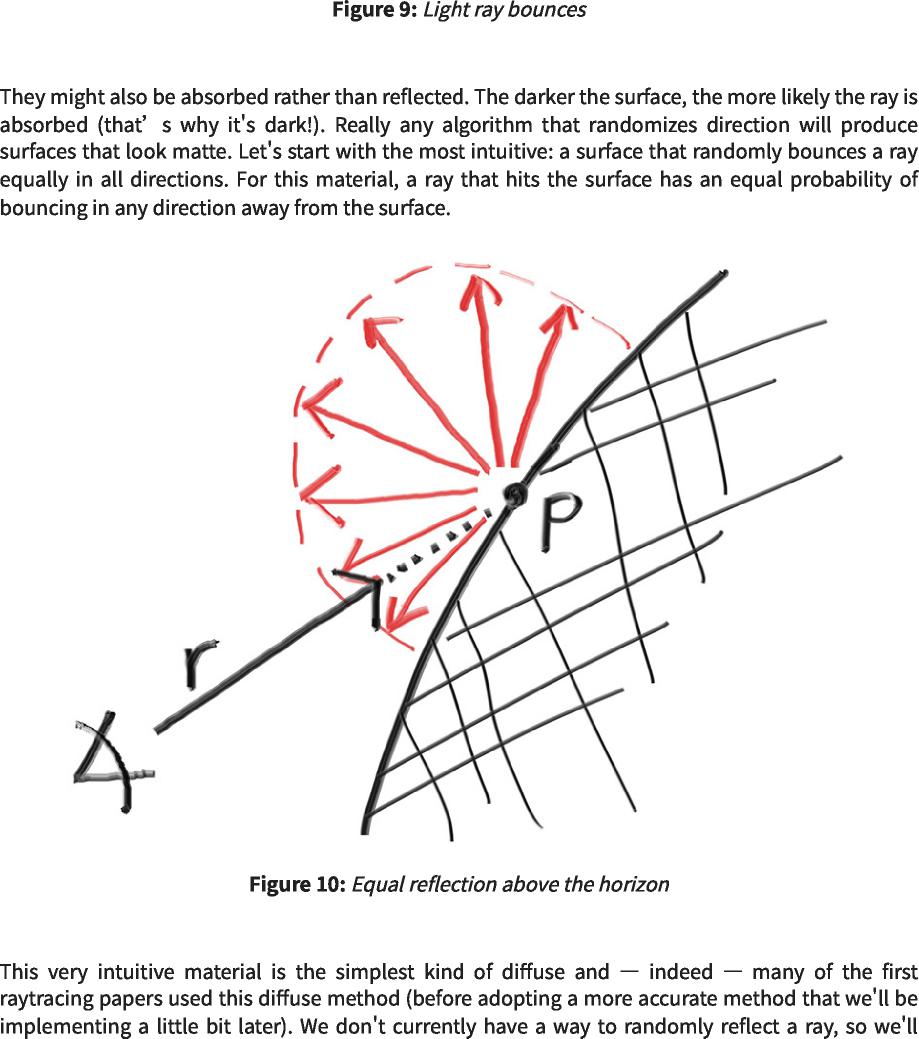
**9.1. A Simple Diffuse Material**  
 9.1.简单的漫射材料



https://raytracing.github.io/books/RayTracingInOneWeekend.html 58/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 58/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

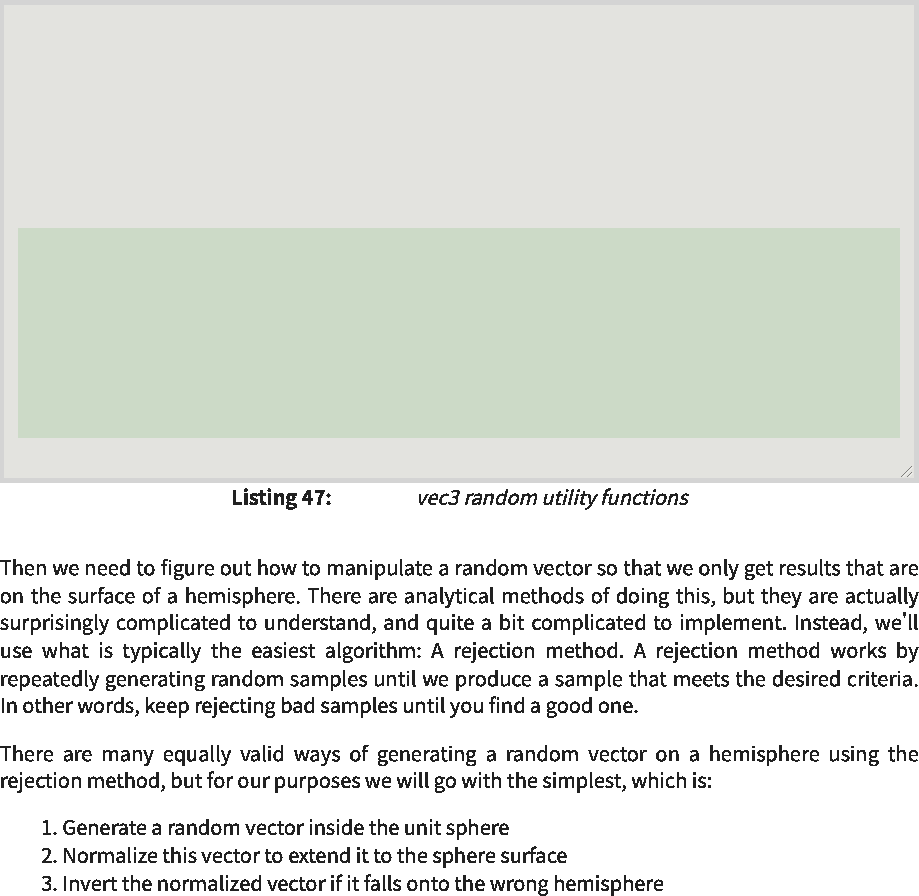




https://raytracing.github.io/books/RayTracingInOneWeekend.html 59/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 59/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





[vec3.h]  
 [vec3.h]

**class vec3 { public:**  
 class vec3{public：

**...**

**double length\_squared() const {**  
 double length\_squared()const{

**return e[**0**]\*e[**0**] + e[**1**]\*e[**1**] + e[**2**]\*e[**2**];**  
 返回e[0]\*e[0]+e[1]\*e[1]+e[2]\*e[2]；

**}**

**static vec3 random() {**  
 静态vec3随机(){

**return vec3(random\_double(), random\_double(), random\_double());**  
 返回vec3(random\_double(),random\_double(),random\_double())；

**}**

**static vec3 random(double min, double max) {**  
 静态vec3随机（双最小值，双最大值）{

**return vec3(random\_double(min,max), random\_double(min,max),**  
 返回vec3(random\_double(min，max),random\_double(min，max),

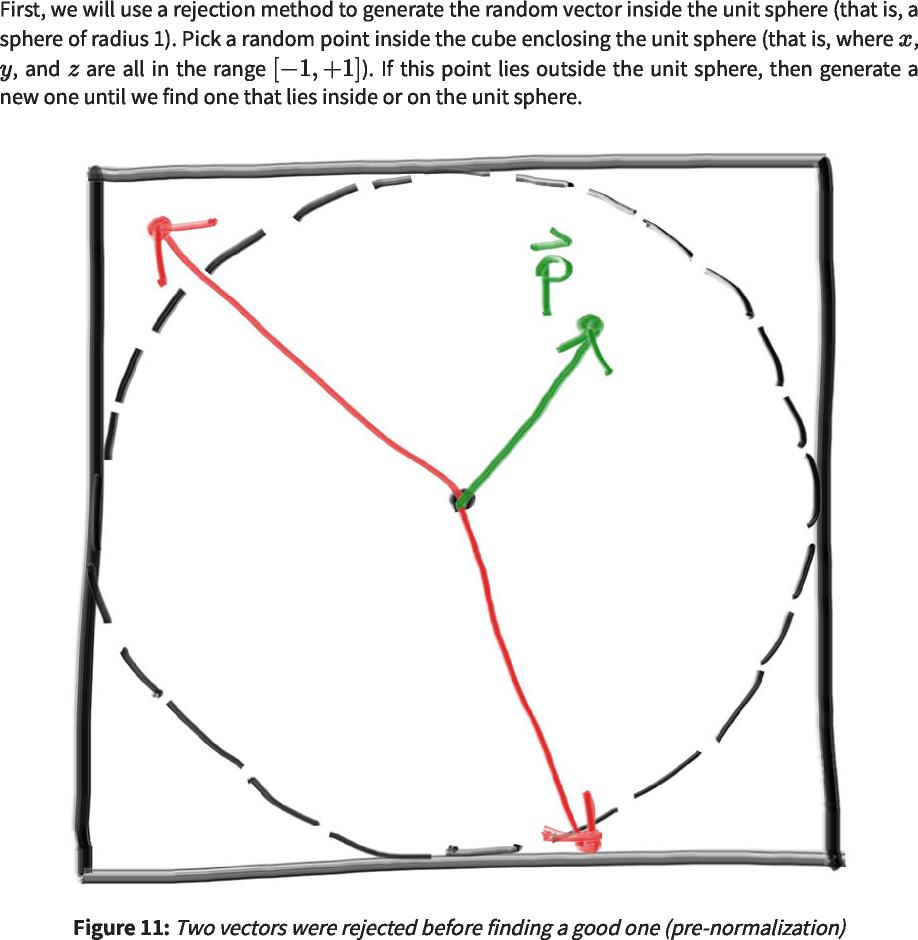
**random\_double(min,max));**  
 random\_double(min，max))；

**}**

**};**

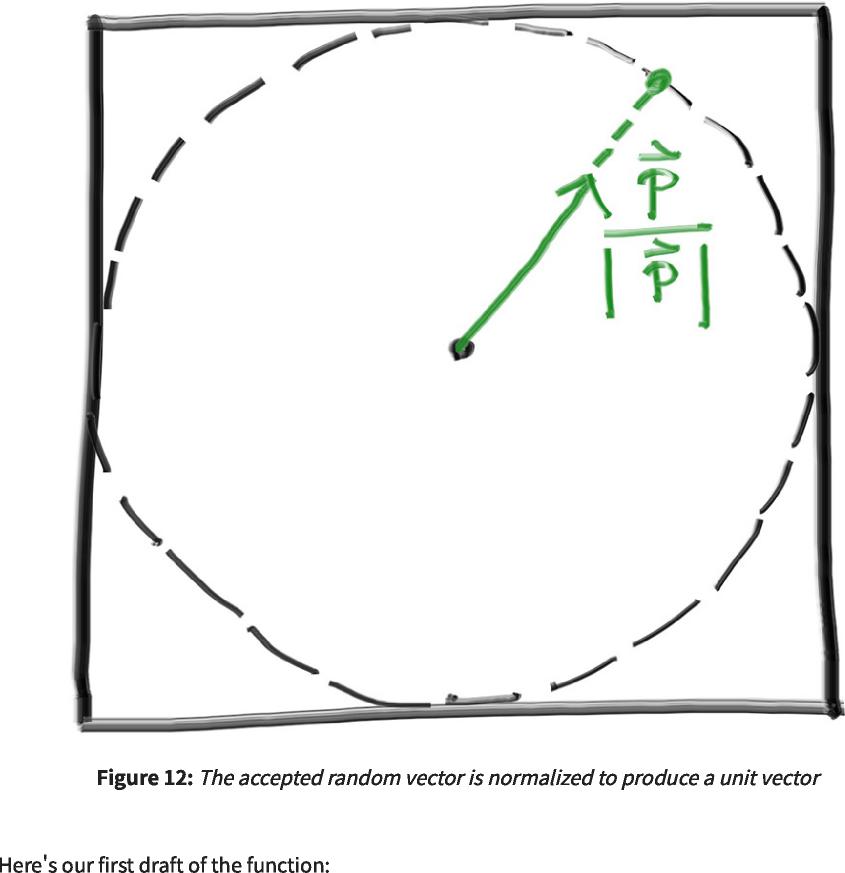
https://raytracing.github.io/books/RayTracingInOneWeekend.html 60/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 60/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

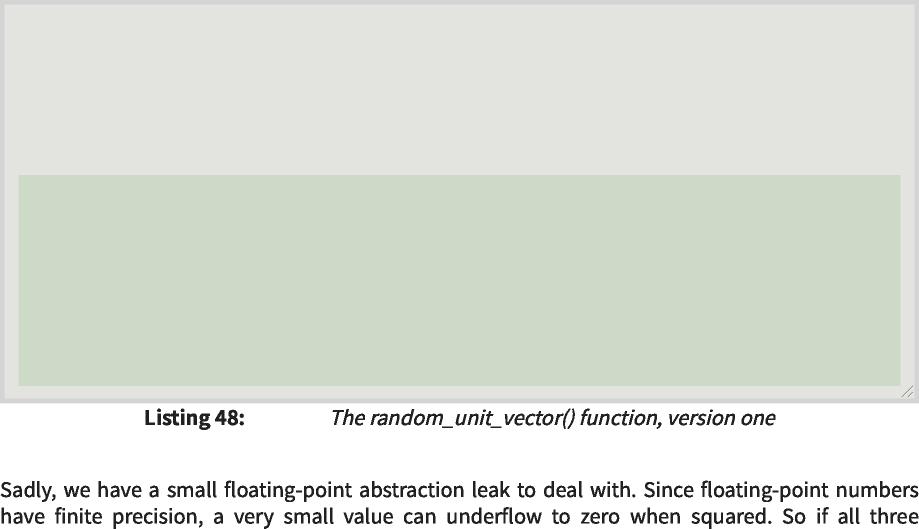


https://raytracing.github.io/books/RayTracingInOneWeekend.html 61/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 61/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



https://raytracing.github.io/books/RayTracingInOneWeekend.html 62/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 62/120



inline vec3 **unit\_vector**(const vec3& v) { return v / v.length();  
 inline vec3 unit\_vector(const vec3&v){return v/v.length()；

inline vec3 **random\_unit\_vector**() { while (true) {  
 inline vec3 random\_unit\_vector(){while(true){

auto p = vec3::random(-1,1); auto lensq = p.length\_squared(); if (lensq <= 1)  
 auto p=vec3：：random(-1,1)；auto lensq=p.length\_squared()；如果(lensq<=1)

return p / sqrt(lensq);  
 返回p/sqrt(lensq)；

}

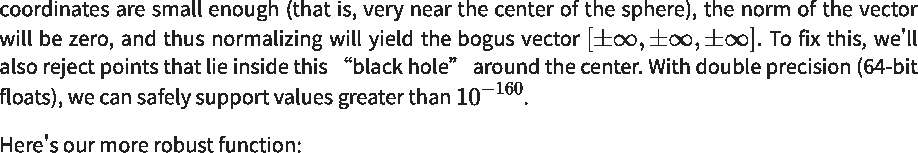
}

...

}

[vec3.h]  
 [vec3.h]

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



inline vec3 **random\_unit\_vector**() {   
while (true) {  
 inline vec3 random\_unit\_vector(){ while(true){

auto p = vec3::random(-1,1);   
auto lensq = p.length\_squared();  
 auto p=vec3：：random(-1,1)； auto lensq=p.length\_squared()；

if (1e-160 < lensq && lensq <= 1)  
 如果(1e-160<lensq&&lensq<=1)

return p / sqrt(lensq);  
 返回p/sqrt(lensq)；

}

}

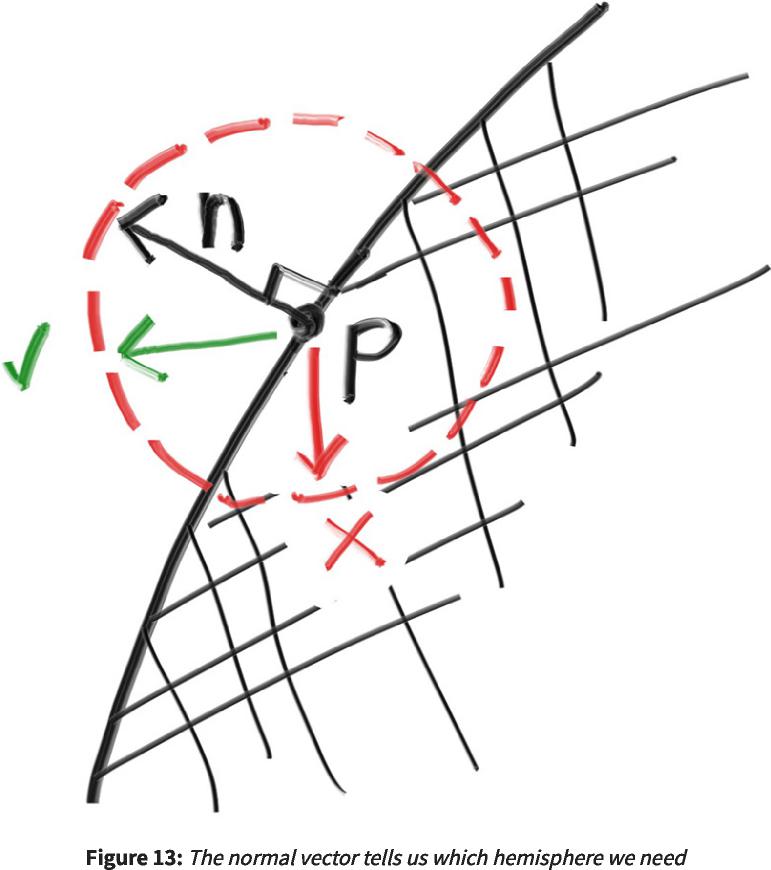


|  |  |  |
| --- | --- | --- |
|  | [vec3.h]  [vec3.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 63/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 63/120

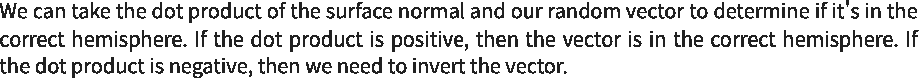
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

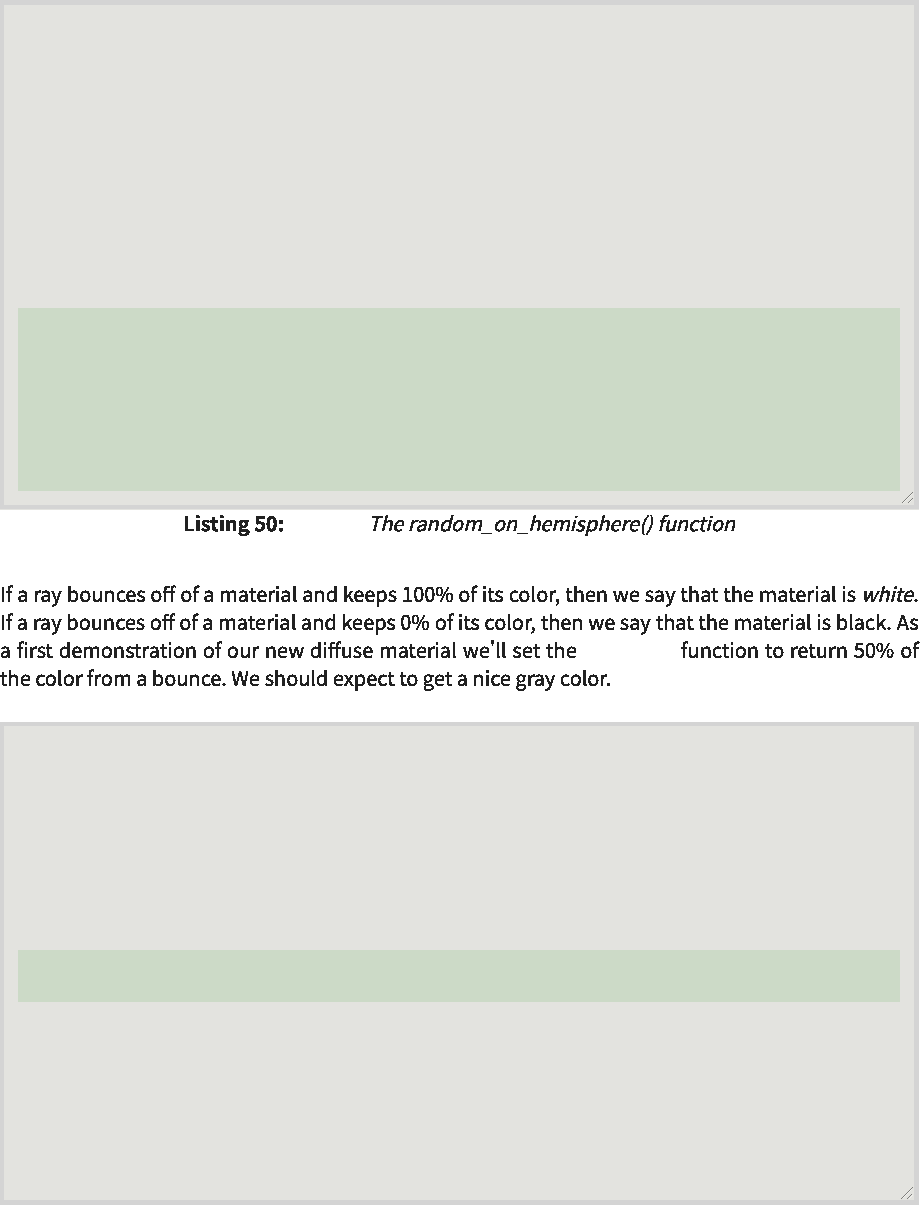




https://raytracing.github.io/books/RayTracingInOneWeekend.html 64/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 64/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





[vec3.h]  
 [vec3.h]

ray\_color  
 射线颜色

...

inline vec3 **random\_unit\_vector**() {  
 inline vec3 random\_unit\_vector(){

while (true) {  
 while(true){

auto p = vec3::random(-1,1);  
 auto p=vec3：：random(-1,1)；

auto lensq = p.length\_squared();  
 auto lensq=p.length\_squared()；

if (1e-160 < lensq && lensq <= 1)  
 如果(1e-160<lensq&&lensq<=1)

return p / sqrt(lensq);  
 返回p/sqrt(lensq)；

}

}

inline vec3 **random\_on\_hemisphere**(const vec3& normal) {  
 内联vec3 random\_on\_hemisphere(const vec3&normal){

vec3 on\_unit\_sphere = random\_unit\_vector();  
 vec3 on\_unit\_sphere=random\_unit\_vector()；

if (dot(on\_unit\_sphere, normal) > 0.0) // In the same hemisphere as the normal  
 如果(dot(on\_unit\_sphere，normal)>0.0)//在与normal相同的半球中

return on\_unit\_sphere;  
 返回on\_unit\_sphere；

else  
 else

return -on\_unit\_sphere;  
 return-on\_unit\_sphere；

}

class camera {  
 类相机{

...

private:  
 私人：

...

color **ray\_color**(const ray& r, const hittable& world) const {  
 color ray\_color(const ray&r，const hittable&world)const{

hit\_record rec;  
 hit\_record rec；

if (world.hit(r, interval(0, infinity), rec)) {  
 if(world.hit(r，interval(0，infinity)，rec)){

vec3 direction = random\_on\_hemisphere(rec.normal); return 0.5 \* ray\_color(ray(rec.p, direction), world);  
 vec3方向=random\_on\_hemisphere(rec.normal)；返回0.5\*ray\_color(ray(rec.p，direction)，world)；

}

vec3 unit\_direction = unit\_vector(r.direction());  
 vec3 unit\_direction=unit\_vector(r.direction())；

auto a = 0.5\*(unit\_direction.y() + 1.0);  
 auto a=0.5\*(unit\_direction.y()+1.0)；

return (1.0-a)\*color(1.0, 1.0, 1.0) + a\*color(0.5, 0.7, 1.0);  
 返回(1.0-a)\*颜色(1.0,1.0,1.0)+A\*颜色(0.5,0.7,1.0)；

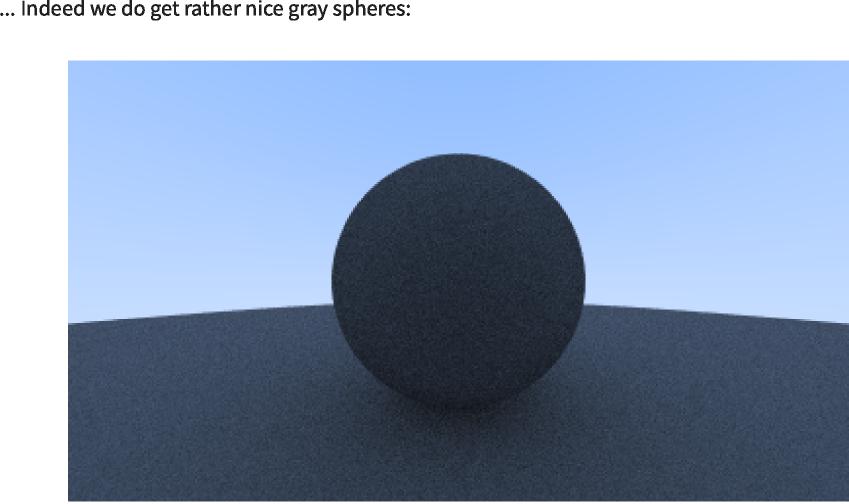
}

};

|  |  |  |
| --- | --- | --- |
|  | [camera.h]  [摄像机.h] |  |

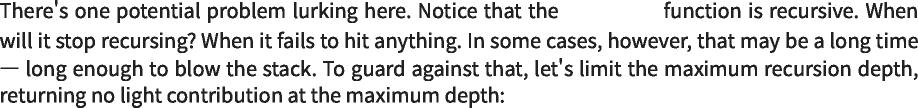
https://raytracing.github.io/books/RayTracingInOneWeekend.html 65/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 65/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





**9.2. Limiting the Number of Child Rays**  
 9.2.限制子光线的数量



ray\_color  
 射线颜色

https://raytracing.github.io/books/RayTracingInOneWeekend.html 66/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 66/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |  |
| --- | --- |
| class camera {  public:  类相机{ 公众：  double aspect\_ratio = 1.0; // Ratio of image width over height  双纵横比=1.0； //图像宽高比  int image\_width = 100; // Rendered image width in pixel count  intimage\_width=100； //以像素计数表示的渲染图像宽度  int samples\_per\_pixel = 10; // Count of random samples for each pixel  intsamples\_per\_pixel=10； //每个像素的随机样本计数 |  |
| int max\_depth = 10; // Maximum number of ray bounces into scene  intmax\_depth=10； //反射到场景中的最大光线数 |  |
| void render(const hittable& world) {  initialize();  void render(const hittable&world){ initialize()；  std::cout << 'P3\n' << image\_width << • • << image\_height << '\n255\n';  std：：cout<<'P3\n'<<image\_width<< ••<<image\_height<<'\n255\n';  for (int j = 0; j < image\_height; j++) {  for(int j=0;j<image\_height;j++){  std::clog << '\rScanlines remaining: ' << (image\_height - j) << • •  std：：clog<<'\rScanlines remaining：'<<(image\_height-j)<< ••  std::flush;  std：：flush；  for (int i = 0; i < image\_width; i++) {  for(int i=0;i<image\_width;i++){  color pixel\_color(0,0,0);  color pixel\_color(0,0,0)；  for (int sample = 0; sample < samples\_per\_pixel; sample++) {  for(int sample=0;sample<samples\_per\_pixel;sample++){  ray r = get\_ray(i, j);  射线r=get\_ray(i，j)； | << |
| pixel\_color += ray\_color(r, max\_depth, world);  pixel\_color+=ray\_color(r，max\_depth，world)； |  |
| }  write\_color(std::cout, pixel\_samples\_scale \* pixel\_color);  write\_color(std：：cout，pixel\_samples\_scale\*pixel\_color)；  }  }  std::clog << '\rDone. \n';  std：：clog<<'\r完成。\n';  }  ...  private:  私人：  ... |  |
| color ray\_color(const ray& r, int depth, const hittable& world) const { // If we've exceeded the ray bounce limit, no more light is gathered. if (depth <= 0)  color ray\_color(const ray&r，int depth，const hittable&world)const{//如果我们超过了光线反弹限制，则不再收集光线。if(depth<=0)  return color(0,0,0);  返回颜色(0,0,0)； |  |
| hit\_record rec;  hit\_record rec；  if (world.hit(r, interval(0, infinity), rec)) { vec3 direction = random\_on\_hemisphere(rec.normal);  if(world.hit(r，interval(0，infinity)，rec)){vec3 direction=random\_on\_hemisphere(rec.normal)； |  |
| return 0.5 \* ray\_color(ray(rec.p, direction), depth-1, world);  返回0.5\*ray\_color(ray(rec.p，direction)，depth-1，world)； |  |
| }  vec3 unit\_direction = unit\_vector(r.direction());  auto a = 0.5\*(unit\_direction.y() + 1.0);  vec3 unit\_direction=unit\_vector(r.direction())； auto a=0.5\*(unit\_direction.y()+1.0)；  return (1.0-a)\*color(1.0, 1.0, 1.0) + a\*color(0.5, 0.7, 1.0);  返回(1.0-a)\*颜色(1.0,1.0,1.0)+A\*颜色(0.5,0.7,1.0)；  }  }; |  |

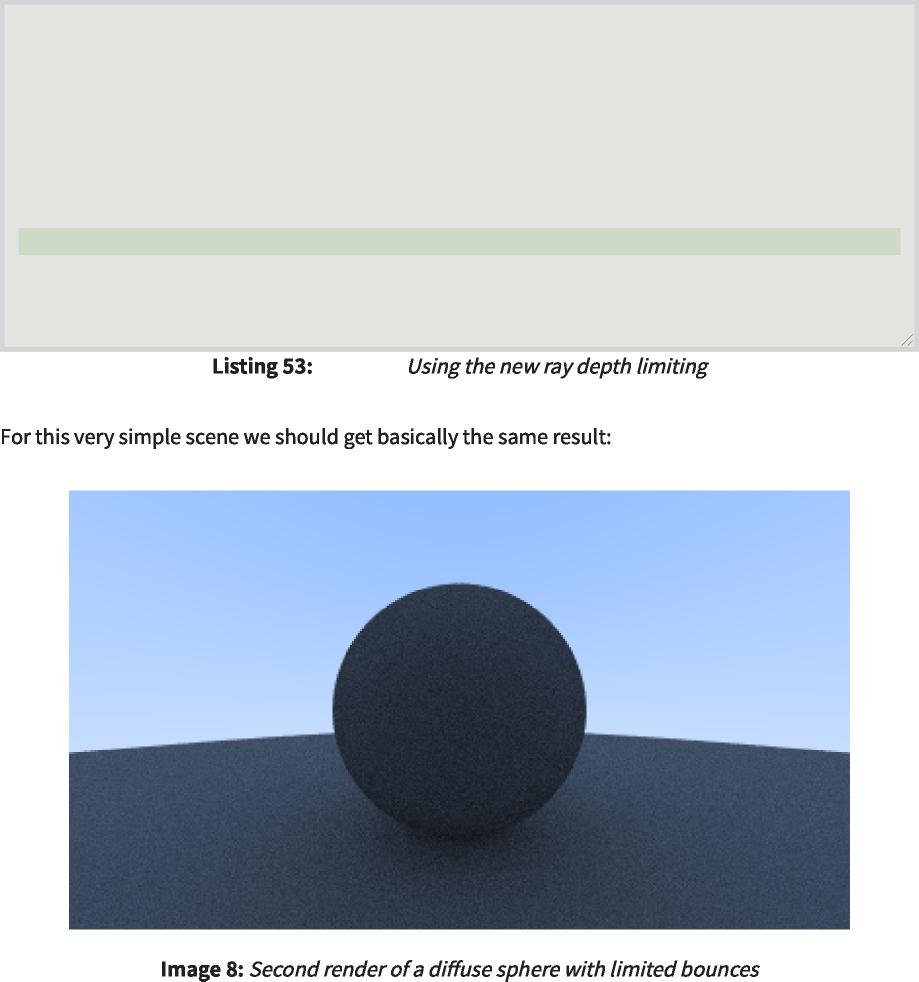
|  |  |  |
| --- | --- | --- |
|  | [camera.h]  [房间。h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 67/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 67/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



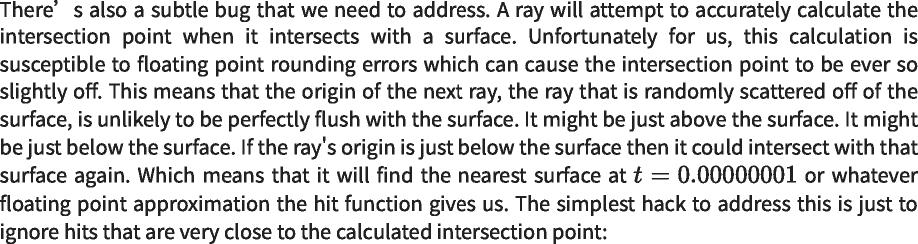
**9.3. Fixing Shadow Acne**  
 9.3.修复阴影痤疮



[main.cc]  
 [main.cc]

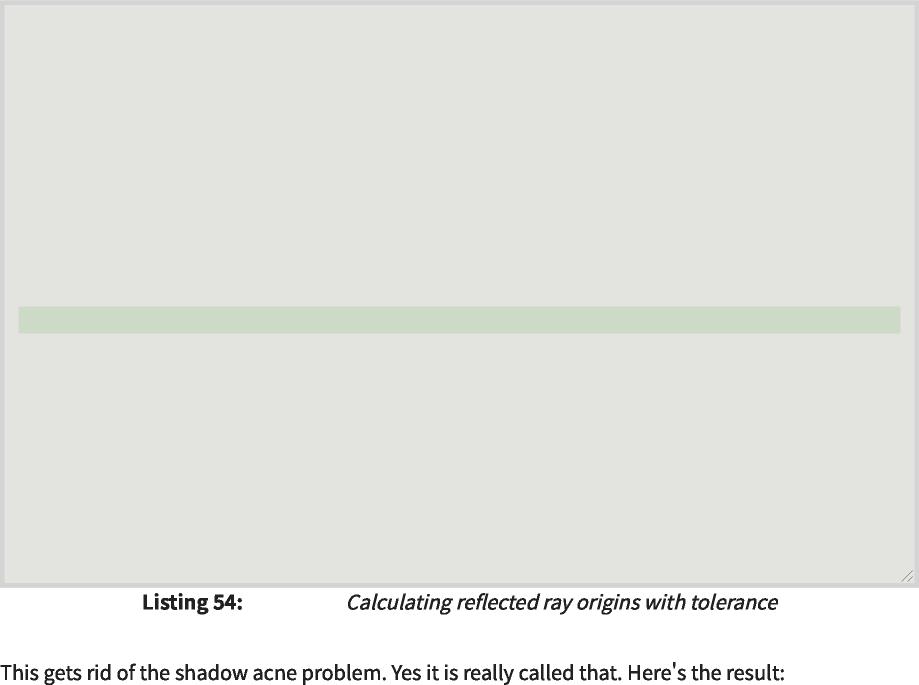
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| int main() {  int main(){  ...  camera cam;  照相机凸轮； |  |  |  |  |
| cam.aspect\_ratio  凸轮纵横比 | = | 16.0 | / | 9.0; |
| cam.image\_width  cam.image\_width | = | 400; |  |  |
| cam.samples\_per\_pixel  cam.samples\_per\_pixel | = | 100; |  |  |
| cam.max\_depth  凸轮最大深度 | = | 50; |  |  |
| cam.render(world);  凸轮。渲染（世界）； |  |  |  |  |

}



https://raytracing.github.io/books/RayTracingInOneWeekend.html 68/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 68/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



[camera.h]  
 [摄像机.h]

class camera {  
 类相机{

...

private:  
 私人：

...

color ray\_color(const ray& r, int depth, const hittable& world) const {  
 color ray\_color(const ray&r，int depth，const hittable&world)const{

// If we've exceeded the ray bounce limit, no more light is gathered.  
 //如果我们超过了光线反弹限制，就不会再收集到更多的光线。

if (depth <= 0)  
 如果(深度<=0)

return color(0,0,0);  
 返回颜色(0,0,0)；

hit\_record rec;  
 hit\_record rec；

if (world.hit(r, interval(0.001, infinity), rec)) {  
 if(world.hit(r，interval(0.001，infinity)，rec)){

vec3 direction = random\_on\_hemisphere(rec.normal);  
 vec3方向=random\_on\_hemisphere(rec.normal)；

return 0.5 \* ray\_color(ray(rec.p, direction), depth-1, world);  
 返回0.5\*ray\_color(ray(rec.p，direction)，depth-1，world)；

}

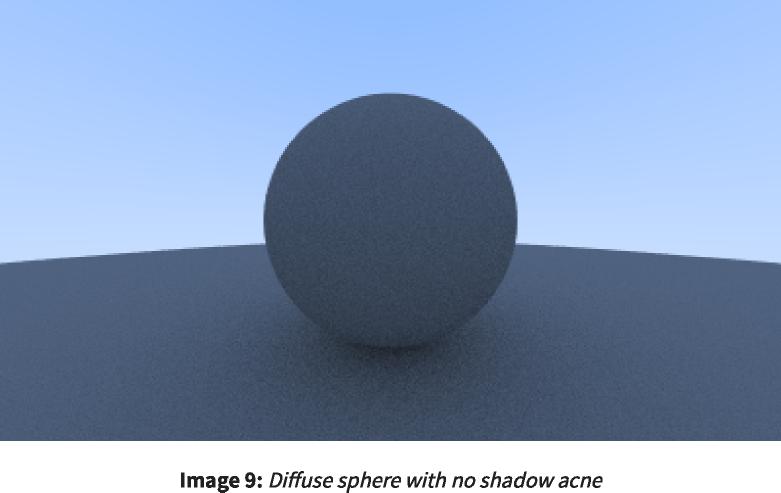
vec3 unit\_direction = unit\_vector(r.direction());  
 vec3 unit\_direction=unit\_vector(r.direction())；

auto a = 0.5\*(unit\_direction.y() + 1.0);  
 auto a=0.5\*(unit\_direction.y()+1.0)；

return (1.0-a)\*color(1.0, 1.0, 1.0) + a\*color(0.5, 0.7, 1.0);  
 返回(1.0-a)\*颜色(1.0,1.0,1.0)+A\*颜色(0.5,0.7,1.0)；

}

};

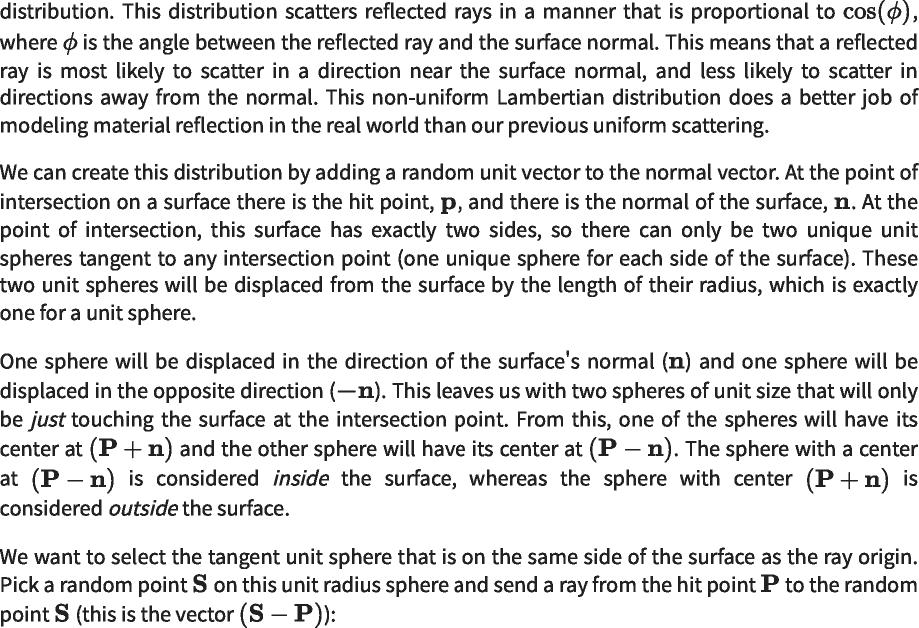


**9.4. True Lambertian Reflection**  
 9.4.真朗伯反射



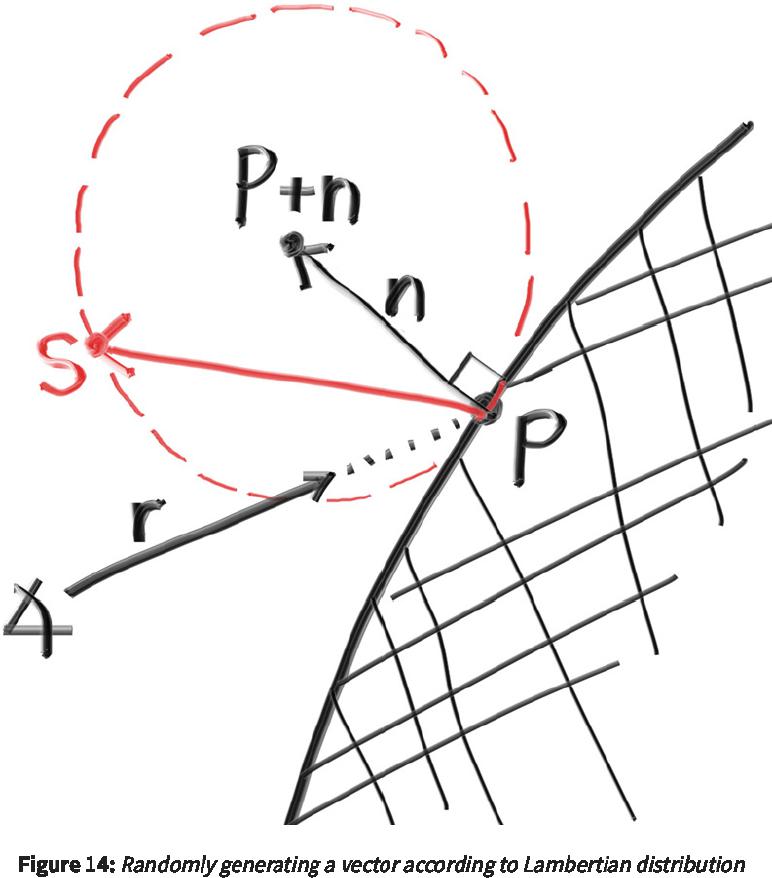
https://raytracing.github.io/books/RayTracingInOneWeekend.html 69/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 69/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



https://raytracing.github.io/books/RayTracingInOneWeekend.html 70/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 70/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

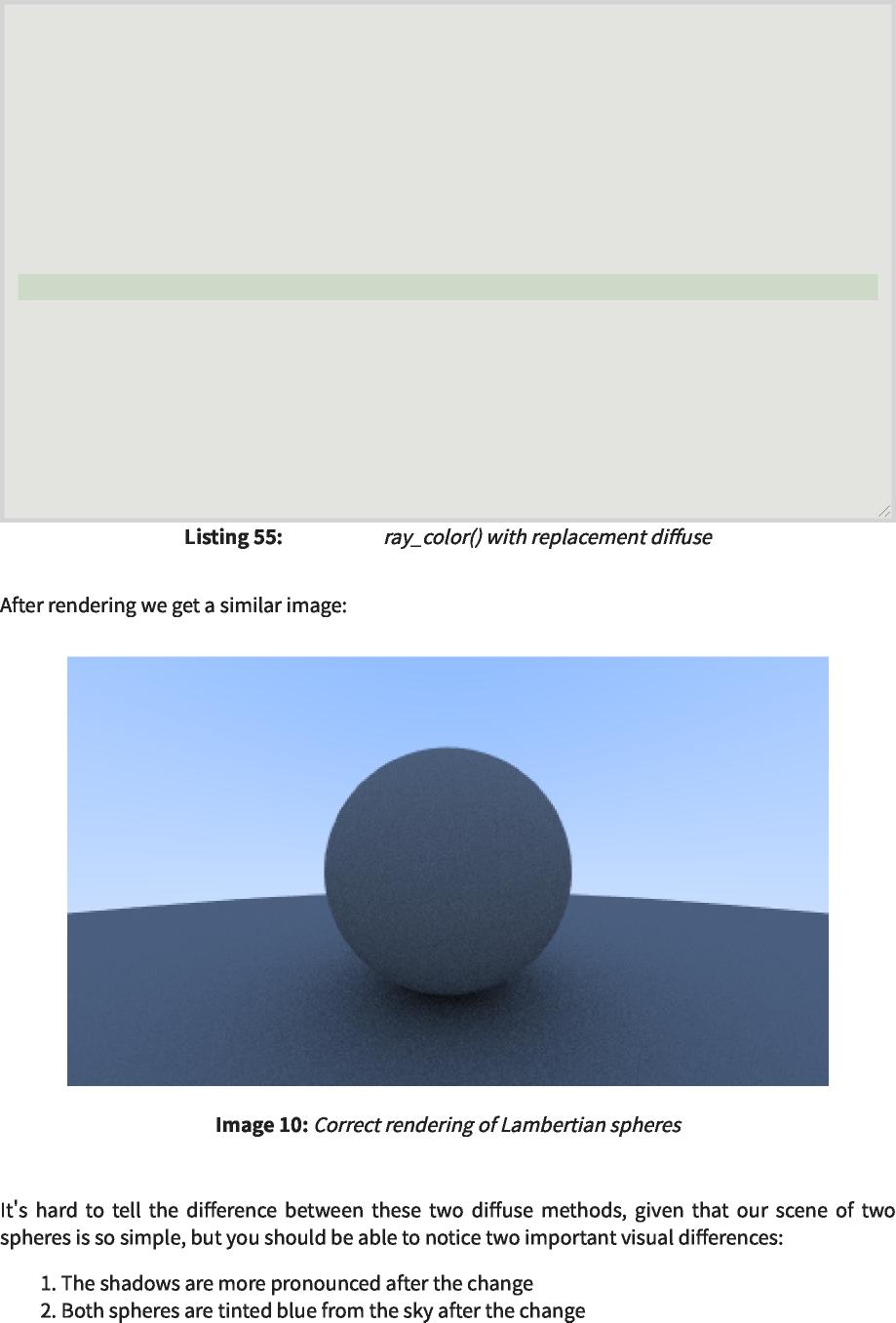


https://raytracing.github.io/books/RayTracingInOneWeekend.html 71/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 71/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



https://raytracing.github.io/books/RayTracingInOneWeekend.html 72/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 72/120



[camera.h]  
 [摄像机.h]

class camera {  
 类相机{

...

color ray\_color(const ray& r, int depth, const hittable& world) const {  
 color ray\_color(const ray&r，int depth，const hittable&world)const{

// If we've exceeded the ray bounce limit, no more light is gathered.  
 //如果我们超过了光线反弹限制，就不会再收集到更多的光线。

if (depth <= 0)  
 如果(深度<=0)

return color(0,0,0);  
 返回颜色(0,0,0)；

hit\_record rec;  
 hit\_record rec；

if (world.hit(r, interval(0.001, infinity), rec)) {  
 if(world.hit(r，interval(0.001，infinity)，rec)){

vec3 direction = rec.normal + random\_unit\_vector();  
 vec3方向=rec.normal+random\_unit\_vector()；

return 0.5 \* ray\_color(ray(rec.p, direction), depth-1, world);  
 返回0.5\*ray\_color(ray(rec.p，direction)，depth-1，world)；

}

vec3 unit\_direction = unit\_vector(r.direction());  
 vec3 unit\_direction=unit\_vector(r.direction())；

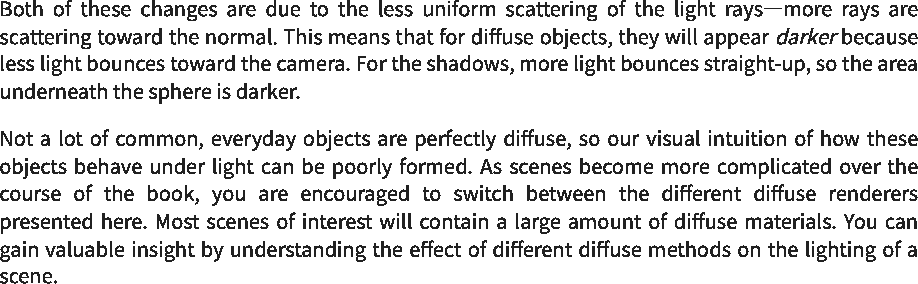
auto a = 0.5\*(unit\_direction.y() + 1.0);  
 auto a=0.5\*(unit\_direction.y()+1.0)；

return (1.0-a)\*color(1.0, 1.0, 1.0) + a\*color(0.5, 0.7, 1.0);  
 返回(1.0-a)\*颜色(1.0,1.0,1.0)+A\*颜色(0.5,0.7,1.0)；

}

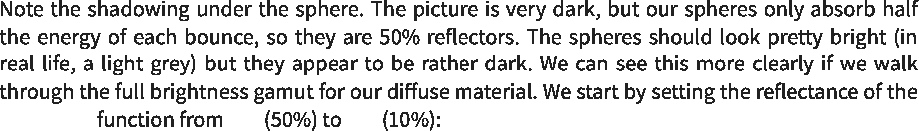
};

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

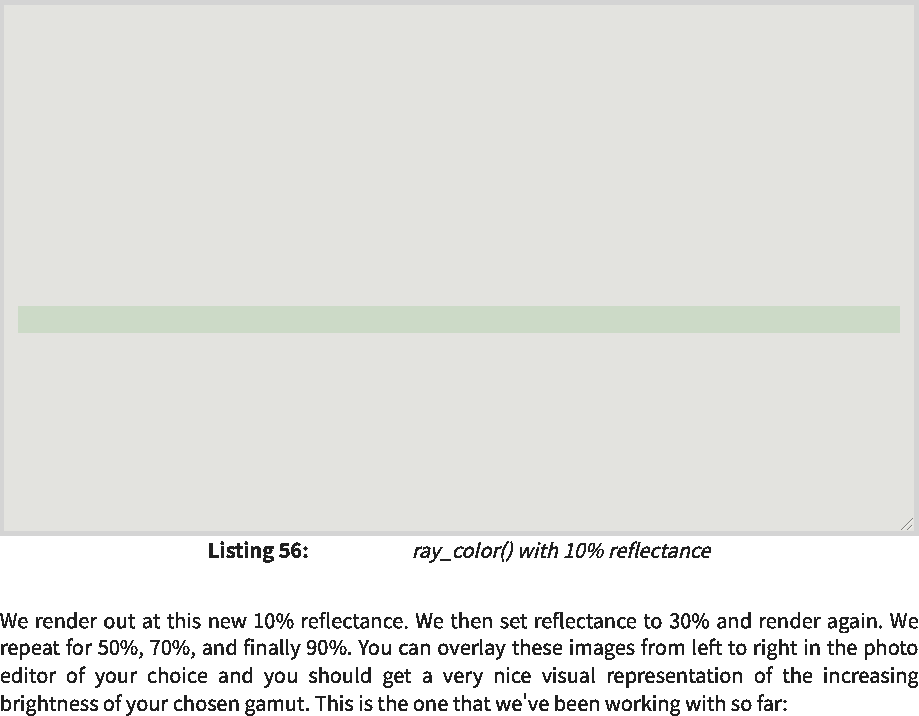


**9.5. Using Gamma Correction for Accurate Color Intensity**  
 9.5.使用伽马校正获得准确的颜色强度

ray\_color 0.5 0.1  
 ray\_color0.50.1



https://raytracing.github.io/books/RayTracingInOneWeekend.html 73/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 73/120



[camera.h]  
 [摄像机.h]

class camera {  
 类相机{

...

color ray\_color(const ray& r, int depth, const hittable& world) const {  
 color ray\_color(const ray&r，int depth，const hittable&world)const{

// If we've exceeded the ray bounce limit, no more light is gathered.  
 //如果我们超过了光线反弹限制，就不会再收集到更多的光线。

if (depth <= **0**)  
 如果(深度<=0)

return color(**0**,**0**,**0**);  
 返回颜色(0,0,0)；

hit\_record rec;  
 hit\_record rec；

if (world.hit(r, interval(**0.001**, infinity), rec)) { vec3 direction = rec.normal + random\_unit\_vector();  
 if(world.hit(r，区间(0.001，无穷大)，rec)){vec3方向=rec.normal+random\_unit\_vector()；

return **0.1** \* ray\_color(ray(rec.p, direction), depth**-1**, world);  
 返回0.1\*ray\_color(ray(rec.p，direction)，depth-1，world)；

}

vec3 unit\_direction = unit\_vector(r.direction());  
 vec3 unit\_direction=unit\_vector(r.direction())；

auto a = **0.5**\*(unit\_direction.y() + **1.0**);  
 auto a=0.5\*(unit\_direction.y()+1.0)；

return (**1.0**-a)\*color(**1.0**, **1.0**, **1.0**) + a\*color(**0.5**, **0.7**, **1.0**);  
 返回(1.0-a)\*颜色(1.0,1.0,1.0)+A\*颜色(0.5,0.7,1.0)；

}

};

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

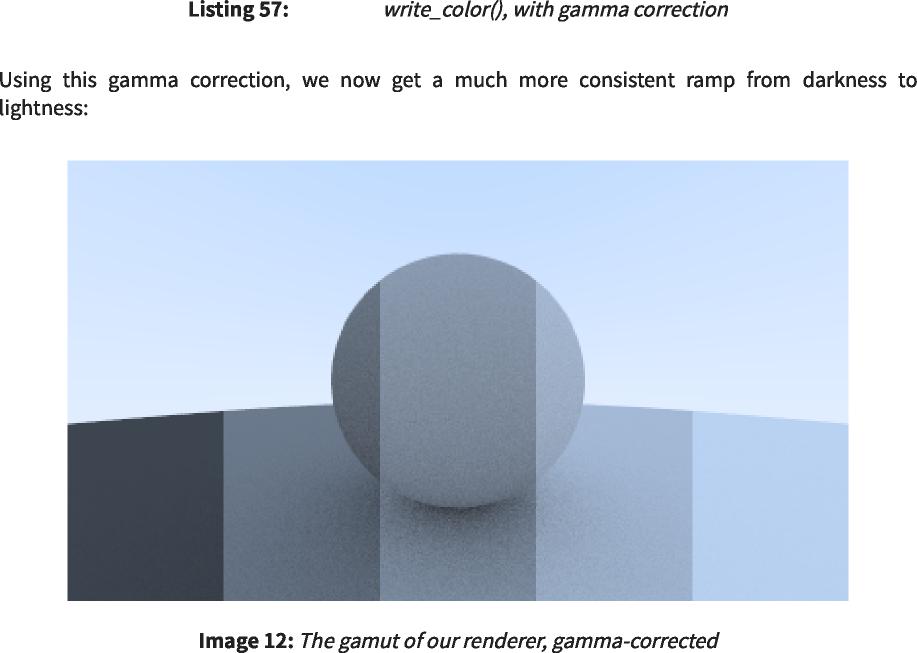


https://raytracing.github.io/books/RayTracingInOneWeekend.html 74/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 74/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |
| --- |
|  |
| inline double **linear\_to\_gamma**(double linear\_component)  内联double linear\_to\_gamma(double linear\_component)  {  if (linear\_component > 0)  如果(linear\_component>0)  return std::sqrt(linear\_component);  返回std：：sqrt(linear\_component)；  return 0;  返回0；  } |
| void **write\_color**(std::ostream& out, const color& pixel\_color) {  void write\_color(std：：ostream&out，const color&pixel\_color){  auto r = pixel\_color.x();  auto r=pixel\_color.x()；  auto g = pixel\_color.y();  auto g=pixel\_color.y()；  auto b = pixel\_color.z();  auto b=pixel\_color.z()； |
| // Apply a linear to gamma transform for gamma 2  //对gamma 2应用线性到gamma变换  r = linear\_to\_gamma(r);  r=线性至γ(r)；  g = linear\_to\_gamma(g);  g=线性至γ(g)；  b = linear\_to\_gamma(b);  b=线性至γ(b)； |
| // Translate the [0,1] component values to the byte range [0,255].  //将[0,1]分量值转换为字节范围[0,255]。  static const interval **intensity**(0.000, 0.999);  静态常数间隔强度(0.000,0.999)；  int rbyte = int(256 \* intensity.clamp(r));  int rbyte=int(256\*intensity.clamp(r))；  int gbyte = int(256 \* intensity.clamp(g));  int gbyte=int(256\*intensity.clamp(g))；  int bbyte = int(256 \* intensity.clamp(b));  int bbyte=int(256\*intensity.clamp(b))；  // Write out the pixel color components.  //写出像素颜色分量。  out << rbyte << • • << gbyte << • • << bbyte << •\n•;  }  输出<<rbyte<< •• <<GB字节<< •• <<b字节<< •\n•; } |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 75/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 75/120



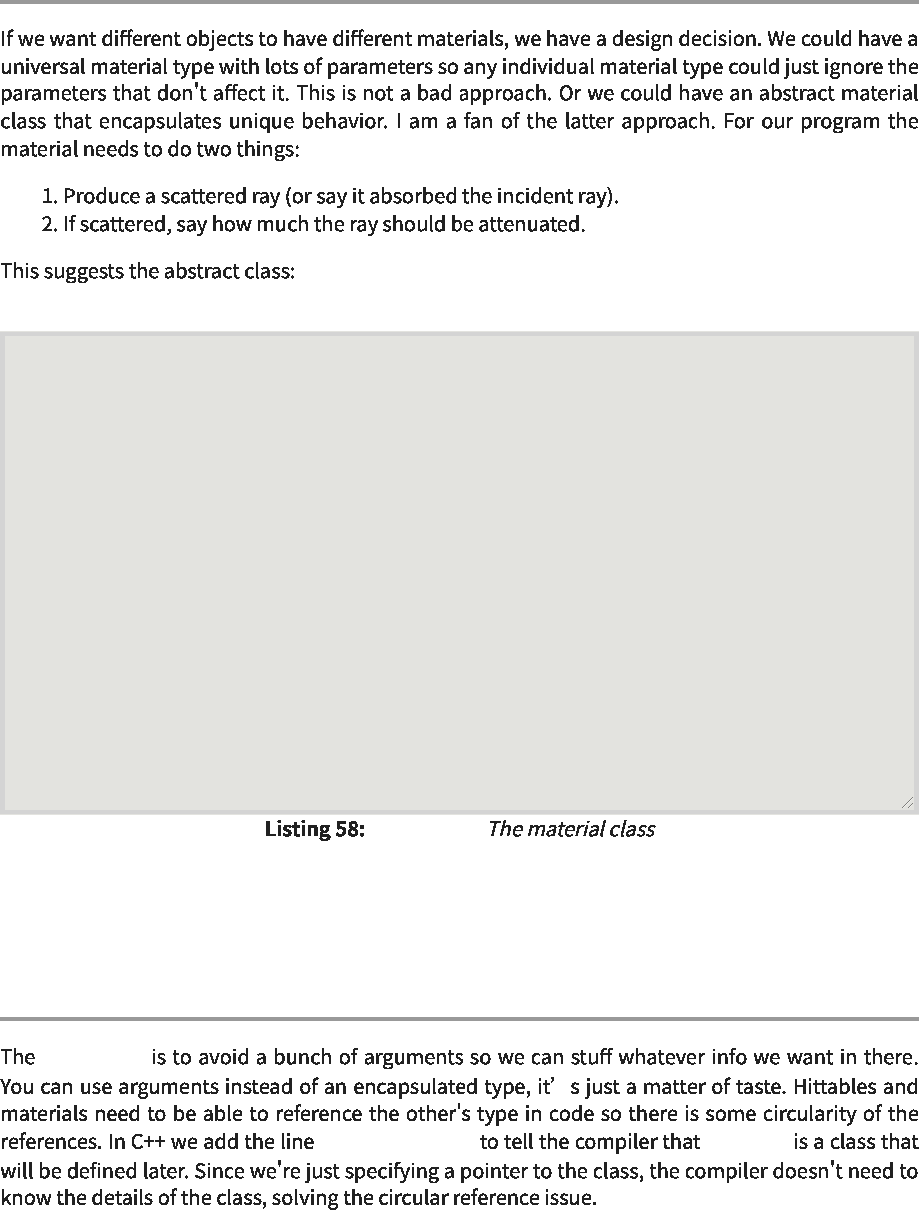
[color.h]  
 [颜色.h]

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**10. Metal**  
 10.金属

**10.1. An Abstract Class for Materials**  
 10.1.材料的抽象类

https://raytracing.github.io/books/RayTracingInOneWeekend.html 76/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 76/120



[material.h]  
 [材料.h]

**10.2. A Data Structure to Describe Ray-Object Intersections**  
 10.2.描述光线-物体相交的数据结构

hit\_record  
 hit\_record

class material; material  
 类材料；材料

**#ifndef MATERIAL\_H #define MATERIAL\_H**  
 #ifndef MATERIAL\_H#define MATERIAL\_H

**#include 'hittable.h'**  
 #包括“hittable.h”

class material {  
 类材料{

public:  
 公众：

virtual ~material() = default;  
 virtual~material()=default；

virtual bool **scatter**(  
 虚拟布尔散点(

const ray& r\_in, const hit\_record& rec, color& attenuation, ray& scattered  
 const ray&r\_in、const hit\_record&rec、color&attenuation、ray&scattered

) const {  
 )常量{

return false;  
 返回false；

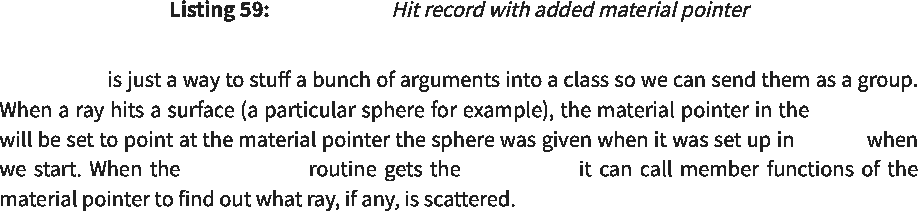
}

};

**#endif**  
 #endif

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |  |
| --- | --- |
|  |  |
| class material;  课堂材料； |  |
| class hit\_record {  class hit\_record{  public:  公众：  point3 p;  点3 p；  vec3 normal;  vec3正常； |  |
| shared\_ptr<material> mat;  shared\_ptr<material>mat； |  |
| double t;  双t；  bool front\_face;  bool front\_face；  void set\_face\_normal(const ray& r, const vec3& outward\_normal)  void set\_face\_normal(const ray&r，const vec3&outward\_normal)  front\_face = dot(r.direction(), outward\_normal) < 0;  front\_face=dot(r.direction(),outward\_normal)<0；  normal = front\_face ? outward\_normal : -outward\_normal;  normal=front\_face？outward\_normal：-outward\_normal；  }  }; | { |



[hittable.h]  
 [hittable.h]

hit\_record  
 hit\_record

hit\_record main()  
 hit\_record main()

ray\_color() hit\_record  
 ray\_color()hit\_record

https://raytracing.github.io/books/RayTracingInOneWeekend.html 77/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 77/120

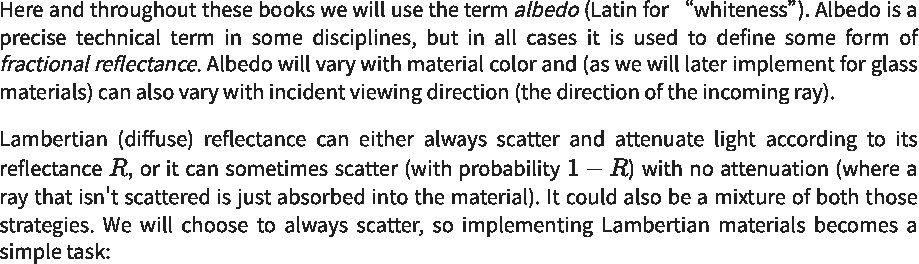
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |  |  |
| --- | --- | --- |
|  | hit\_record  hit\_record |  |

|  |
| --- |
| class sphere : public hittable {  public:  类范围：public hittable{ 公众： |
| sphere(const point3& center, double radius) : center(center),  球体（常量点3&中心，双半径）：中心（中心），  radius(std::fmax(**0**,radius)) {  radius(std：：fmax(0,radius)){  // **TODO:** Initialize the material pointer `mat`.  //TODO：初始化材质指针`mat`。  } |
| bool **hit**(const ray& r, interval ray\_t, hit\_record& rec) const override { ...  bool hit(const ray&r，interval ray\_t，hit\_record&rec)const override{...  rec.t = root;  rec.t=根；  rec.p = r.at(rec.t);  rec.p=r.at(rec.t)；  vec3 outward\_normal = (rec.p - center) / radius;  vec3 outward\_normal=(rec.p-中心)/半径；  rec.set\_face\_normal(r, outward\_normal);  rec.set\_face\_normal(r，outward\_normal)； |
| rec.mat = mat;  rec.mat=mat； |
| return true;  返回true；  }  private:  私人：  point3 center;  点3中心；  double radius;  双半径； |
| shared\_ptr<material> mat;  shared\_ptr<material>mat； |
| }; |

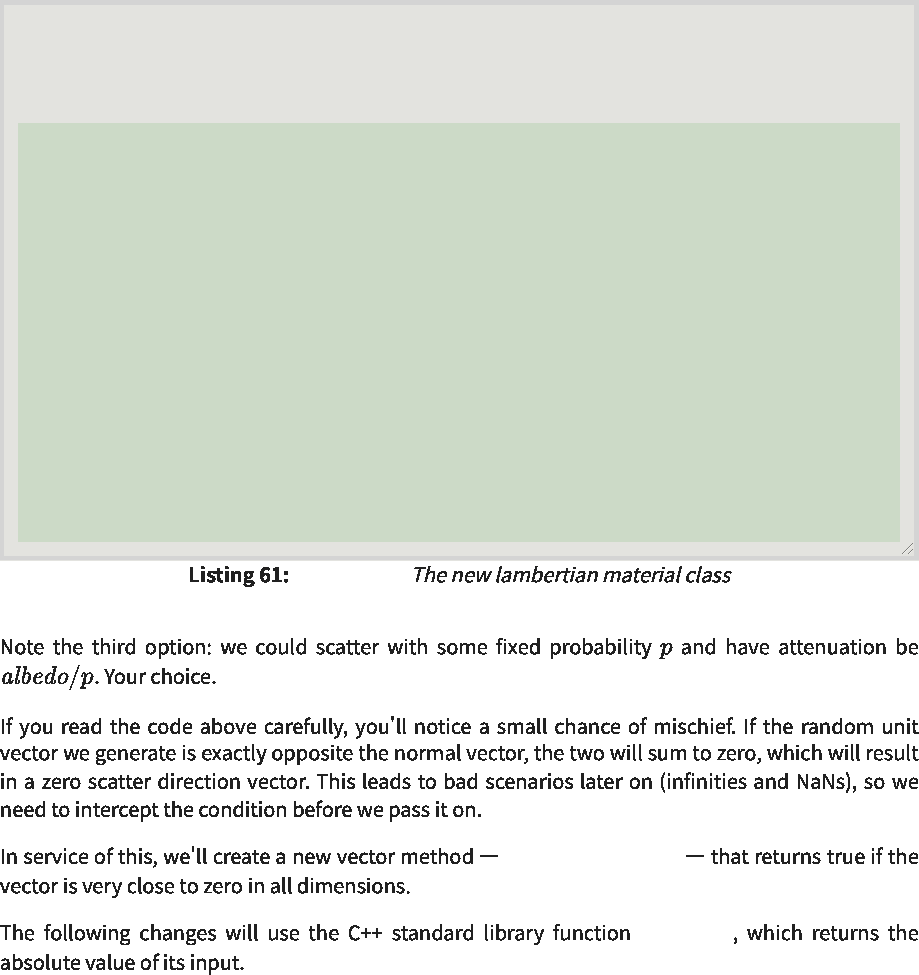
|  |  |  |
| --- | --- | --- |
|  | [sphere.h]  [球体.h] |  |

**10.3. Modeling Light Scatter and Reflectance**  
 10.3.光散射和反射建模



https://raytracing.github.io/books/RayTracingInOneWeekend.html 78/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 78/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



[material.h]  
 [材料.h]

std::fabs  
 标准品：：晶圆厂

vec3::near\_zero()  
 vec3：：near\_zero()

class material {  
 类材料{

...

};

class lambertian : public material {  
 lambertian类：公共材料{

public:  
 公众：

lambertian(const color& albedo) : albedo(albedo) {}  
 朗伯（常量颜色和反照率）：反照率（反照率）{}

bool **scatter**(const ray& r\_in, const hit\_record& rec, color& attenuation, ray&  
 布尔散射（const ray&r\_in、const hit\_record&rec、color&attenuation、ray&attenuation

scattered)  
 分散的)

const override {  
 常量覆盖{

auto scatter\_direction = rec.normal + random\_unit\_vector();  
 auto scatter\_direction=rec.normal+random\_unit\_vector()；

scattered = ray(rec.p, scatter\_direction);  
 散射=射线(rec.p，scatter\_direction)；

attenuation = albedo;  
 衰减=反照率；

return true;  
 返回true；

}

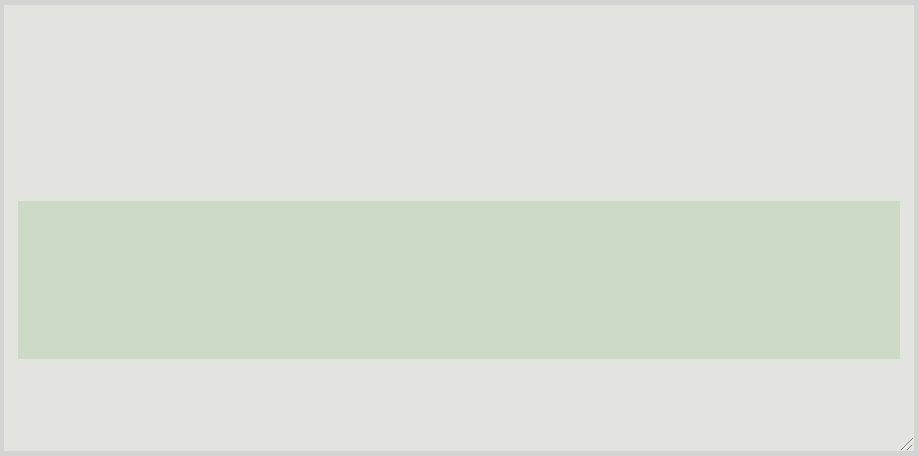
private:  
 私人：

color albedo;  
 颜色反照率；

};

https://raytracing.github.io/books/RayTracingInOneWeekend.html 79/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 79/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



class vec3 {  
 vec3类{

...

double length\_squared() const {  
 double length\_squared()const{

return e[0]\*e[0] + e[1]\*e[1] + e[2]\*e[2];  
 返回e[0]\*e[0]+e[1]\*e[1]+e[2]\*e[2]；

}

bool near\_zero() const {  
 bool near\_zero()const{

// Return true if the vector is close to zero in all dimensions.  
 //如果向量在所有维度上都接近于零，则返回true。

auto s = 1e-8;  
 自动s=1e-8；

return (std::fabs(e[0]) < s) && (std::fabs(e[1]) < s) && (std::fabs(e[2]) <  
 返回(std：：fabs(e[0])<s)&&(std：：fabs(e[1])<s)&&(std：：fabs(e[2])<

s);  
 s);

}

...

};

|  |  |  |
| --- | --- | --- |
|  | [vec3.h]  [vec3.h] |  |

class lambertian : public material {  
 lambertian类：公共材料{

public:  
 公众：

lambertian(const color& albedo) : albedo(albedo) {}  
 朗伯（常量颜色和反照率）：反照率（反照率）{}

bool scatter(const ray& r\_in, const hit\_record& rec, color& attenuation, ray&  
 布尔散射（const ray&r\_in、const hit\_record&rec、color&attenuation、ray&attenuation

scattered)  
 分散的)

const override {  
 常量覆盖{

auto scatter\_direction = rec.normal + random\_unit\_vector();  
 auto scatter\_direction=rec.normal+random\_unit\_vector()；

// Catch degenerate scatter direction   
if (scatter\_direction.near\_zero())   
scatter\_direction = rec.normal;  
 //捕捉简并散射方向 if(scatter\_direction.near\_zero()) scatter\_direction=rec.normal；

scattered = ray(rec.p, scatter\_direction);  
 散射=射线(rec.p，scatter\_direction)；

attenuation = albedo;  
 衰减=反照率；

return true;  
 返回true；

}

private:  
 私人：

color albedo;  
 颜色反照率；

};



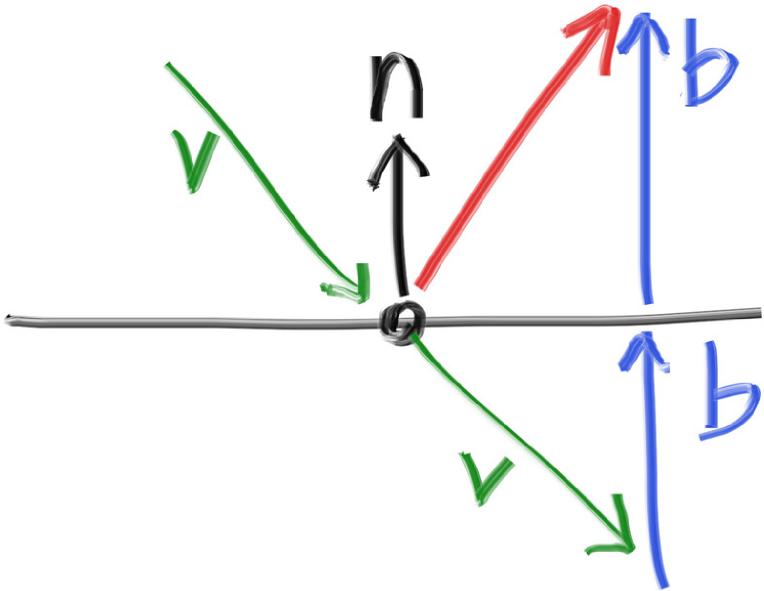
|  |  |  |
| --- | --- | --- |
|  | [material.h]  [材料.h] |  |

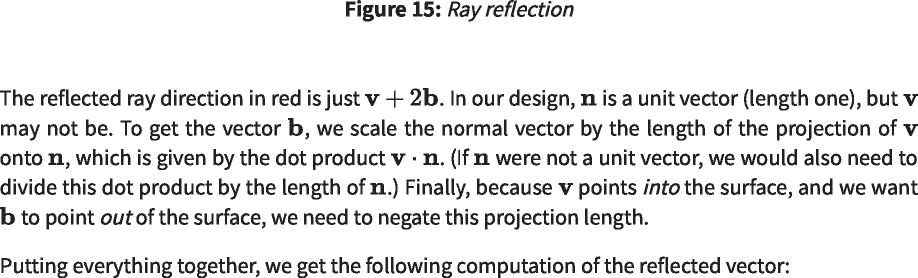
**10.4. Mirrored Light Reflection**  
 10.4.镜面光反射



https://raytracing.github.io/books/RayTracingInOneWeekend.html 80/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 80/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





...

inline vec3 random\_on\_hemisphere(const vec3& normal) {  
 内联vec3 random\_on\_hemisphere(const vec3&normal){

...

}

inline vec3 reflect(const vec3& v, const vec3& n) {  
 内联vec3反射(const vec3&v，const vec3&n){

return v - 2\*dot(v,n)\*n;  
 返回v-2\*dot(v，n)\*n；

}



|  |  |  |
| --- | --- | --- |
|  | [vec3.h]  [vec3.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 81/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 81/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





...

class lambertian : public material {  
 lambertian类：公共材料{

...

};

class metal : public material {  
 金属类：公共材料{

public:  
 公众：

metal(const color& albedo) : albedo(albedo) {}  
 金属（常量颜色和反照率）：反照率（反照率）{}

bool scatter(const ray& r\_in, const hit\_record& rec, color& attenuation, ray&  
 布尔散射（const ray&r\_in、const hit\_record&rec、color&attenuation、ray&attenuation

scattered)  
 分散的)

const override {  
 常量覆盖{

vec3 reflected = reflect(r\_in.direction(), rec.normal);  
 vec3 reflected=reflect(r\_in.direction()，rec.normal)；

scattered = ray(rec.p, reflected);  
 散射=射线(rec.p，反射)；

attenuation = albedo;  
 衰减=反照率；

return true;  
 返回true；

}

private:  
 私人：

color albedo;  
 颜色反照率；

};

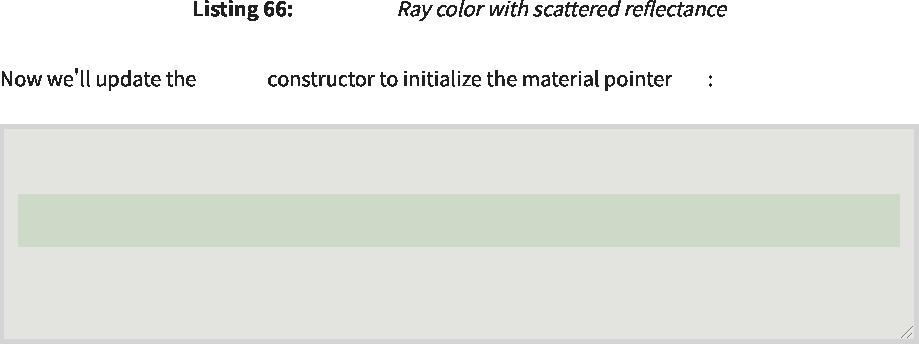
|  |  |  |
| --- | --- | --- |
|  | [material.h]  [材料.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 82/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 82/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |  |  |
| --- | --- | --- |
|  | ray\_color()  ray\_color() |  |

|  |
| --- |
| #include 'hittable.h'  #包括“hittable.h” |
| #include 'material.h'  #包括“材料.h” |
| ...  class camera {  类相机{  ...  private:  私人：  ...  color ray\_color(const ray& r, int depth, const hittable& world) const {  color ray\_color(const ray&r，int depth，const hittable&world)const{  **// If we've exceeded the ray bounce limit, no more light is gathered.**  //如果我们超过了光线反弹限制，就不会再收集到更多的光线。  if (depth <= 0)  如果(深度<=0)  return color(0,0,0);  返回颜色(0,0,0)；  hit\_record rec;  hit\_record rec；  if (world.hit(r, interval(0.001, infinity), rec)) {  如果(world.hit(r, 区间(0.00 1, 无穷大),rec)){ |
| ray scattered;  color attenuation;  射线散射； 颜色衰减；  if (rec.mat->scatter(r, rec, attenuation, scattered))  if(rec.mat->散射(r，rec，衰减，散射))  return attenuation \* ray\_color(scattered, depth-1, world);  返回衰减\*ray\_color(散射，深度-1，世界)；  return color(0,0,0);  返回颜色(0,0,0)； |
| }  vec3 unit\_direction = unit\_vector(r.direction());  auto a = 0.5\*(unit\_direction.y() + 1.0);  vec3 unit\_direction=unit\_vector(r.direction())； auto a=0.5\*(unit\_direction.y()+1.0)；  return (1.0-a)\*color(1.0, 1.0, 1.0) + a\*color(0.5, 0.7, 1.0);  返回(1.0-a)\*颜色(1.0,1.0,1.0)+A\*颜色(0.5,0.7,1.0)；  }  }; |



[camera.h]  
 [摄像机.h]

sphere mat  
 spheremat

class sphere : public hittable { public:  
 类范围：public hittable{public：

sphere(const point3& center, double radius, shared\_ptr<material> mat) : center(center), radius(std::fmax(0,radius)), mat(mat) {}  
 球体(常量点3&中心，双半径，shared\_ptr<material>mat)：中心(中心)，半径(std：：fmax(0，半径))，垫(mat){}

...

};

|  |  |  |
| --- | --- | --- |
|  | [sphere.h]  [球体.h] |  |

**10.5. A Scene with Metal Spheres**  
 10.5.有金属球的场景



https://raytracing.github.io/books/RayTracingInOneWeekend.html 83/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 83/120

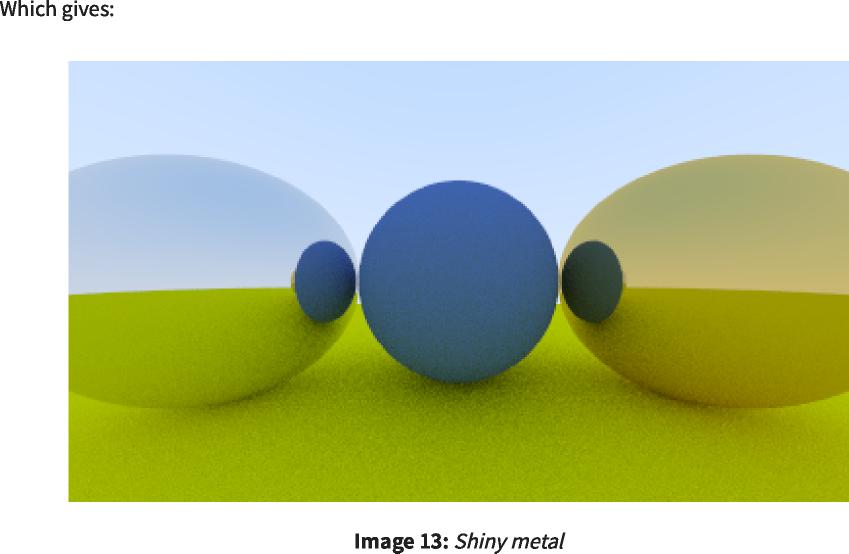
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |  |
| --- | --- |
| #include 'rtweekend.h'  #包括“rtweekend.h”  #include 'camera.h'  #include 'hittable.h'  #包括“camera.h” #包括“hittable.h”  #include 'hittable\_list.h'  #include'hittable\_list.h' |  |
| #include 'material.h'  #包括“材料.h” |  |
| #include 'sphere.h'  #包括“球体.h”  **int** main**() {**  int main(){  **hittable\_list world;**  hittable\_list world； |  |
| auto **material\_ground = make\_shared<lambertian>(color(**0.8**,**  auto material\_ground=make\_shared<lambertian>(color(0.8, | 0.8**,** 0.0**));** |
| auto **material\_center = make\_shared<lambertian>(color(**0.1**,**  auto material\_center=make\_shared<lambertian>(color(0.1, | 0.2**,** 0.5**));** |
| auto **material\_left = make\_shared<metal>(color(**0.8**,** 0.8**,**  auto material\_left=make\_shared<metal>(color(0.8,0.8, | 0.8**));** |
| auto **material\_right = make\_shared<metal>(color(**0.8**,** 0.6**,**  auto material\_right=make\_shared<metal>(color(0.8,0.6, | 0.2**));** |
| **world.add(make\_shared<sphere>(point3(** 0.0**,** -100.5**,** -1.0**),**  world.add(make\_shared<sphere>(point3(0.0,-100.5,-1.0),  **material\_ground));**  material\_ground)); | 100.0**,** |
| **world.add(make\_shared<sphere>(point3(** 0.0**,** 0.0**,** -1.2**),**  world.add(make\_shared<sphere>(point3(0.0,0.0,-1.2),  **material\_center));**  material\_center)); | 0.5**,** |
| **world.add(make\_shared<sphere>(point3(**-1.0**,** 0.0**,** -1.0**),**  world.add(make\_shared<sphere>(point3(-1.0, 0.0,-1.0), | 0.5**, material\_left));**  0.5，material\_left))； |
| **world.add(make\_shared<sphere>(point3(** 1.0**,** 0.0**,** -1.0**),**  world.add(make\_shared<sphere>(point3(1.0,0.0,-1.0),  **material\_right));**  material\_right)); | 0.5**,** |
| **camera cam;**  照相机凸轮；  **cam.aspect\_ratio =** 16.0 **/** 9.0**;**  cam.aspect\_ratio=16.0/9.0；  **cam.image\_width =** 400**;  cam.samples\_per\_pixel =** 100**;**  cam.image\_width=400； cam.samples\_per\_pixel=100；  **cam.max\_depth =** 50**;**  cam.max\_depth=50；  **cam.render(world);  }**  凸轮。渲染（世界）； } |  |

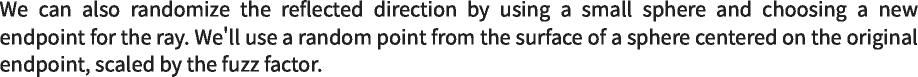
|  |  |  |
| --- | --- | --- |
|  | [main.cc]  [main.cc] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 84/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 84/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

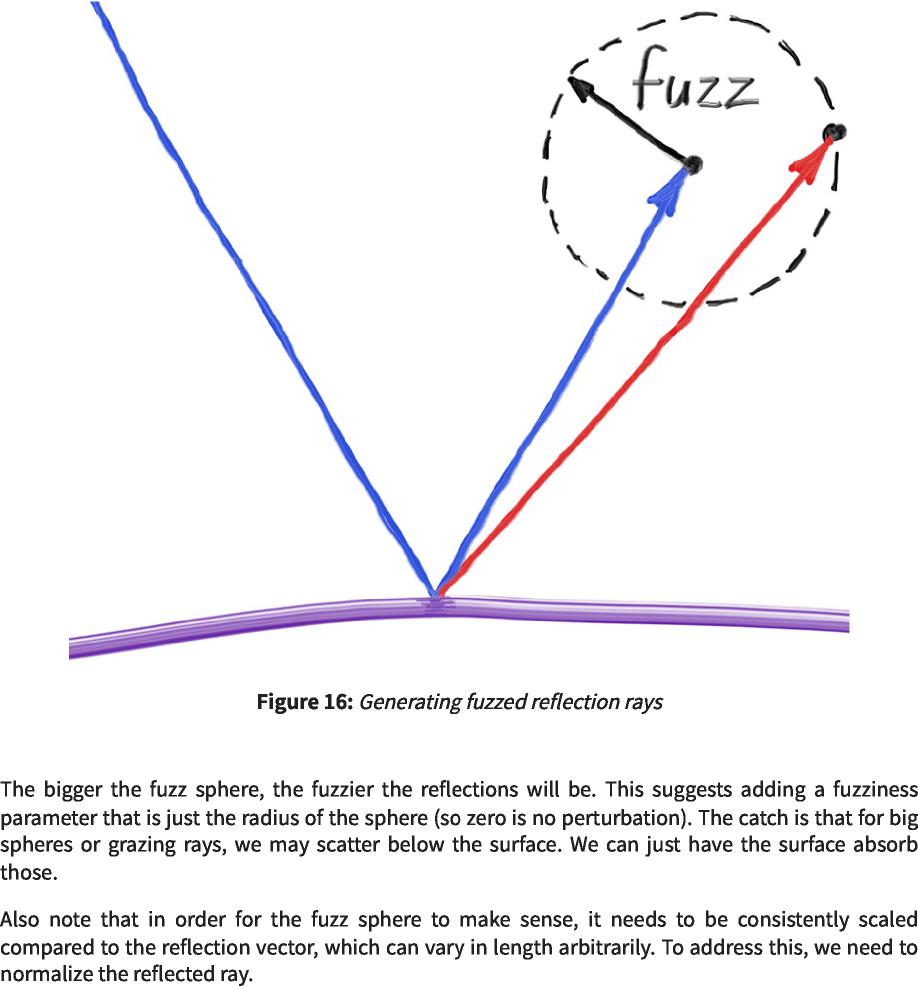


**10.6. Fuzzy Reflection**  
 10.6.模糊反射



https://raytracing.github.io/books/RayTracingInOneWeekend.html 85/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 85/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



https://raytracing.github.io/books/RayTracingInOneWeekend.html 86/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 86/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |  |
| --- | --- |
| class metal : public material {  public:  金属类：公共材料{ 公众： |  |
| metal(const color& albedo, double fuzz) : albedo(albedo), fuzz(fuzz < 1 ? fuzz  金属（恒定颜色和反照率，双绒毛）：反照率（反照率），绒毛（绒毛<1？绒毛  1) {} | : |
| bool scatter(const ray& r\_in, const hit\_record& rec, color& attenuation, ray&  布尔散射（const ray&r\_in、const hit\_record&rec、color&attenuation、ray&attenuation  scattered)  分散的)  const override {  常量覆盖{  vec3 reflected = reflect(r\_in.direction(), rec.normal);  vec3 reflected=reflect(r\_in.direction()，rec.normal)； |  |
| reflected = unit\_vector(reflected) + (fuzz \* random\_unit\_vector());  reflected=unit\_vector(reflected)+(fuzz\*random\_unit\_vector())； |  |
| scattered = ray(rec.p, reflected);  attenuation = albedo;  散射=射线(rec.p，反射)； 衰减=反照率； |  |
| return (dot(scattered.direction(), rec.normal) > 0);  返回(dot(scattered.direction()，rec.normal)>0)； |  |
| }  private:  私人：  color albedo;  颜色反照率； |  |
| double fuzz;  双绒毛； |  |
| }; |  |

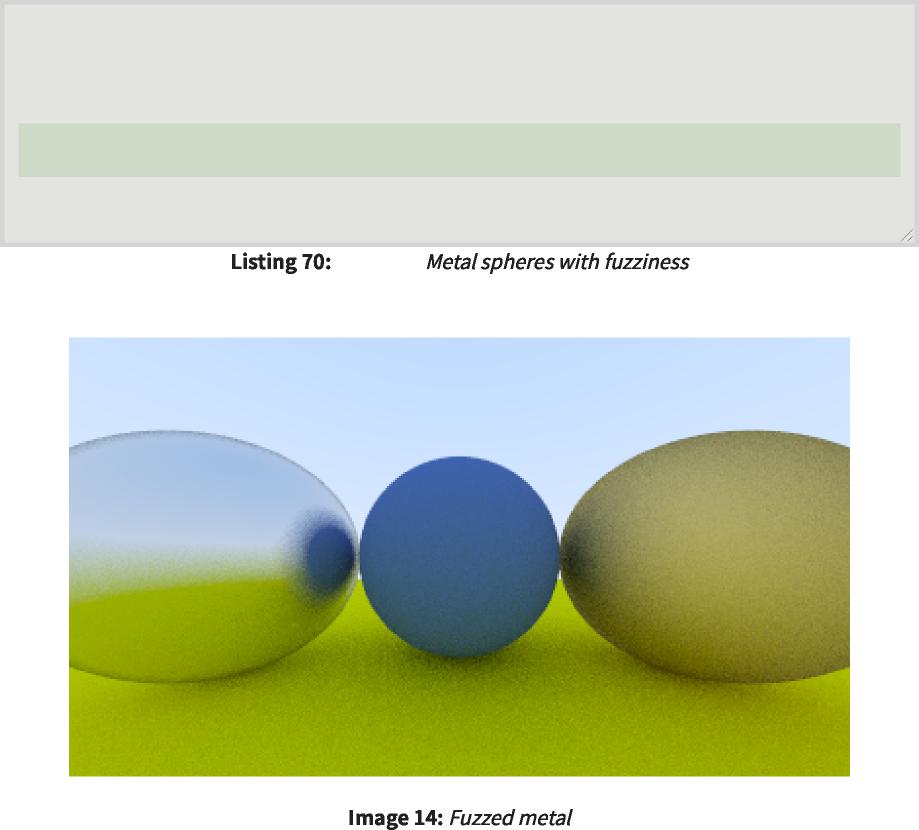
|  |  |  |
| --- | --- | --- |
|  | [material.h]  [材料.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 87/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 87/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**11. Dielectrics**  
 11.电介质

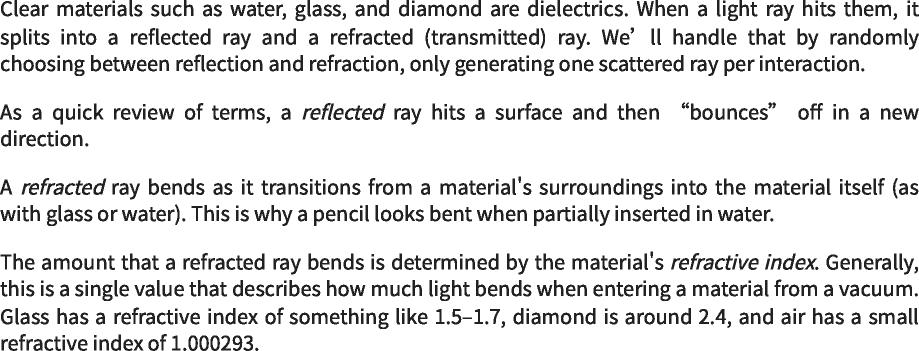


[main.cc]  
 [main.cc]

|  |  |  |
| --- | --- | --- |
| **int** main**() {**  int main(){  **...** |  |  |
| auto **material\_ground = make\_shared<lambertian>(color(**0.8**,**  auto material\_ground=make\_shared<lambertian>(color(0.8, | 0.8**,** | 0.0**));** |
| auto **material\_center = make\_shared<lambertian>(color(**0.1**,**  auto material\_center=make\_shared<lambertian>(color(0.1, | 0.2**,** | 0.5**));** |
| auto **material\_left = make\_shared<metal>(color(**0.8**,** 0.8**,**  auto material\_left=make\_shared<metal>(color(0.8,0.8, | 0.8**),** | 0.3**);** |
| auto **material\_right = make\_shared<metal>(color(**0.8**,** 0.6**,**  auto material\_right=make\_shared<metal>(color(0.8,0.6, | 0.2**),** | 1.0**);** |

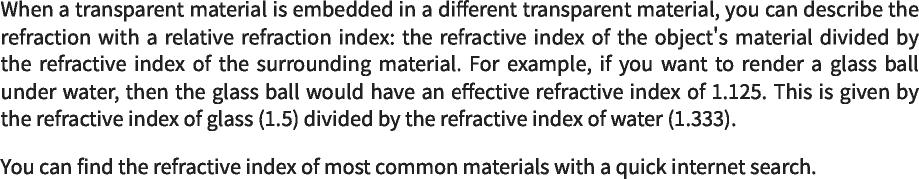
**...**

**}**

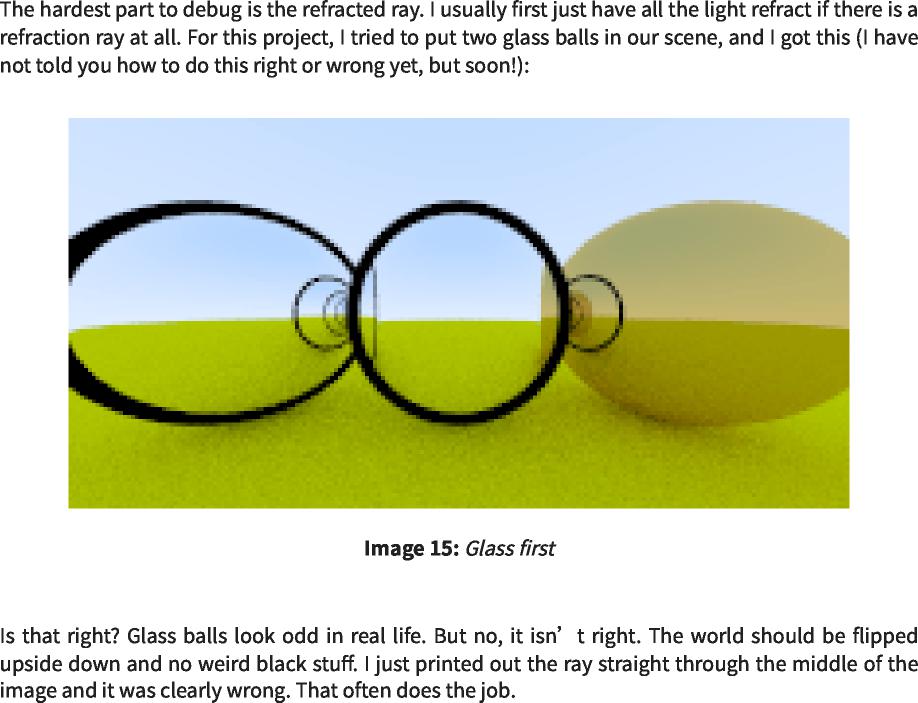


https://raytracing.github.io/books/RayTracingInOneWeekend.html 88/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 88/120

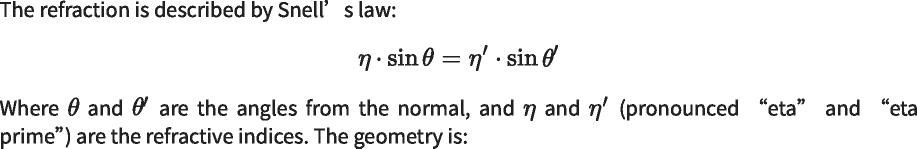
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**11.1. Refraction**  
 11.1.折射

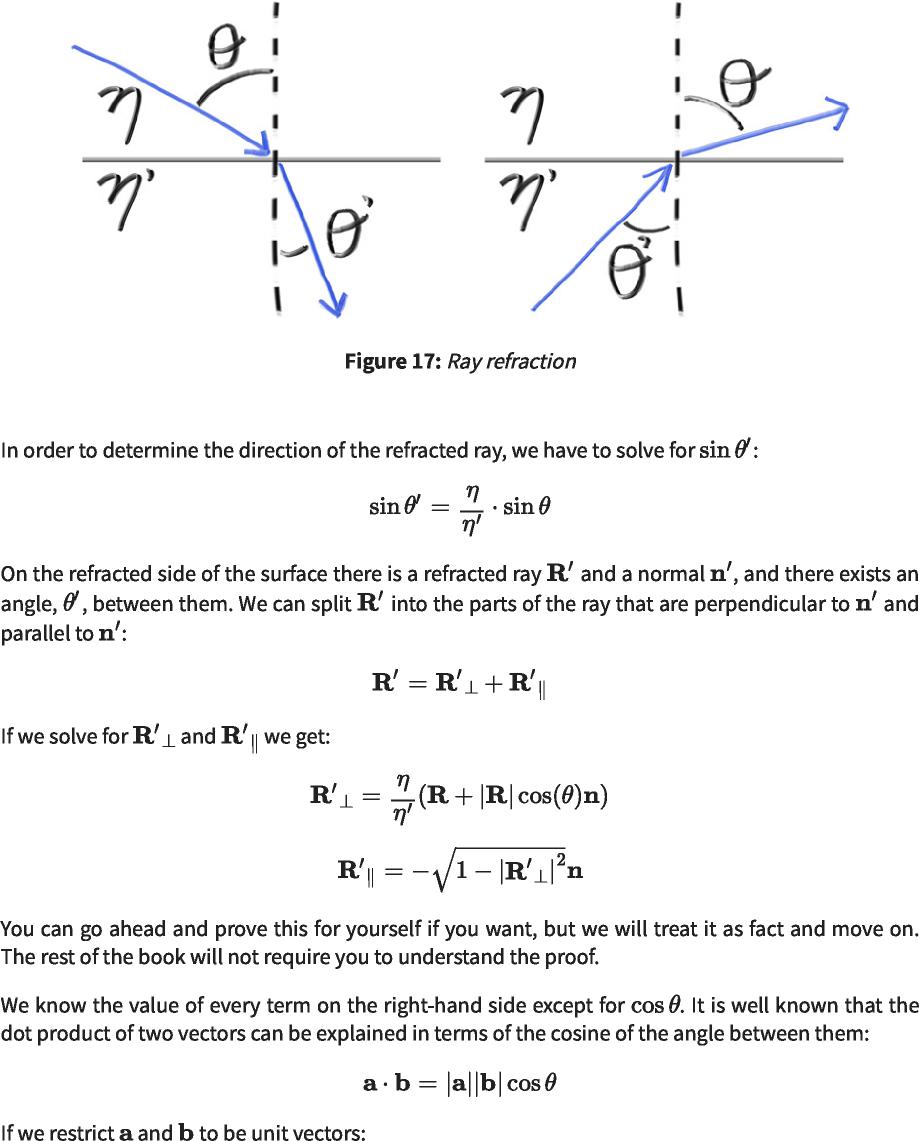


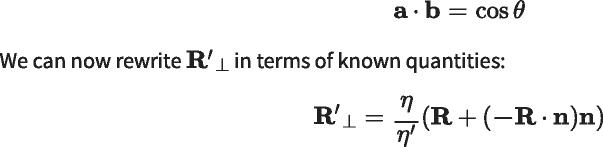
**11.2. Snell's Law**  
 11.2.斯内尔定律



https://raytracing.github.io/books/RayTracingInOneWeekend.html 89/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 89/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

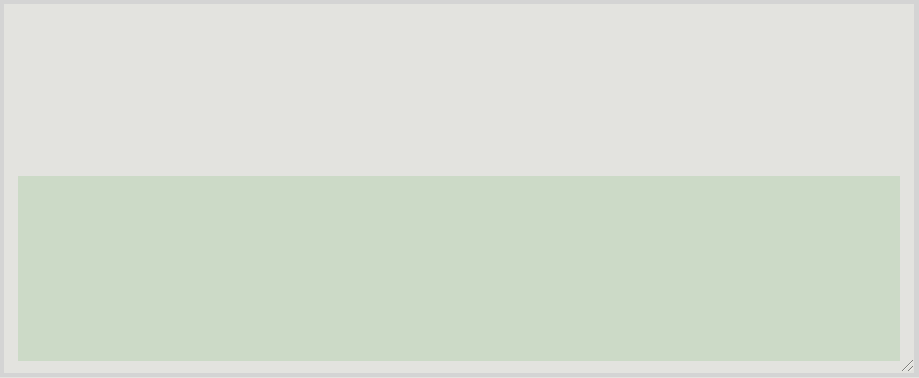




https://raytracing.github.io/books/RayTracingInOneWeekend.html 90/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 90/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





**...**

**inline vec3** reflect**(const vec3& v, const vec3& n) { return v - 2\*dot(v,n)\*n;**  
 inline vec3 reflect(const vec3&v，const vec3&n){return v-2\*dot(v，n)\*n；

**}**

**inline vec3** refract**(const vec3& uv, const vec3& n, double etai\_over\_etat) {**  
 在线vec3折射（const vec3&uv，const vec3&n，double etai\_over\_etat）{

**auto cos\_theta = std::fmin(dot(-uv, n), 1.0);**  
 auto cos\_theta=std：：fmin(dot(-uv，n)，1.0)；

**vec3 r\_out\_perp = etai\_over\_etat \* (uv + cos\_theta\*n);**  
 VEC3R\_OUT\_PERP=etai\_over\_etat\*(uv+COS\_θ\*n)；

**vec3 r\_out\_parallel = -std::sqrt(std::fabs(1.0 - r\_out\_perp.length\_squared())) \***  
 vec3 r\_out\_parallel=-std：：sqrt(std：：fabs(1.0-r\_out\_perp.length\_squared()))\*

**n;**  
 n;

**return r\_out\_perp + r\_out\_parallel;**  
 返回r\_out\_perp+r\_out\_parallel；

**}**



[vec3.h]  
 [vec3.h]



|  |
| --- |
| **...**  **class metal : public material {**  金属类：公共材料{  **...**  **};** |
| **class dielectric : public material {**  介电类：公共材料{ |
| **public:**  公众： |
| **dielectric(double refraction\_index) : refraction\_index(refraction\_index) {}**  电介质(双折射率)：折射率(折射率){} |
| **bool** scatter**(const ray& r\_in, const hit\_record& rec, color& attenuation, ray&**  布尔散射（const ray&r\_in、const hit\_record&rec、color&attenuation、ray&attenuation |
| **scattered)**  分散的) |
| **const override {**  常量覆盖{ |
| **attenuation = color(1.0, 1.0, 1.0);**  衰减=颜色(1.0, 1.0,1.0)； |
| **double ri = rec.front\_face ? (1.0/refraction\_index) : refraction\_index;**  双ri=rec.front\_face？(1.0/refraction\_index)：refraction\_index； |
| **vec3 unit\_direction = unit\_vector(r\_in.direction());**  vec3 unit\_direction=unit\_vector(r\_in.direction())； |
| **vec3 refracted = refract(unit\_direction, rec.normal, ri);**  vec3折射=折射(unit\_direction，rec.normal，ri)； |
| **scattered = ray(rec.p, refracted);**  散射=射线(rec.p，折射)； |
| **return true;**  返回true； |
| **}** |
| **private:**  私人： |
| **// Refractive index in vacuum or air, or the ratio of the material's refractive**  //真空或空气中的折射率，或材料折射率的比值 |
| **index over**  索引结束 |
| **// the refractive index of the enclosing media**  //封闭介质的折射率 |
| **double refraction\_index;**  双折射率； |
| **};** |
|  |

|  |  |  |
| --- | --- | --- |
|  | [material.h]  [材料.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 91/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 91/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

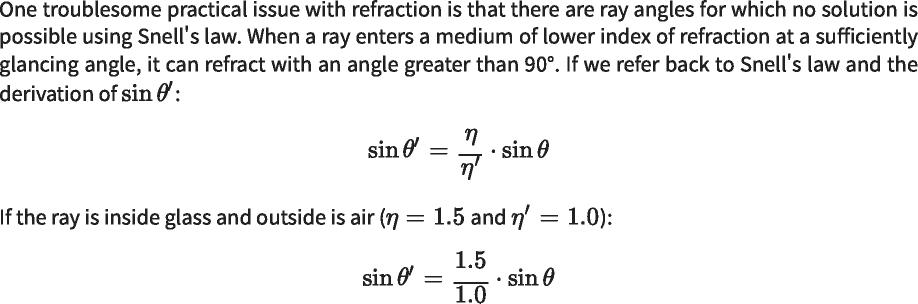


**11.3. Total Internal Reflection**  
 11.3.全内反射



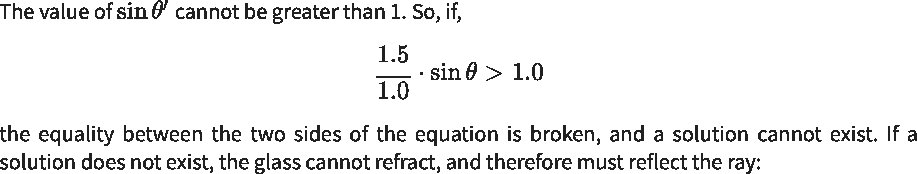
[main.cc]  
 [main.cc]

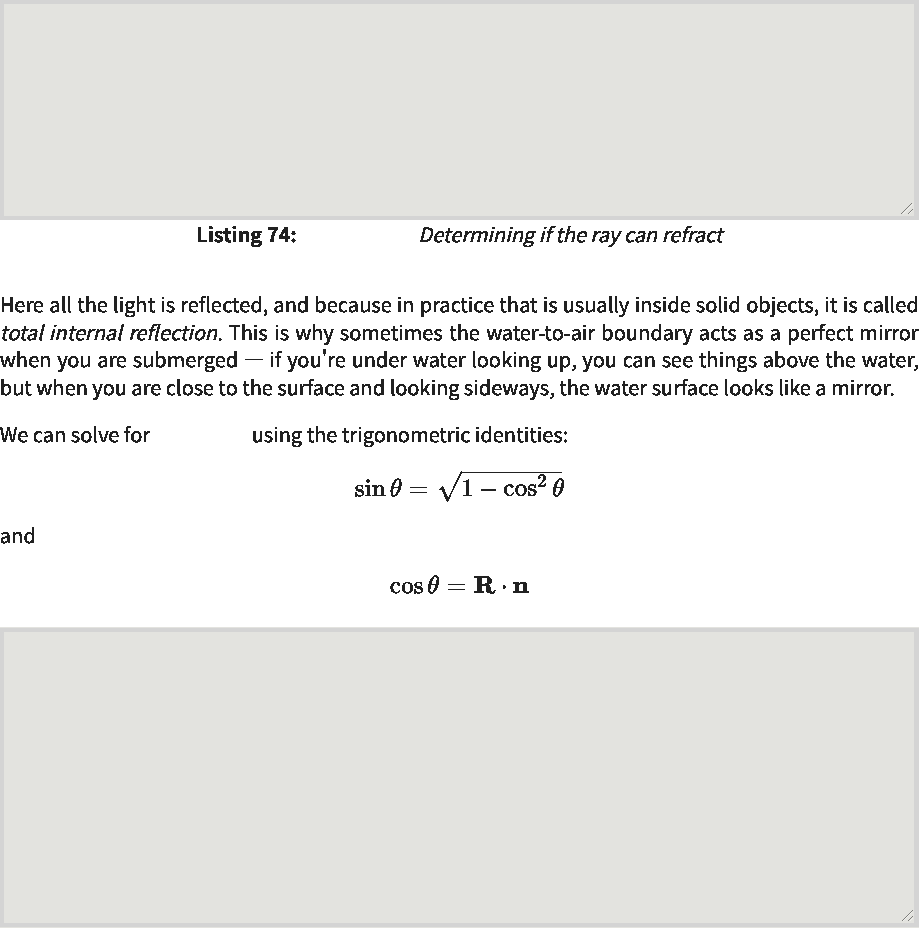
|  |  |  |
| --- | --- | --- |
| auto **material\_ground = make\_shared<lambertian>(color(**0.8**,**  auto material\_ground=make\_shared<lambertian>(color(0.8, | 0.8**,** | 0.0**));** |
| auto **material\_center = make\_shared<lambertian>(color(**0.1**,**  auto material\_center=make\_shared<lambertian>(color(0.1, | 0.2**,** | 0.5**));** |
| auto **material\_left = make\_shared<dielectric>(**1.50**);**  auto material\_left=make\_shared<dielectric>(1.50)； |  |  |
| auto **material\_right = make\_shared<metal>(color(**0.8**,** 0.6**,**  auto material\_right=make\_shared<metal>(color(0.8,0.6, | 0.2**),** | 1.0**);** |



https://raytracing.github.io/books/RayTracingInOneWeekend.html 92/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 92/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





sin\_theta  
 正弦θ

[material.h]  
 [材料.h]

if **(ri \* sin\_theta >** 1.0**) {**  
 如果(ri\*sin\_theta>1.0){

// Must Reflect  
 //必须反映

**...**

**}** else **{**  
 }else{

// Can Refract  
 //可以折射

**...**

**}**

**double cos\_theta = std::fmin(dot(-unit\_direction, rec.normal),** 1.0**); double sin\_theta = std::sqrt(**1.0 **- cos\_theta\*cos\_theta);**  
 double cos\_theta=std：：fmin(dot(-unit\_direction，rec.normal)，1.0)；双sin\_theta=std：：sqrt(1.0-cos\_theta\*cos\_theta)；

if **(ri \* sin\_theta >** 1.0**) {**  
 如果(ri\*sin\_theta>1.0){

// Must Reflect  
 //必须反映

**...**

**}** else **{**  
 }else{

// Can Refract  
 //可以折射

**...**

**}**

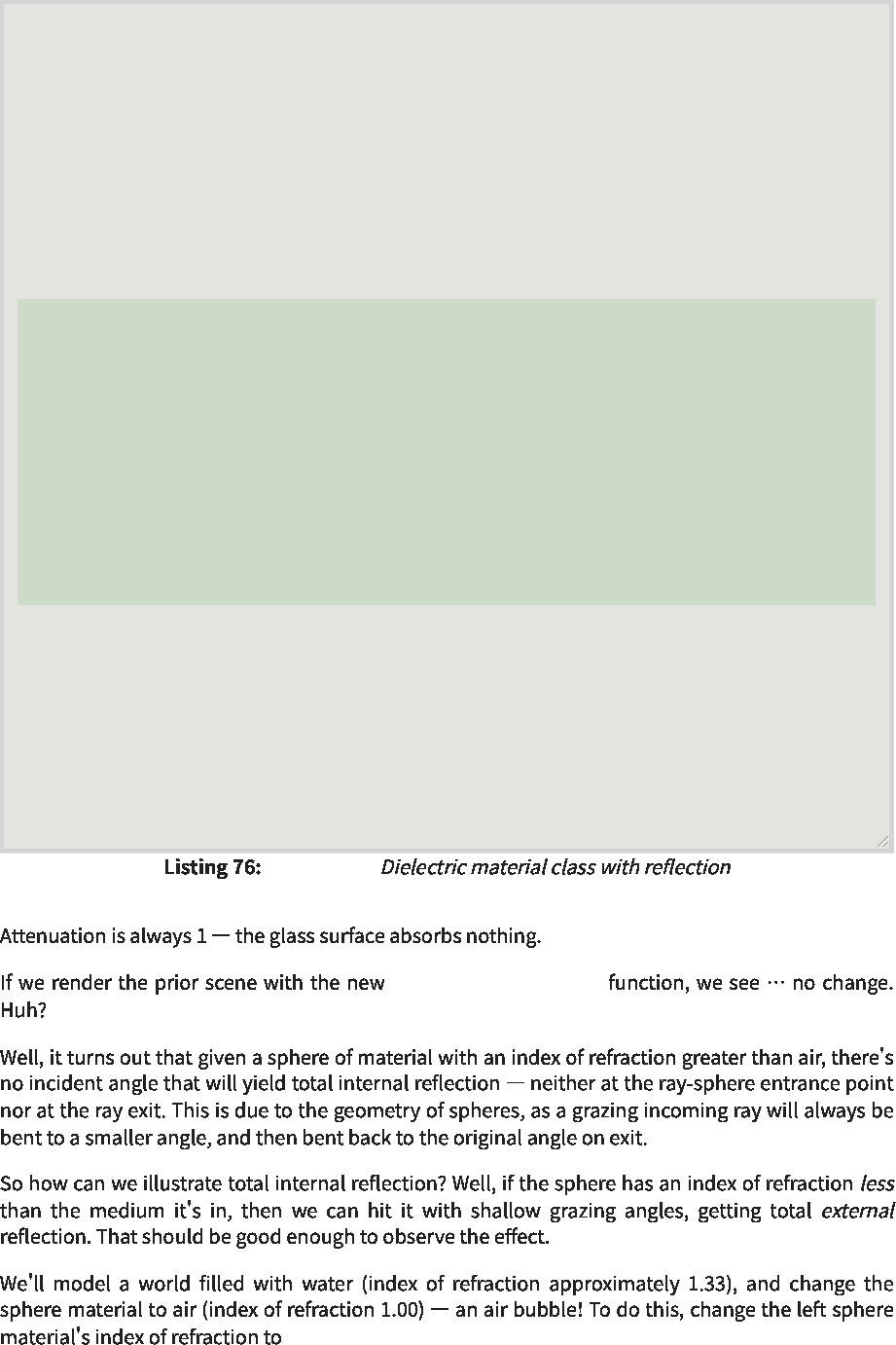
|  |  |  |
| --- | --- | --- |
|  | [material.h]  [材料.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 93/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 93/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



https://raytracing.github.io/books/RayTracingInOneWeekend.html 94/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 94/120



[material.h]  
 [材料.h]

dielectric::scatter()  
 电介质：：散射()

class dielectric : public material {  
 介电类：公共材料{

public:  
 公众：

dielectric(double refraction\_index) : refraction\_index(refraction\_index) {}  
 电介质(双折射率)：折射率(折射率){}

bool scatter(const ray& r\_in, const hit\_record& rec, color& attenuation, ray&  
 布尔散射（const ray&r\_in、const hit\_record&rec、color&attenuation、ray&attenuation

scattered)  
 分散的)

const override {  
 常量覆盖{

attenuation = color(1.0, 1.0, 1.0);  
 衰减=颜色(1.0,1.0,1.0)；

double ri = rec.front\_face ? (1.0/refraction\_index) : refraction\_index;  
 双ri=rec.front\_face？(1.0/refraction\_index)：refraction\_index；

vec3 unit\_direction = unit\_vector(r\_in.direction());  
 vec3 unit\_direction=unit\_vector(r\_in.direction())；

double cos\_theta = std::fmin(dot(-unit\_direction, rec.normal), 1.0); double sin\_theta = std::sqrt(1.0 - cos\_theta\*cos\_theta);  
 double cos\_theta=std：：fmin(dot(-unit\_direction，rec.normal)，1.0)；双sin\_theta=std：：sqrt(1.0-cos\_theta\*cos\_theta)；

bool cannot\_refract = ri \* sin\_theta > 1.0; vec3 direction;  
 bool cannot\_refract=ri\*sin\_theta>1.0；vec3方向；

if (cannot\_refract)  
 如果(cannot\_refract)

direction = reflect(unit\_direction, rec.normal);  
 方向=反射(unit\_direction，rec.normal)；

else  
 else

direction = refract(unit\_direction, rec.normal, ri);  
 方向=折射(unit\_direction，rec.normal，ri)；

scattered = ray(rec.p, direction);  
 散射=射线(rec.p，方向)；

return true;  
 返回true；

}

private:  
 私人：

// Refractive index in vacuum or air, or the ratio of the material's refractive  
 //真空或空气中的折射率，或材料折射率的比值

index over  
 索引结束

// the refractive index of the enclosing media  
 //封闭介质的折射率

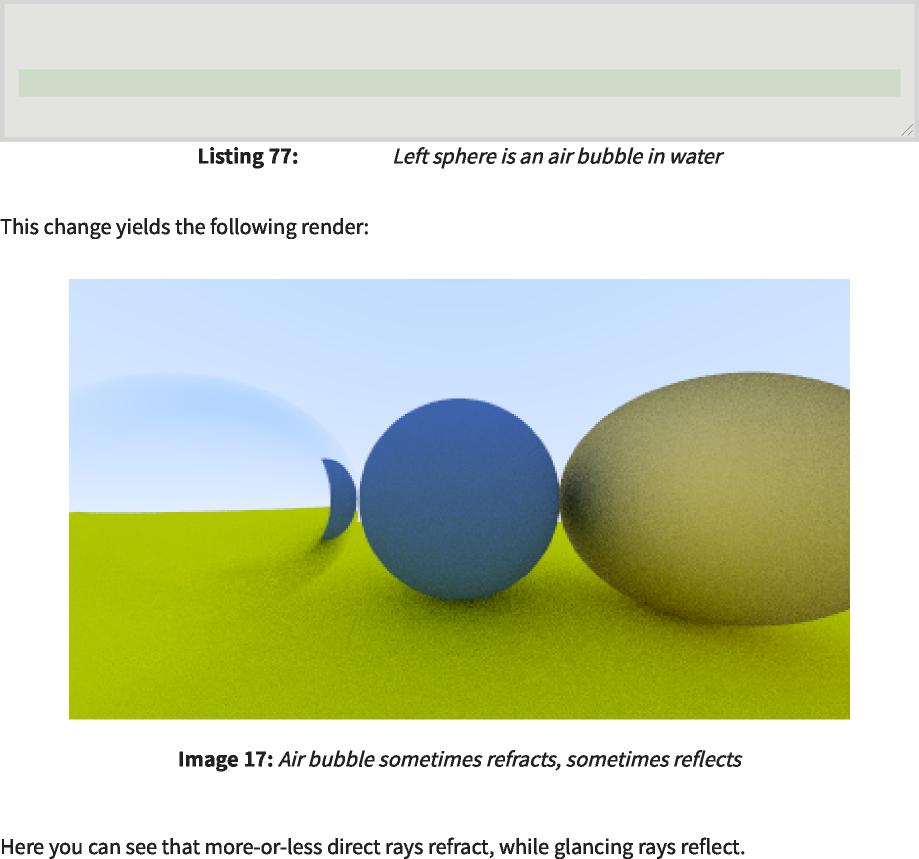
double refraction\_index;  
 双折射率；

};

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**11.4. Schlick Approximation**  
 11.4.Schlick近似



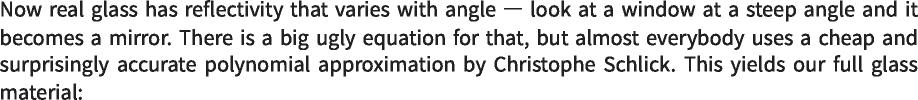
auto **material\_right**  
 auto material\_right

**= make\_shared<metal>(color(**0.8**,** 0.6**,** 0.2**),** 1.0**);**  
 =make\_shared<metal>(颜色(0.8,0.6,0.2),1.0)；

[main.cc]  
 [主抄送]

auto **material\_ground = make\_shared<lambertian>(color(**0.8**,** 0.8**,** 0.0**));** auto **material\_center = make\_shared<lambertian>(color(**0.1**,** 0.2**,** 0.5**));**  
 auto material\_ground=make\_shared<lambertian>(color(0.8,0.8,0.0))；auto material\_center=make\_shared<lambertian>(color(0.1,0.2,0.5))；

auto **material\_left = make\_shared<dielectric>(**1.00 **/** 1.33**);**  
 auto material\_left=make\_shared<dielectric>(1.00/1.33)；



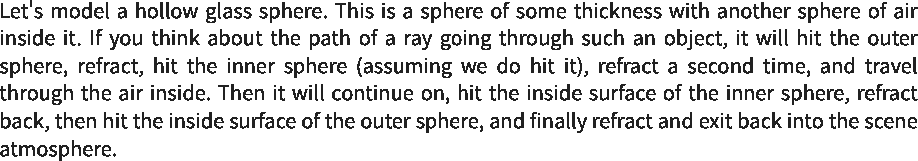
https://raytracing.github.io/books/RayTracingInOneWeekend.html 95/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 95/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |
| --- |
| class dielectric : public material {  介电类：公共材料{  public:  公众：  dielectric(double refraction\_index) : refraction\_index(refraction\_index) {}  电介质(双折射率)：折射率(折射率){}  bool scatter(const ray& r\_in, const hit\_record& rec, color& attenuation, ray&  布尔散射（const ray&r\_in、const hit\_record&rec、color&attenuation、ray&attenuation  scattered)  分散的)  const override {  常量覆盖{  attenuation = color(1.0, 1.0, 1.0);  衰减=颜色(1.0, 1.0,1.0)；  double ri = rec.front\_face ? (1.0/refraction\_index) : refraction\_index;  双ri=rec.front\_face？(1.0/refraction\_index)：refraction\_index；  vec3 unit\_direction = unit\_vector(r\_in.direction());  vec3 unit\_direction=unit\_vector(r\_in.direction())；  double cos\_theta = std::fmin(dot(-unit\_direction, rec.normal), 1.0);  double cos\_theta=std：：fmin(dot(-unit\_direction，rec.normal)，1.0)；  double sin\_theta = std::sqrt(1.0 - cos\_theta\*cos\_theta);  双sin\_theta=std：：sqrt(1.0-cos\_theta\*cos\_theta)；  bool cannot\_refract = ri \* sin\_theta > 1.0;  vec3 direction;  bool cannot\_refract=ri\*sin\_theta>1.0； vec3方向； |
| if (cannot\_refract || reflectance(cos\_theta, ri) > random\_double())  如果(cannot\_refract反射率(cos\_theta，ri)>random\_double()) |
| direction = reflect(unit\_direction, rec.normal);  方向=反射(unit\_direction，rec.normal)；  else  else  direction = refract(unit\_direction, rec.normal, ri);  方向=折射(unit\_direction，rec.normal，ri)；  scattered = ray(rec.p, direction);  散射=射线(rec.p，方向)；  return true;  返回true；  }  private:  私人：  // Refractive index in vacuum or air, or the ratio of the material's refractive  //真空或空气中的折射率，或材料折射率的比值  index over  索引结束  // the refractive index of the enclosing media  //封闭介质的折射率  double refraction\_index;  双折射率； |
| static double reflectance(double cosine, double refraction\_index) {  静态双反射率(双余弦，双折射率){  // Use Schlick's approximation for reflectance.  //对反射率使用Schlick近似。  auto r0 = (1 - refraction\_index) / (1 + refraction\_index);  auto r0=(1-refraction\_index)/(1+refraction\_index)；  r0 = r0\*r0;  r0=r0\*r0；  return r0 + (1-r0)\*std::pow((1 - cosine),5);  返回r0+(1-r0)\*std：：pow((1-余弦),5)；  } |
| }; |

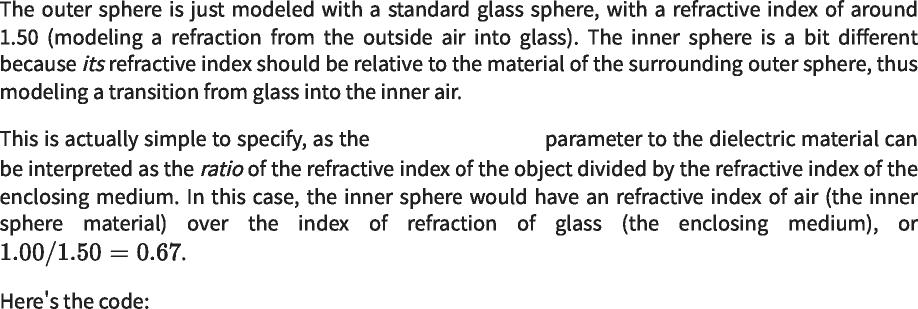
|  |  |  |
| --- | --- | --- |
|  | [material.h]  [材料.h] |  |

**11.5. Modeling a Hollow Glass Sphere**  
 11.5.中空玻璃球建模



https://raytracing.github.io/books/RayTracingInOneWeekend.html 96/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 96/120

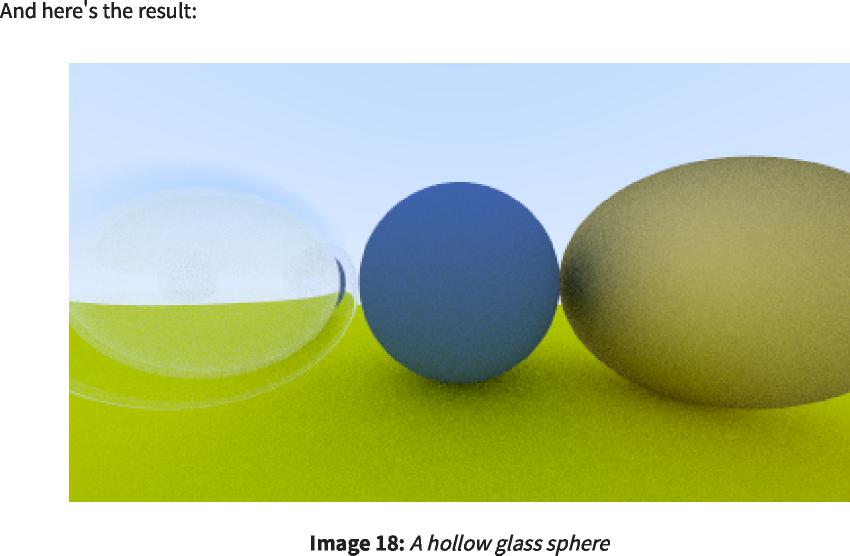
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



refraction\_index  
 折射率

|  |  |  |
| --- | --- | --- |
| **...**  auto **material\_ground = make\_shared<lambertian>(color(**0.8**,**  auto material\_ground=make\_shared<lambertian>(color(0.8,  auto **material\_center = make\_shared<lambertian>(color(**0.1**,**  auto material\_center=make\_shared<lambertian>(color(0.1, | 0.8**,** 0.0**));**  0.2**,** 0.5**));** | |
| auto **material\_left = make\_shared<dielectric>(**1.50**);**  auto material\_left=make\_shared<dielectric>(1.50)； |  |  |
| auto **material\_bubble = make\_shared<dielectric>(**1.00 **/** 1.50**);**  auto material\_bubble=make\_shared<dielectric>(1.00/1.50)； | |  |
| auto **material\_right = make\_shared<metal>(color(**0.8**,** 0.6**,**  auto material\_right=make\_shared<metal>(color(0.8,0.6, | 0.2**),** | 0.0**);** |
| **world.add(make\_shared<sphere>(point3(** 0.0**,** -100.5**,** -1.0**),**  world.add(make\_shared<sphere>(point3(0.0,-100.5,-1.0), | 100.0**,** | **material\_ground));**  material\_ground)); |
| **world.add(make\_shared<sphere>(point3(** 0.0**,** 0.0**,** -1.2**),**  world.add(make\_shared<sphere>(point3(0.0,0.0,-1.2), | 0.5**,** | **material\_center));**  material\_center)); |
| **world.add(make\_shared<sphere>(point3(**-1.0**,** 0.0**,** -1.0**),**  world.add(make\_shared<sphere>(point3(-1.0, 0.0,-1.0), | 0.5**,** | **material\_left));**  material\_left)); |
| **world.add(make\_shared<sphere>(point3(**-1.0**,** 0.0**,** -1.0**),**  world.add(make\_shared<sphere>(point3(-1.0, 0.0,-1.0), | 0.4**,** | **material\_bubble));**  material\_bubble)); |
| **world.add(make\_shared<sphere>(point3(** 1.0**,** 0.0**,** -1.0**),**  world.add(make\_shared<sphere>(point3(1.0,0.0,-1.0), | 0.5**,** | **material\_right));**  material\_right)); |
| **...** |  |  |

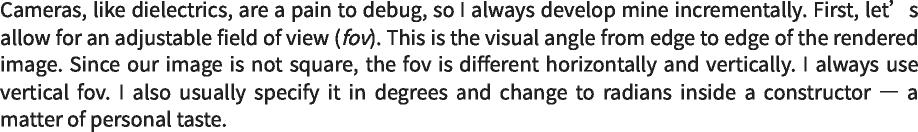
|  |  |  |
| --- | --- | --- |
|  | [main.cc]  [main.cc] |  |



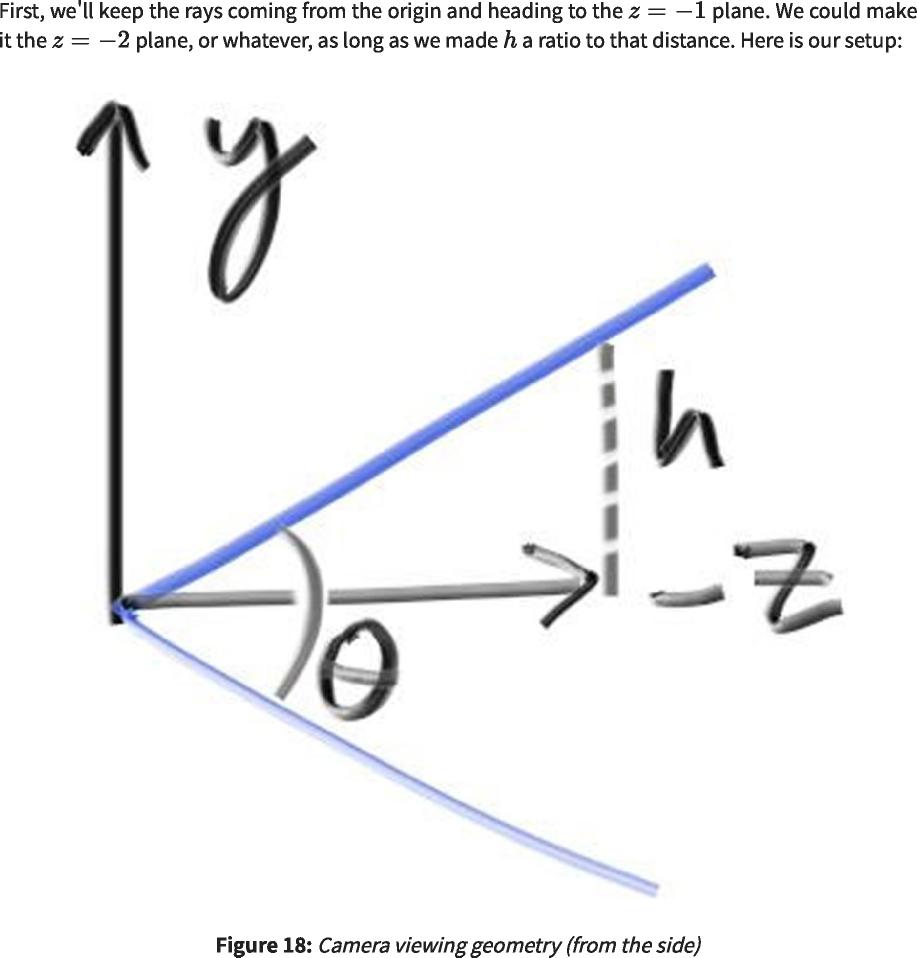
https://raytracing.github.io/books/RayTracingInOneWeekend.html 97/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 97/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**12. Positionable Camera**  
 12.可定位摄像机



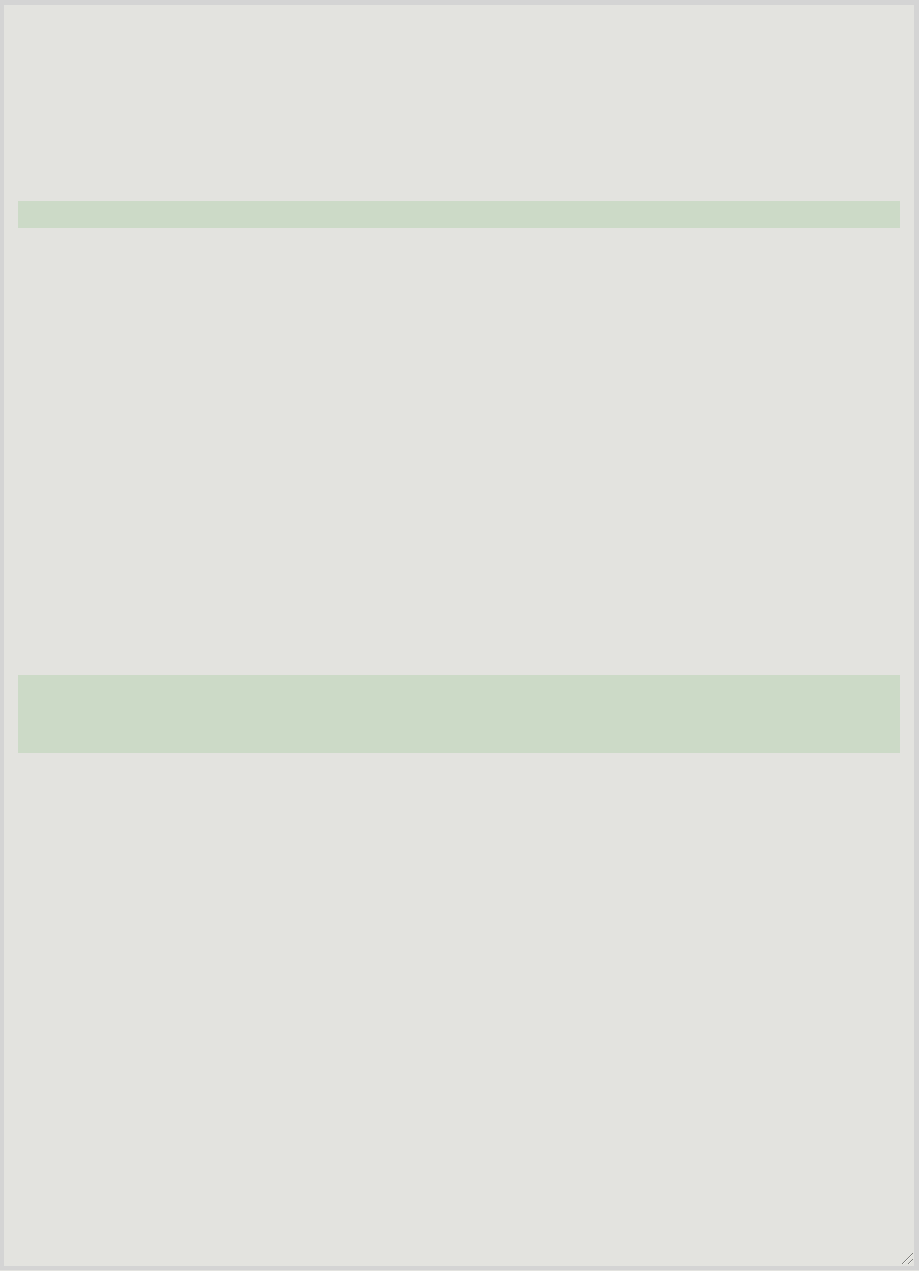
**12.1. Camera Viewing Geometry**  
 12.1.摄像机观察几何



https://raytracing.github.io/books/RayTracingInOneWeekend.html 98/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 98/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





class camera {  
 类相机{

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| public:  公众： |  |  |  |  |  |
| double  double | aspect\_ratio  纵横比 | = | 1.0; | // | Ratio of image width over height  图像宽高比 |
| int  int | image\_width  图像宽度 | = | 100; | // | Rendered image width in pixel count  渲染图像宽度（以像素计） |
| int  int | samples\_per\_pixel  每像素样本数 | = | 10; | // | Count of random samples for each pixel  每个像素的随机样本计数 |
| int  int | max\_depth  最大深度 | = | 10; | // Maximum number of ray bounces into scene  //反射到场景中的最大光线数 | |

double vfov = 90; // Vertical view angle (field of view)  
 双vfov=90；//垂直视角(视场)

void render(const hittable& world) {  
 void render(const hittable&world){

...

private:  
 私人：

...

void initialize() {  
 void initialize(){

image\_height = int(image\_width / aspect\_ratio);  
 image\_height=int(image\_width/aspect\_ratio)；

image\_height = (image\_height < 1) ? 1 : image\_height;  
 image\_height=(image\_height<1)？1：image\_height；

pixel\_samples\_scale = 1.0 / samples\_per\_pixel; center = point3(0, 0, 0);  
 pixel\_samples\_scale=1.0/samples\_per\_pixel；中心=点3(0,0,0)；

// Determine viewport dimensions.   
auto focal\_length = 1.0;  
 //确定视口尺寸。 auto focal\_length=1.0；

auto theta = degrees\_to\_radians(vfov);  
 自动θ=度数到弧度(vfov)；

auto h = std::tan(theta/2);  
 auto h=std：：tan(θ/2)；

auto viewport\_height = 2 \* h \* focal\_length;  
 auto viewport\_height=2\*h\*focal\_length；

auto viewport\_width = viewport\_height \* (double(image\_width)/image\_height);  
 auto viewport\_width=viewport\_height\*(double(image\_width)/image\_height)；

// Calculate the vectors across the horizontal and down the vertical viewport  
 //计算水平视口和垂直视口向下的向量

edges.  
 边缘。

auto viewport\_u = vec3(viewport\_width, 0, 0); auto viewport\_v = vec3(0, -viewport\_height, 0);  
 auto viewport\_u=vec3(viewport\_width,0,0)；auto viewport\_v=vec3(0,-viewport\_height,0)；

// Calculate the horizontal and vertical delta vectors from pixel to pixel.  
 //计算从像素到像素的水平和垂直增量向量。

pixel\_delta\_u = viewport\_u / image\_width;  
 pixel\_delta\_u=viewport\_u/image\_width；

pixel\_delta\_v = viewport\_v / image\_height;  
 pixel\_delta\_v=viewport\_v/image\_height；

// Calculate the location of the upper left pixel.  
 //计算左上像素的位置。

auto viewport\_upper\_left =  
 auto viewport\_upper\_left=

center - vec3(0, 0, focal\_length) - viewport\_u/2 - viewport\_v/2;  
 center-vec3(0,0,focal\_length)-viewport\_u/2-viewport\_v/2；

pixel00\_loc = viewport\_upper\_left + 0.5 \* (pixel\_delta\_u + pixel\_delta\_v);  
 pixel00\_loc=viewport\_upper\_left+0.5\*(pixel\_delta\_u+pixel\_delta\_v)；

}

...

};

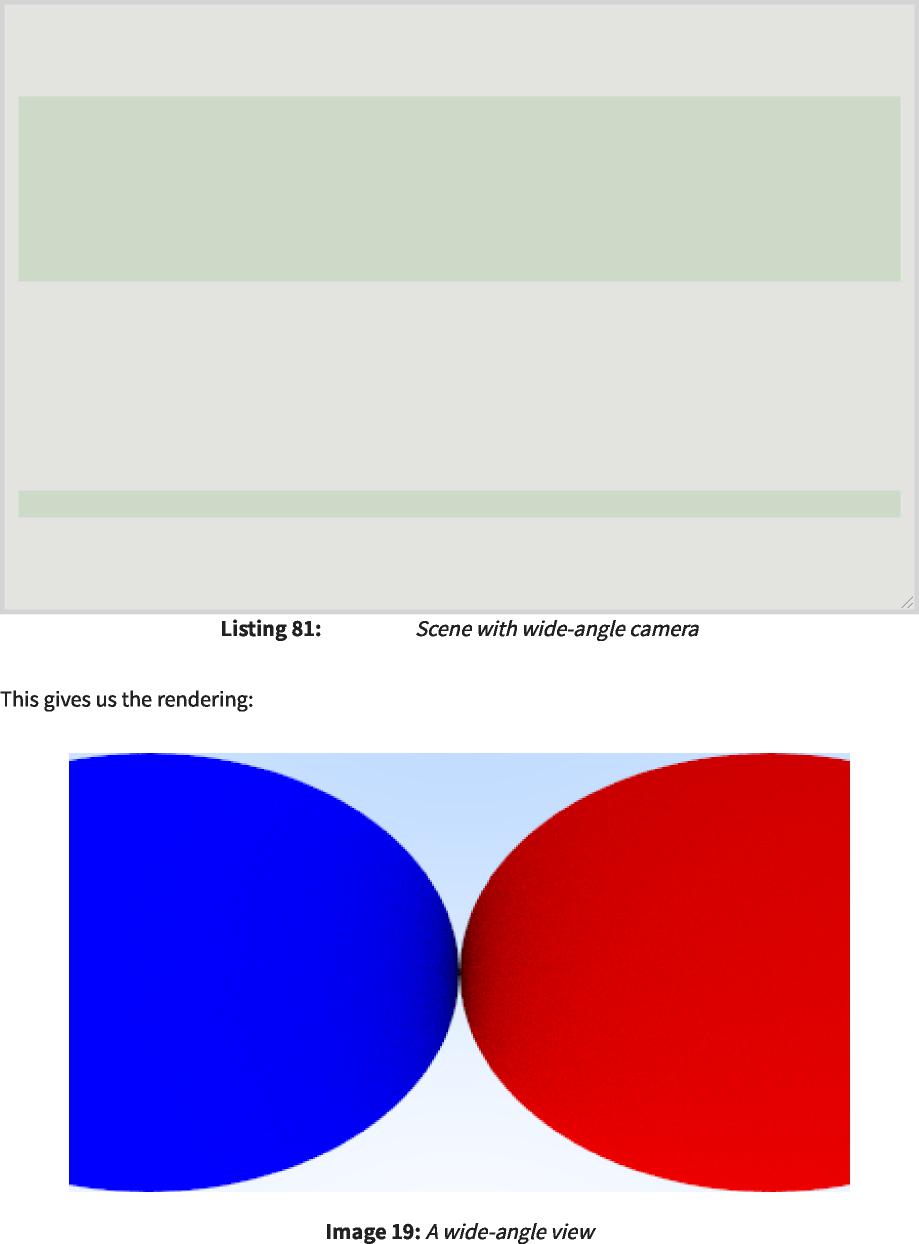
|  |  |  |
| --- | --- | --- |
|  | [camera.h]  [摄像机.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 99/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 99/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



https://raytracing.github.io/books/RayTracingInOneWeekend.html 100/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 100/120



[main.cc]  
 [main.cc]

int main() {  
 int main(){

hittable\_list world;  
 hittable\_list world；

auto R = std::cos(pi/4);  
 auto R=std：：cos(pi/4)；

auto material\_left = make\_shared<lambertian>(color(0,0,1)); auto material\_right = make\_shared<lambertian>(color(1,0,0));  
 auto material\_left=make\_shared<lambertian>(color(0,0,1))；auto material\_right=make\_shared<lambertian>(color(1,0,0))；

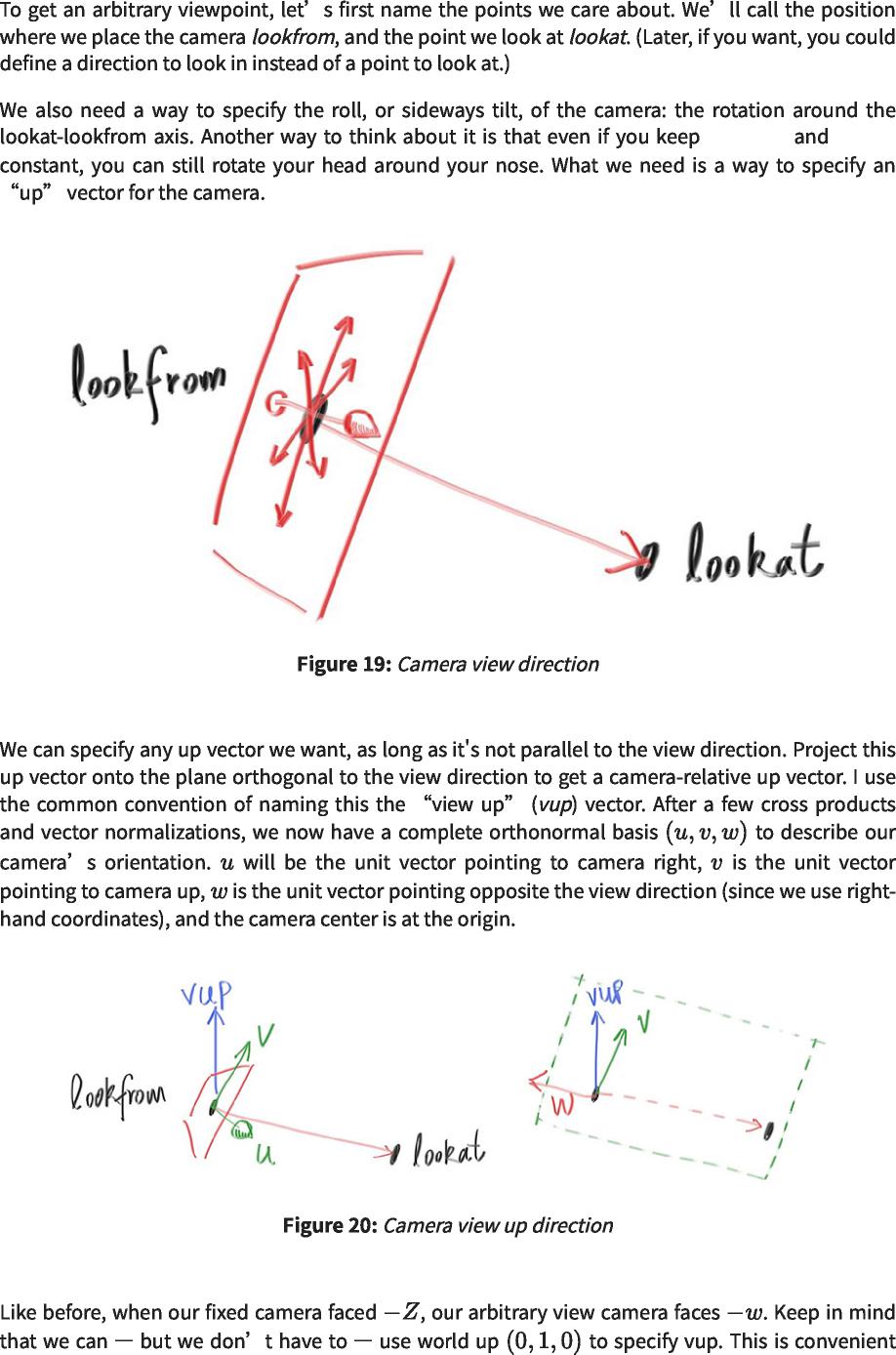
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| world.add(make\_shared<sphere>(point3(-R, 0, -1), R, material\_left));  world.add(make\_shared<sphere>(point3(-R,0, -1)，R，material\_left))；  world.add(make\_shared<sphere>(point3( R, 0, -1), R, material\_right));  world.add(make\_shared<sphere>(point3(R,0, -1)，R，material\_right))； | | | | |
| camera cam;  照相机凸轮； |  |  |  |  |
| cam.aspect\_ratio  凸轮纵横比 | = | 16.0 | / | 9.0; |
| cam.image\_width  cam.image\_width | = | 400; |  |  |
| cam.samples\_per\_pixel  cam.samples\_per\_pixel | = | 100; |  |  |
| cam.max\_depth  凸轮最大深度 | = | 50; |  |  |
| cam.vfov = 90;  cam.vfov=90； |  |  |  |  |
| cam.render(world);  凸轮。渲染（世界）； |  |  |  |  |

}

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**12.2. Positioning and Orienting the Camera**  
 12.2.定位和定向摄像机

https://raytracing.github.io/books/RayTracingInOneWeekend.html 101/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 101/120



lookfrom lookat  
 lookfromlookat

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



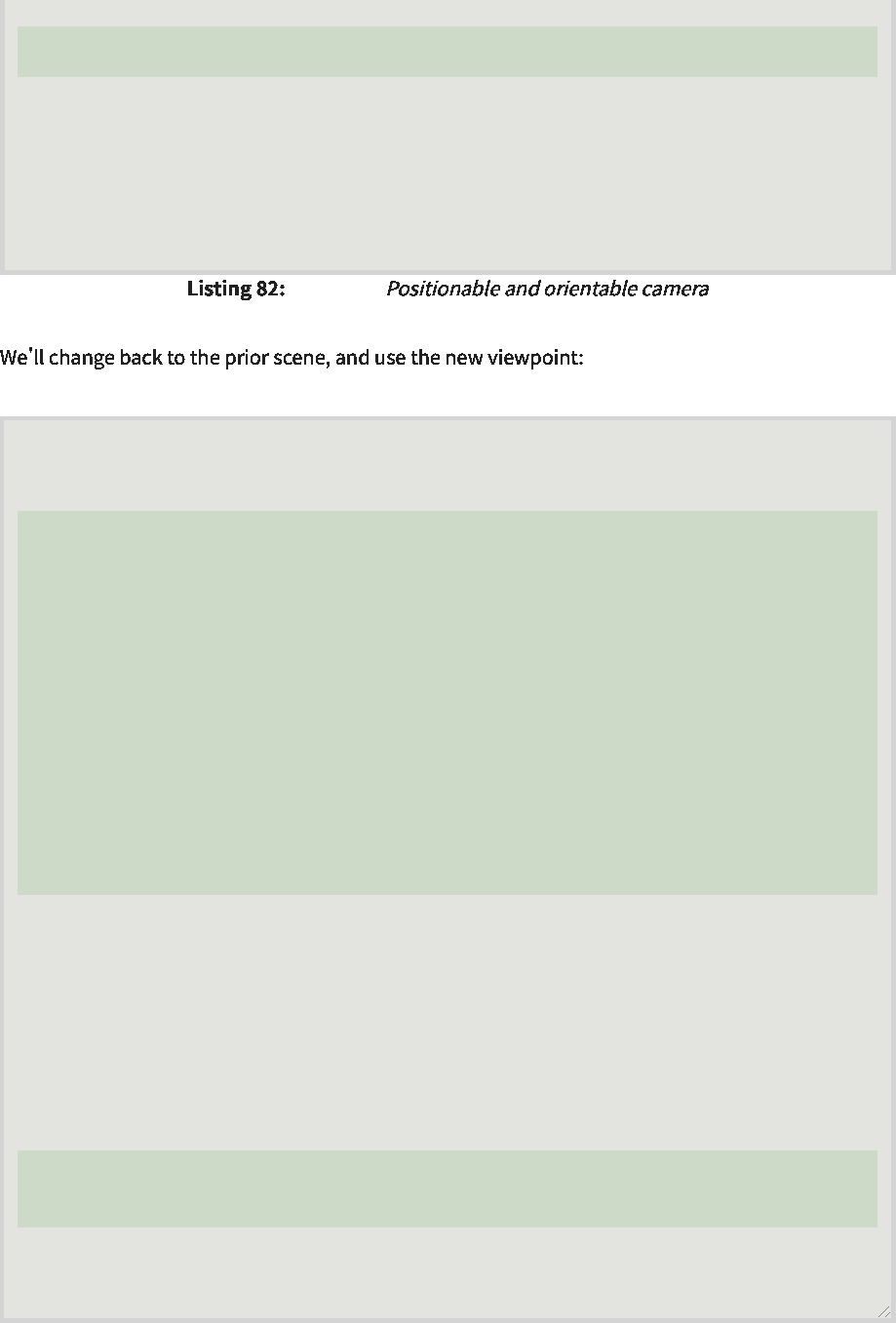
https://raytracing.github.io/books/RayTracingInOneWeekend.html 102/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 102/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |
| --- |
| class camera {  public:  类相机{ 公众：  double aspect\_ratio = 1.0; // Ratio of image width over height  双纵横比=1.0； //图像宽高比  int image\_width = 100; // Rendered image width in pixel count  intimage\_width=100； //以像素计数表示的渲染图像宽度  int samples\_per\_pixel = 10; // Count of random samples for each pixel  intsamples\_per\_pixel=10； //每个像素的随机样本计数  int max\_depth = 10; // Maximum number of ray bounces into scene  intmax\_depth=10； //反射到场景中的最大光线数  double vfov = 90; // Vertical view angle (field of view)  双vfov=90； //垂直视角(视场) |
| point3 lookfrom = point3(0,0,0); // Point camera is looking from  point3 lookfrom=point3(0,0,0)； //点摄像机从  point3 lookat = point3(0,0,-1); // Point camera is looking at  点3 lookat=点3(0,0,-1)； //点摄像机正在查看  vec3 vup = vec3(0,1,0); // Camera-relative 'up' direction  vec3vup=vec3(0,1,0)； //相机相对“向上”方向 |
| ...  private:  私人：  int image\_height; // Rendered image height  intimage\_height； //渲染图像高度  double pixel\_samples\_scale; // Color scale factor for a sum of pixel samples  双pixel\_samples\_scale； //像素样本总和的色标因子  point3 center; // Camera center  点3中心； //相机中心  point3 pixel00\_loc; // Location of pixel 0, 0  点3 pixel00\_loc； //像素0,0的位置  vec3 pixel\_delta\_u; // Offset to pixel to the right  vec3pixel\_delta\_u； //向右偏移到像素  vec3 pixel\_delta\_v; // Offset to pixel below  vec3pixel\_delta\_v； //偏移量到下面的像素 |
| vec3 u, v, w; // Camera frame basis vectors  vec3u，v，w； //相机帧基向量 |
| void **initialize**() {  void initialize(){  image\_height = int(image\_width / aspect\_ratio);  image\_height=int(image\_width/aspect\_ratio)；  image\_height = (image\_height < 1) ? 1 : image\_height;  image\_height=(image\_height<1)？1：image\_height；  pixel\_samples\_scale = 1.0 / samples\_per\_pixel;  pixel\_samples\_scale=1.0/samples\_per\_pixel； |
| center = lookfrom;  center=lookfrom； |
| // Determine viewport dimensions.  //确定视口尺寸。 |
| auto focal\_length = (lookfrom - lookat).length();  auto focal\_length=(lookfrom-lookat).length()； |
| auto theta = degrees\_to\_radians(vfov);  自动θ=度数到弧度(vfov)；  auto h = std::tan(theta/2);  auto h=std：：tan(θ/2)；  auto viewport\_height = 2 \* h \* focal\_length;  auto viewport\_height=2\*h\*focal\_length；  auto viewport\_width = viewport\_height \* (double(image\_width)/image\_height);  auto viewport\_width=viewport\_height\*(double(image\_width)/image\_height)； |
| // Calculate the u,v,w unit basis vectors for the camera coordinate frame.  //计算相机坐标系的u、v、w单位基向量。  w = unit\_vector(lookfrom - lookat);  w=unit\_vector(lookfrom-lookat)；  u = unit\_vector(cross(vup, w));  u=unit\_vector(cross(vup，w))；  v = cross(w, u);  v=十字(w，u)； |
| // Calculate the vectors across the horizontal and down the vertical viewport edges.  //计算水平和垂直视口边缘的向量。 |
| vec3 viewport\_u = viewport\_width \* u; // Vector across viewport horizontal edge  vec3 viewport\_u=viewport\_width\*u； //跨视口水平边缘的矢量  vec3 viewport\_v = viewport\_height \* -v; // Vector down viewport vertical edge  vec3 viewport\_v=viewport\_height\*-v； //向下向视口垂直边缘 |
| // Calculate the horizontal and vertical delta vectors from pixel to pixel.  //计算从像素到像素的水平和垂直增量向量。  pixel\_delta\_u = viewport\_u / image\_width;  pixel\_delta\_u=viewport\_u/image\_width；  pixel\_delta\_v = viewport\_v / image\_height;  pixel\_delta\_v=viewport\_v/image\_height； |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 103/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 103/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



[camera.h]  
 [摄像机.h]

// Calculate the location of the upper left pixel.  
 //计算左上像素的位置。

auto viewport\_upper\_left = center - (focal\_length \* w) - viewport\_u/2 - viewport\_v/2;  
 auto viewport\_upper\_left=center-(focal\_length\*w)-viewport\_u/2-viewport\_v/2；

pixel00\_loc = viewport\_upper\_left + 0.5 \* (pixel\_delta\_u + pixel\_delta\_v);  
 pixel00\_loc=viewport\_upper\_left+0.5\*(pixel\_delta\_u+pixel\_delta\_v)；

}

...

private:  
 私人：

};

int main() {  
 int main(){

hittable\_list world;  
 hittable\_list world；

auto material\_ground = make\_shared<lambertian>(color(0.8, 0.8, 0.0)); auto material\_center = make\_shared<lambertian>(color(0.1, 0.2, 0.5));  
 auto material\_ground=make\_shared<lambertian>(color(0.8,0.8,0.0))；auto material\_center=make\_shared<lambertian>(color(0.1,0.2,0.5))；

auto material\_left = make\_shared<dielectric>(1.50);  
 auto material\_left=make\_shared<dielectric>(1.50)；

auto material\_bubble = make\_shared<dielectric>(1.00 / 1.50);  
 auto material\_bubble=make\_shared<dielectric>(1.00/1.50)；

auto material\_right = make\_shared<metal>(color(0.8, 0.6, 0.2), 1.0);  
 auto material\_right=make\_shared<metal>(color(0.8,0.6,0.2),1.0)；

world.add(make\_shared<sphere>(point3( 0.0, -100.5, -1.0), 100.0, material\_ground));  
 world.add(make\_shared<sphere>(point3(0.0,-100.5,-1.0),100.0,material\_ground))；

world.add(make\_shared<sphere>(point3( 0.0, 0.0, -1.2), 0.5,   
material\_center));  
 world.add(make\_shared<sphere>(point3(0.0,0.0,-1.2),0.5, material\_center));

world.add(make\_shared<sphere>(point3(-1.0, 0.0, -1.0), 0.5, material\_left));  
 world.add(make\_shared<sphere>(point3(-1.0, 0.0,-1.0),0.5,material\_left))；

world.add(make\_shared<sphere>(point3(-1.0, 0.0, -1.0), 0.4,  
 world.add(make\_shared<sphere>(point3(-1.0, 0.0,-1.0),0.4,

material\_bubble));  
 material\_bubble));

world.add(make\_shared<sphere>(point3( 1.0, 0.0, -1.0), 0.5,   
material\_right));  
 world.add(make\_shared<sphere>(point3(1.0,0.0,-1.0),0.5, material\_right));

camera cam;  
 照相机凸轮；

cam.aspect\_ratio = 16.0 / 9.0;  
 cam.aspect\_ratio=16.0/9.0；

cam.image\_width = 400; cam.samples\_per\_pixel = 100;  
 cam.image\_width=400；cam.samples\_per\_pixel=100；

cam.max\_depth = 50;  
 cam.max\_depth=50；

cam.vfov = 90;  
 cam.vfov=90；

cam.lookfrom = point3(-2,2,1);  
 凸轮。look from=point 3（-2，2，1）；

cam.lookat = point3(0,0,-1);  
 cam.lookat=point3(0,0,-1)；

cam.vup = vec3(0,1,0);  
 cam.vup=vec3(0,1,0)；

cam.render(world);  
 凸轮。渲染（世界）；

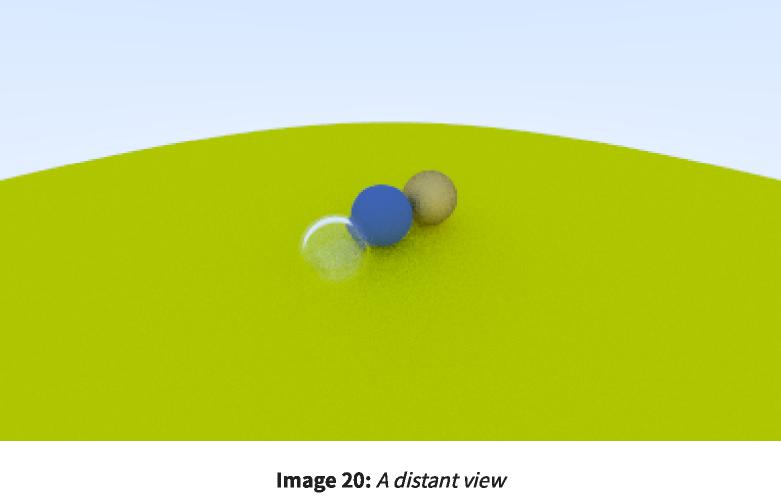
}

|  |  |  |
| --- | --- | --- |
|  | [main.cc]  [main.cc] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 104/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 104/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪





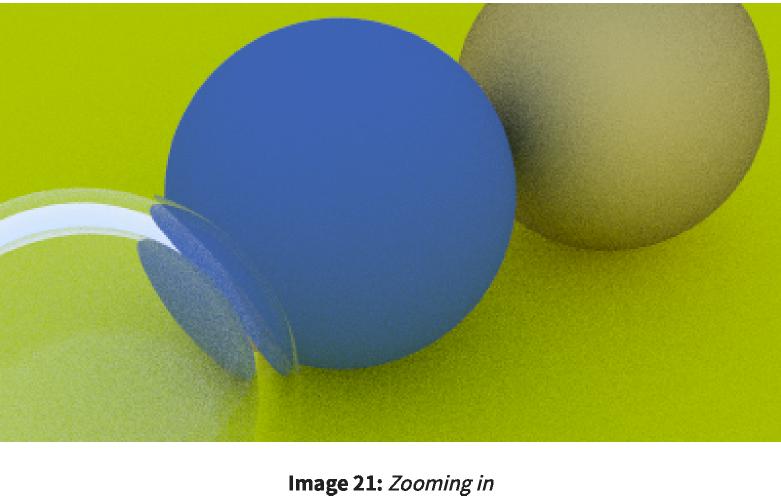


cam.vfov = 20;  
 cam.vfov=20；



|  |  |  |
| --- | --- | --- |
|  | [main.cc]  [main.cc] |  |

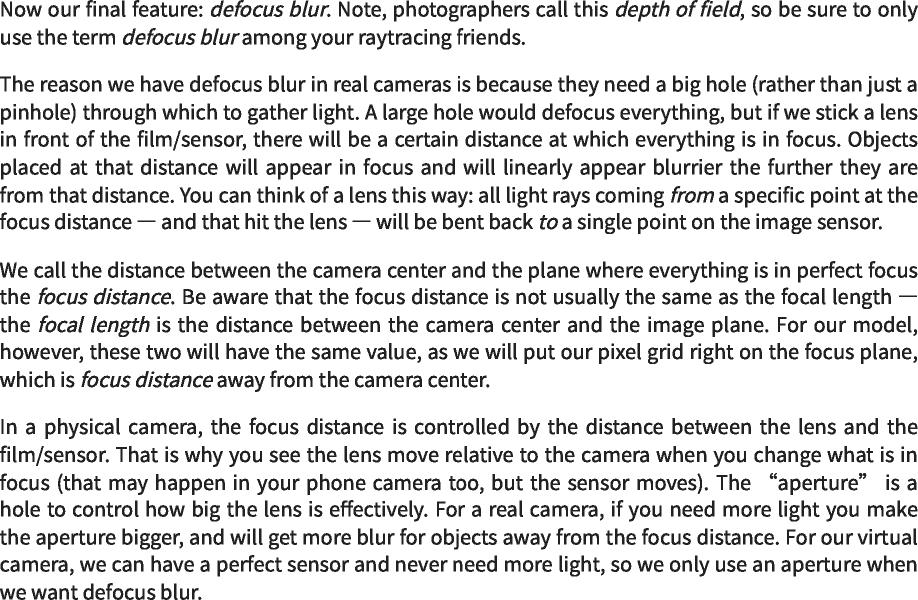




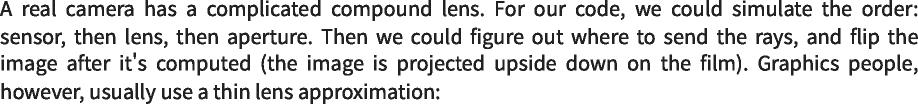
https://raytracing.github.io/books/RayTracingInOneWeekend.html 105/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 105/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**13. Defocus Blur**  
 13.散焦模糊

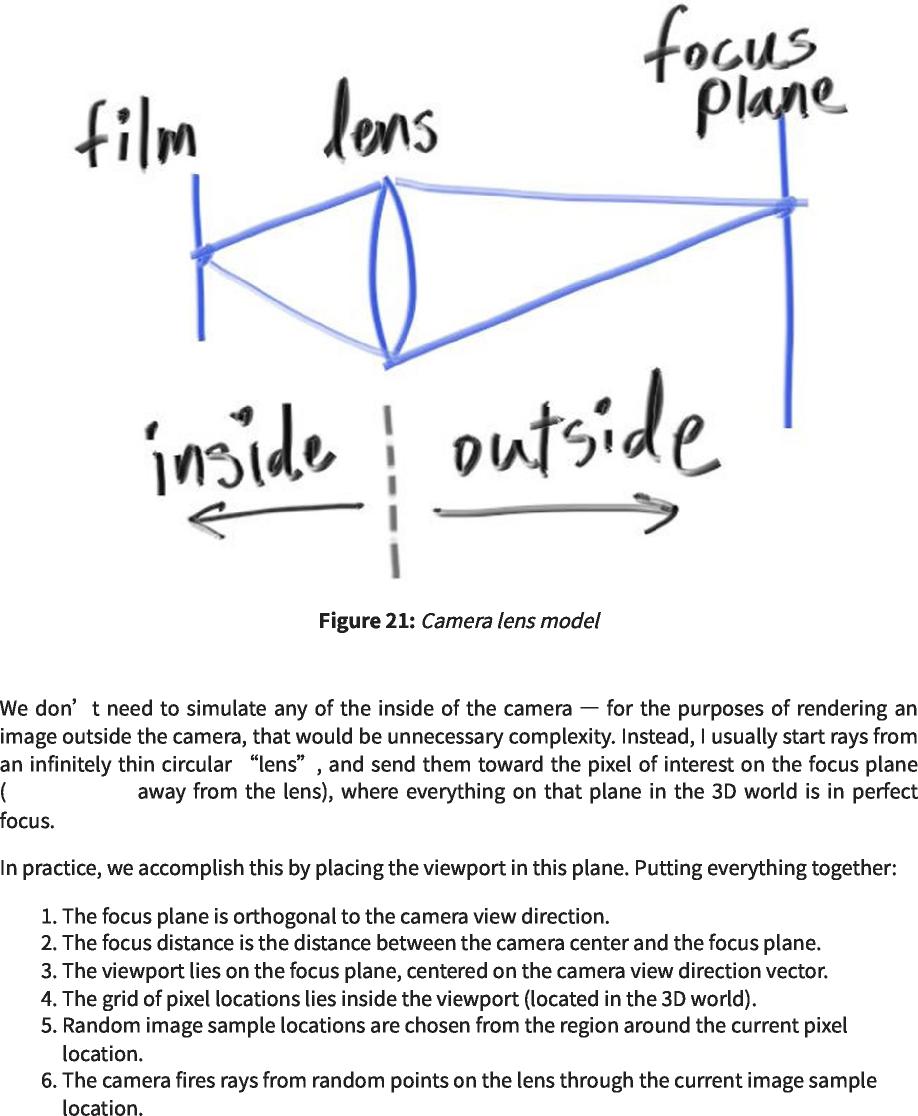


**13.1. A Thin Lens Approximation**  
 13.1.薄透镜近似



https://raytracing.github.io/books/RayTracingInOneWeekend.html 106/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 106/120

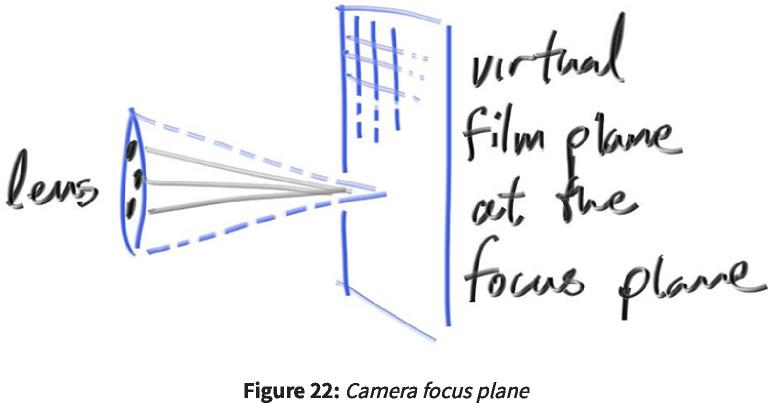
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



focal\_length  
 焦距

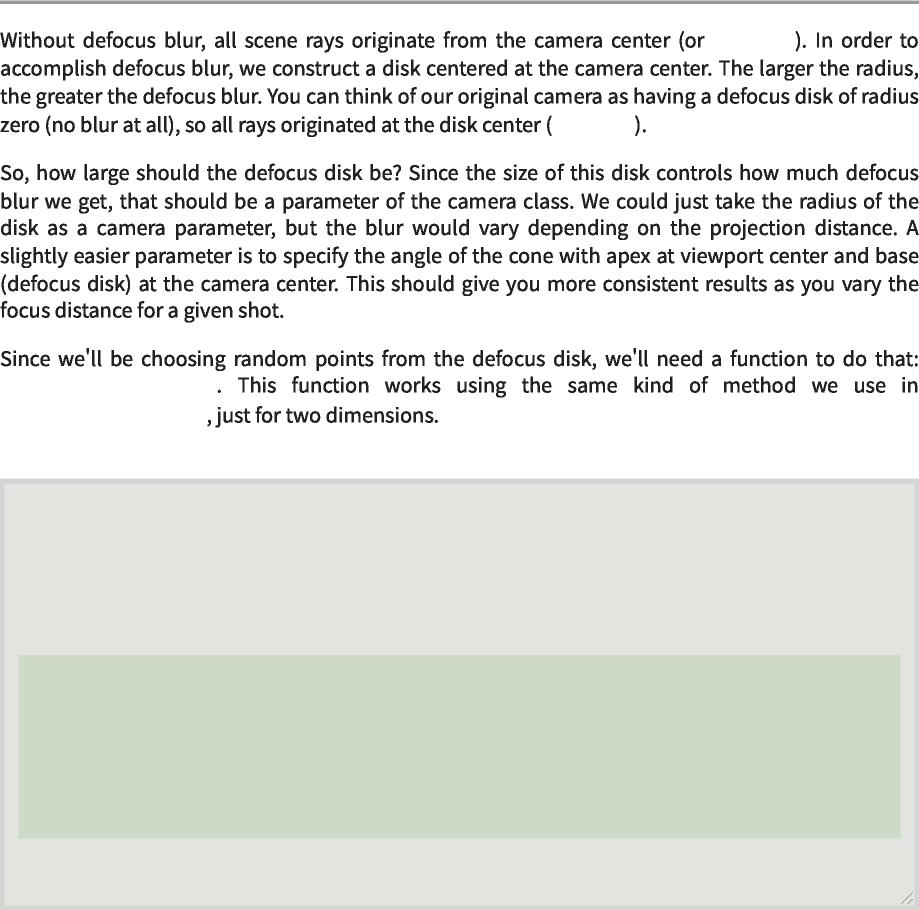
https://raytracing.github.io/books/RayTracingInOneWeekend.html 107/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 107/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**13.2. Generating Sample Rays**  
 13.2.生成样品射线

https://raytracing.github.io/books/RayTracingInOneWeekend.html 108/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 108/120



lookfrom  
 lookfrom

lookfrom  
 lookfrom

random\_in\_unit\_disk() random\_unit\_vector()  
 random\_in\_unit\_disk()random\_unit\_vector()

...

inline vec3 **unit\_vector**(const vec3& u) { return v / v.length();  
 inline vec3 unit\_vector(const vec3&u){return v/v.length()；

}

inline vec3 **random\_in\_unit\_disk**() {  
 inline vec3 random\_in\_unit\_disk(){

while (true) {  
 while(true){

auto p = vec3(random\_double(-1,1), random\_double(-1,1), 0);  
 auto p=vec3(random\_double(-1,1),random\_double(-1,1),0)；

if (p.length\_squared() < 1)  
 如果(p.length\_squared()<1)

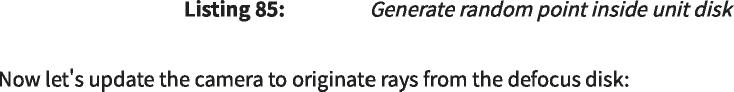
return p;  
 返回p；

}

}

...

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



[vec3.h]  
 [vec3.h]

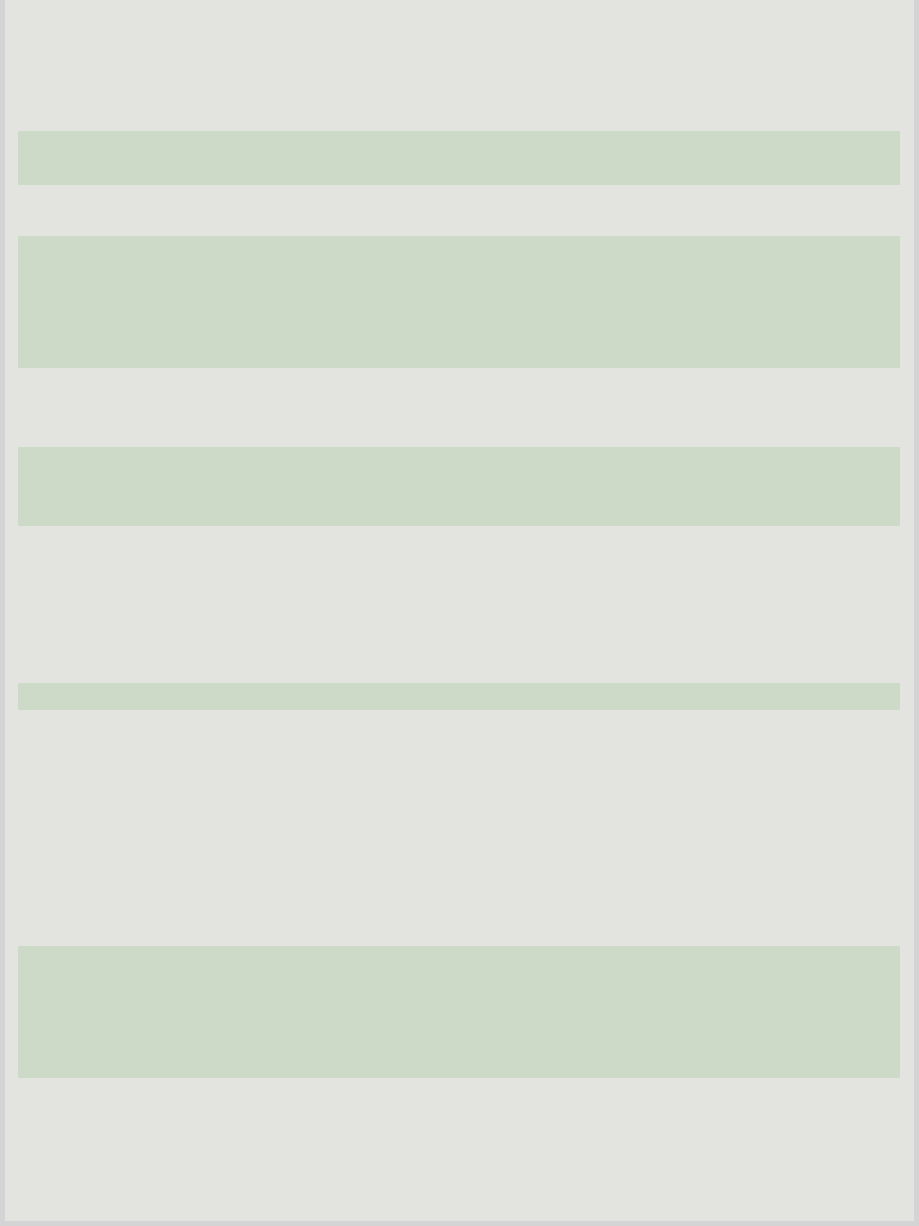
https://raytracing.github.io/books/RayTracingInOneWeekend.html 109/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 109/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

|  |
| --- |
| class camera {  public:  类相机{ 公众：  double aspect\_ratio = 1.0; // Ratio of image width over height  双纵横比=1.0； //图像宽高比  int image\_width = 100; // Rendered image width in pixel count  intimage\_width=100； //以像素计数表示的渲染图像宽度  int samples\_per\_pixel = 10; // Count of random samples for each pixel  intsamples\_per\_pixel=10； //每个像素的随机样本计数  int max\_depth = 10; // Maximum number of ray bounces into scene  intmax\_depth=10； //反射到场景中的最大光线数  double vfov = 90; // Vertical view angle (field of view)  双vfov=90； //垂直视角(视场)  point3 lookfrom = point3(0,0,0); // Point camera is looking from  point3 lookfrom=point3(0,0,0)； //点摄像机从  point3 lookat = point3(0,0,-1); // Point camera is looking at  点3 lookat=点3(0,0,-1)； //点摄像机正在查看  vec3 vup = vec3(0,1,0); // Camera-relative 'up' direction  vec3vup=vec3(0,1,0)； //相机相对“向上”方向 |
| double defocus\_angle = 0; // Variation angle of rays through each pixel  双散焦角=0； //通过每个像素的光线变化角  double focus\_dist = 10; // Distance from camera lookfrom point to plane of perfect focus  double focus\_dist=10； //从相机观察点到完美聚焦平面的距离 |
| ...  private:  私人：  int image\_height; // Rendered image height  intimage\_height； //渲染图像高度  double pixel\_samples\_scale; // Color scale factor for a sum of pixel samples  双pixel\_samples\_scale； //像素样本总和的色标因子  point3 center; // Camera center  点3中心； //相机中心  point3 pixel00\_loc; // Location of pixel 0, 0  点3 pixel00\_loc； //像素0,0的位置  vec3 pixel\_delta\_u; // Offset to pixel to the right  vec3pixel\_delta\_u； //向右偏移到像素  vec3 pixel\_delta\_v; // Offset to pixel below  vec3pixel\_delta\_v； //偏移量到下面的像素  vec3 u, v, w; // Camera frame basis vectors  vec3u，v，w； //相机帧基向量 |
| vec3 defocus\_disk\_u; // Defocus disk horizontal radius  vec3defocus\_disk\_u； //散焦盘水平半径  vec3 defocus\_disk\_v; // Defocus disk vertical radius  vec3defocus\_disk\_v； //散焦盘垂直半径 |
| void **initialize**() {  void initialize(){  image\_height = int(image\_width / aspect\_ratio);  image\_height=int(image\_width/aspect\_ratio)；  image\_height = (image\_height < 1) ? 1 : image\_height;  image\_height=(image\_height<1)？1：image\_height；  pixel\_samples\_scale = 1.0 / samples\_per\_pixel;  pixel\_samples\_scale=1.0/samples\_per\_pixel；  center = lookfrom;  center=lookfrom；  // Determine viewport dimensions.  //确定视口尺寸。 |
| focal\_length lookat).length();  焦距 lookat).length()；  auto = (lookfrom  汽车 =（从  auto theta = degrees\_to\_radians(vfov);  自动θ=度数到弧度(vfov)；  auto h = std::tan(theta/2);  auto h=std：：tan(θ/2)； |
| auto viewport\_height = 2 \* h \* focus\_dist;  auto viewport\_height=2\*h\*focus\_dist； |
| auto viewport\_width = viewport\_height \* (double(image\_width)/image\_height);  auto viewport\_width=viewport\_height\*(double(image\_width)/image\_height)；  // Calculate the u,v,w unit basis vectors for the camera coordinate frame.  //计算相机坐标系的u、v、w单位基向量。  w = unit\_vector(lookfrom - lookat);  w=unit\_vector(lookfrom-lookat)；  u = unit\_vector(cross(vup, w));  u=unit\_vector(cross(vup，w))；  v = cross(w, u);  v=十字(w，u)；  // Calculate the vectors across the horizontal and down the vertical viewport edges.  //计算水平和垂直视口边缘的向量。  vec3 viewport\_u = viewport\_width \* u; // Vector across viewport horizontal edge  vec3 viewport\_u=viewport\_width\*u； //跨视口水平边缘的矢量  vec3 viewport\_v = viewport\_height \* -v; // Vector down viewport vertical edge  vec3 viewport\_v=viewport\_height\*-v； //向下向视口垂直边缘 |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 110/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 110/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



// Calculate the horizontal and vertical delta vectors to the next pixel.  
 //计算到下一个像素的水平和垂直增量向量。

pixel\_delta\_u = viewport\_u / image\_width;  
 pixel\_delta\_u=viewport\_u/image\_width；

pixel\_delta\_v = viewport\_v / image\_height;  
 pixel\_delta\_v=viewport\_v/image\_height；

// Calculate the location of the upper left pixel.  
 //计算左上像素的位置。

auto viewport\_upper\_left = center - (focus\_dist \* w) - viewport\_u/2 - viewport\_v/2;  
 auto viewport\_upper\_left=center-(focus\_dist\*w)-viewport\_u/2-viewport\_v/2；

pixel00\_loc = viewport\_upper\_left + 0.5 \* (pixel\_delta\_u + pixel\_delta\_v);  
 pixel00\_loc=viewport\_upper\_left+0.5\*(pixel\_delta\_u+pixel\_delta\_v)；

// Calculate the camera defocus disk basis vectors.  
 //计算相机散焦盘基向量。

auto defocus\_radius = focus\_dist \* std::tan(degrees\_to\_radians(defocus\_angle  
 auto defocus\_radius=focus\_dist\*std：：tan(degrees\_to\_radians(defocus\_angle)

/ 2));

defocus\_disk\_u = u \* defocus\_radius; defocus\_disk\_v = v \* defocus\_radius;  
 defocus\_disk\_u=u\*defocus\_radius；defocus\_disk\_v=v\*defocus\_radius；

}

ray **get\_ray**(int i, int j) const {  
 射线get\_ray(int i,int j)const{

// Construct a camera ray originating from the defocus disk and directed at a  
 //构造一条源自散焦盘并指向

randomly  
 随机地

// sampled point around the pixel location i, j.  
 //像素位置i，j周围的采样点。

auto offset = sample\_square();  
 auto offset=sample\_square()；

auto pixel\_sample = pixel00\_loc  
 auto pixel\_sample=pixel00\_loc

+ ((i + offset.x()) \* pixel\_delta\_u)  
 +((i+offset.x())\*pixel\_delta\_u)

+ ((j + offset.y()) \* pixel\_delta\_v);  
 +((j+offset.y())\*pixel\_delta\_v)；

auto ray\_origin = (defocus\_angle <= 0) ? center : defocus\_disk\_sample();  
 auto ray\_origin=(defocus\_angle<=0)？中心：defocus\_disk\_sample()；

auto ray\_direction = pixel\_sample - ray\_origin;  
 auto ray\_direction=pixel\_sample-ray\_origin；

return ray(ray\_origin, ray\_direction);  
 返回光线(ray\_origin，ray\_direction)；

}

vec3 **sample\_square**() const {  
 vec3 sample\_square()const{

...

}

point3 **defocus\_disk\_sample**() const {  
 point3 defocus\_disk\_sample()const{

// Returns a random point in the camera defocus disk.  
 //返回相机散焦盘中的一个随机点。

auto p = random\_in\_unit\_disk();  
 auto p=random\_in\_unit\_disk()；

return center + (p[0] \* defocus\_disk\_u) + (p[1] \* defocus\_disk\_v);  
 返回中心+(p[0]\*defocus\_disk\_u)+(p[1]\*defocus\_disk\_v)；

}

color **ray\_color**(const ray& r, int depth, const hittable& world) const {  
 color ray\_color(const ray&r，int depth，const hittable&world)const{

...

}

};

|  |  |  |
| --- | --- | --- |
|  | [camera.h]  [摄像机.h] |  |

https://raytracing.github.io/books/RayTracingInOneWeekend.html 111/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 111/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| int main() {  int main(){  ... |  |  |  |  |
| camera cam;  照相机凸轮； |  |  |  |  |
| cam.aspect\_ratio  凸轮纵横比 | = | 16.0 | / | 9.0; |
| cam.image\_width  cam.image\_width | = | 400; |  |  |
| cam.samples\_per\_pixel  cam.samples\_per\_pixel | = | 100; |  |  |
| cam.max\_depth  凸轮最大深度 | = | 50; |  |  |

cam.vfov = 20;  
 cam.vfov=20；

cam.lookfrom = point3(-2,2,1);  
 cam.lookfrom=point3(-2,2,1)；

cam.lookat = point3(0,0,-1);  
 cam.lookat=point3(0,0,-1)；

cam.vup = vec3(0,1,0);  
 vup=vec3(0,1,0)；

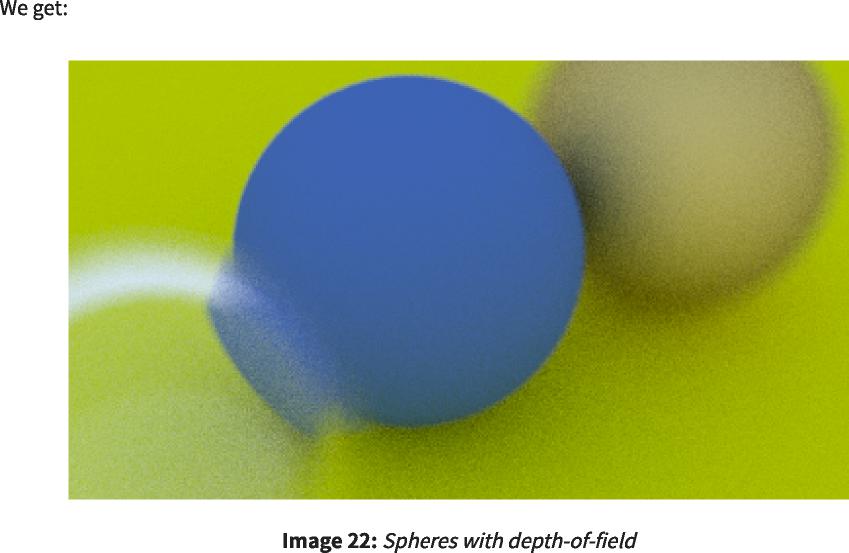
cam.defocus\_angle = 10.0;  
 cam.defocus\_angle=10.0；

cam.focus\_dist = 3.4;  
 cam.focus\_dist=3.4；

cam.render(world);   
}  
 凸轮。渲染（世界）； }



|  |  |  |
| --- | --- | --- |
|  | [main.cc]  [main.cc] |  |



**14. Where Next?**  
 14.下一步去哪里？

https://raytracing.github.io/books/RayTracingInOneWeekend.html 112/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 112/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

14.1. A Final Render  
 14.1.最终渲染



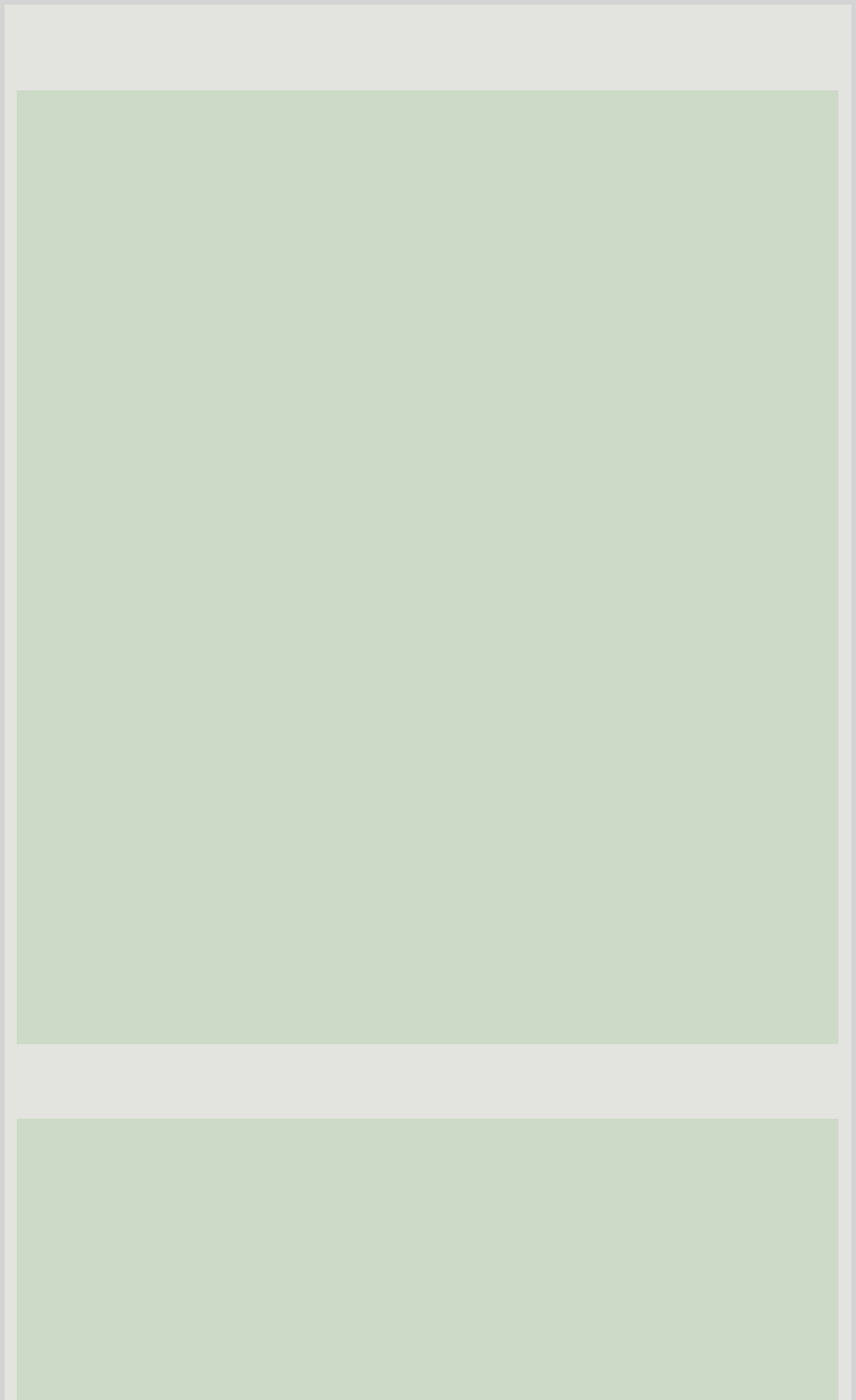
https://raytracing.github.io/books/RayTracingInOneWeekend.html 113/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 113/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 114/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 114/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

https://raytracing.github.io/books/RayTracingInOneWeekend.html 115/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 115/120



int main() {  
 int main(){

hittable\_list world;  
 hittable\_list world；

auto ground\_material = make\_shared<lambertian>(color(0.5, 0.5, 0.5)); world.add(make\_shared<sphere>(point3(0,-1000,0), 1000, ground\_material));  
 auto ground\_material=make\_shared<lambertian>(color(0.5,0.5,0.5))；world.add(make\_shared<sphere>(point3(0,-1000,0),1000,ground\_material))；

for (int a = -11; a < 11; a++) {  
 for(int a=-11;a<11;a++){

for (int b = -11; b < 11; b++) {  
 for(int b=-11;b<11;b++){

auto choose\_mat = random\_double();  
 auto choose\_mat=random\_double()；

point3 center(a + 0.9\*random\_double(), 0.2, b + 0.9\*random\_double());  
 点3中心(a+0.9\*random\_double(),0.2,b+0.9\*random\_double())；

if ((center - point3(4, 0.2, 0)).length() > 0.9) { shared\_ptr<material> sphere\_material;  
 if((center-point3(4,0.2,0)).length()>0.9){shared\_ptr<material>sphere\_material；

if (choose\_mat < 0.8) {  
 if(choose\_mat<0.8){

// diffuse  
 //扩散

auto albedo = color::random() \* color::random();  
 自动反照率=color：：random()\*color：：random()；

sphere\_material = make\_shared<lambertian>(albedo);  
 sphere\_material=make\_shared<lambertian>(反照率)；

world.add(make\_shared<sphere>(center, 0.2, sphere\_material));  
 world.add(make\_shared<sphere>(center，0.2，sphere\_material))；

} else if (choose\_mat < 0.95) {  
 }else if(choose\_mat<0.95){

// metal  
 //金属

auto albedo = color::random(0.5, 1);  
 自动反照率=颜色：：随机（0.5，1）；

auto fuzz = random\_double(0, 0.5);  
 auto fuzz=random\_double(0,0.5)；

sphere\_material = make\_shared<metal>(albedo, fuzz);  
 sphere\_material=make\_shared<metal>(反照率，模糊)；

world.add(make\_shared<sphere>(center, 0.2, sphere\_material));  
 world.add(make\_shared<sphere>(center，0.2，sphere\_material))；

} else {  
 }else{

// glass  
 //玻璃

sphere\_material = make\_shared<dielectric>(1.5);  
 sphere\_material=make\_shared<电介质>(1.5)；

world.add(make\_shared<sphere>(center, 0.2, sphere\_material));  
 world.add(make\_shared<sphere>(center，0.2，sphere\_material))；

}

}

}

}

auto material1 = make\_shared<dielectric>(1.5);  
 auto material1=make\_shared<电介质>(1.5)；

world.add(make\_shared<sphere>(point3(0, 1, 0), 1.0, material1));  
 world.add(make\_shared<sphere>(point3(0,1,0),1.0,material1))；

auto material2 = make\_shared<lambertian>(color(0.4, 0.2, 0.1)); world.add(make\_shared<sphere>(point3(-4, 1, 0), 1.0, material2));  
 auto material2=make\_shared<lambertian>(color(0.4,0.2,0.1))；world.add(make\_shared<sphere>(point3(-4,1,0),1.0,material2))；

auto material3 = make\_shared<metal>(color(0.7, 0.6, 0.5), 0.0); world.add(make\_shared<sphere>(point3(4, 1, 0), 1.0, material3));  
 auto material3=make\_shared<metal>(color(0.7,0.6,0.5),0.0)；world.add(make\_shared<sphere>(point3(4,1,0),1.0,material3))；

camera cam;  
 照相机凸轮；

cam.aspect\_ratio = 16.0 / 9.0;  
 cam.aspect\_ratio=16.0/9.0；

cam.image\_width = 1200; cam.samples\_per\_pixel = 500;  
 cam.image\_width=1200；cam.samples\_per\_pixel=500；

cam.max\_depth = 50;  
 cam.max\_depth=50；

cam.vfov = 20;  
 cam.vfov=20；

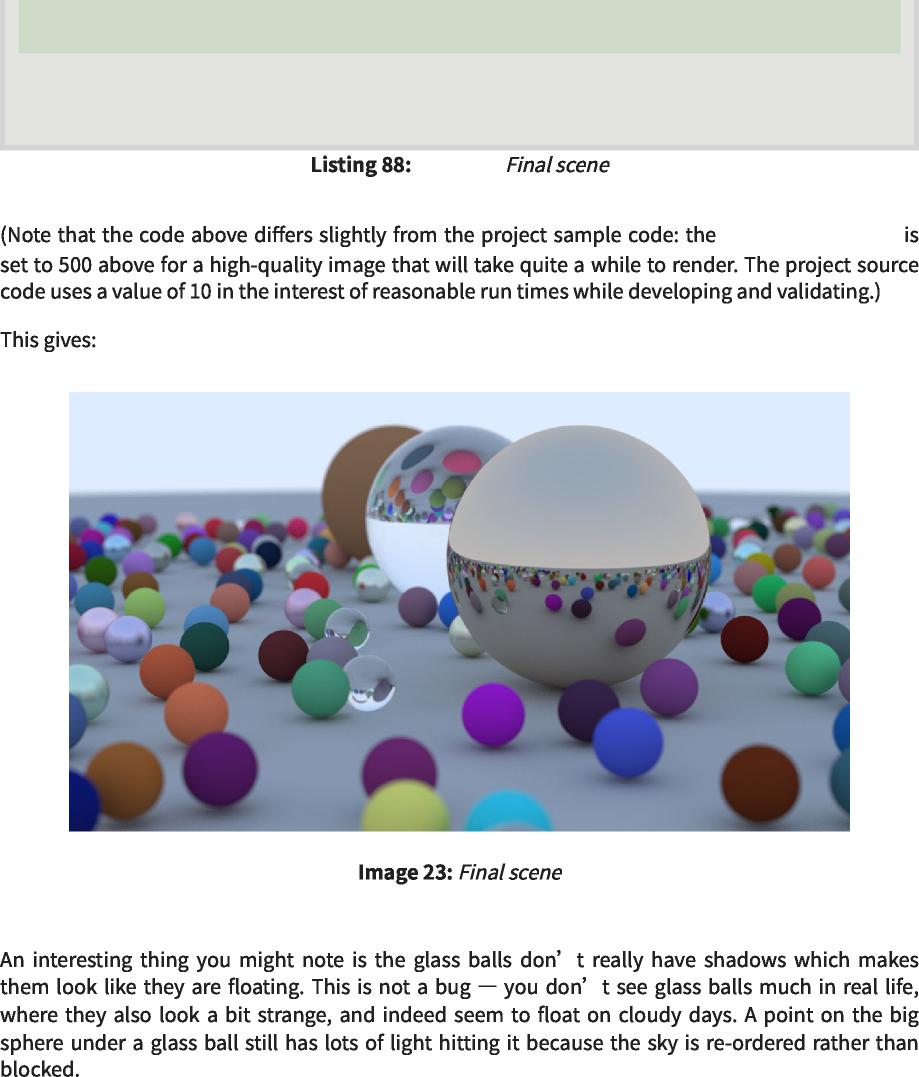
cam.lookfrom = point3(13,2,3);  
 cam.lookfrom=point 3(13,2,3)；

cam.lookat = point3(0,0,0);  
 凸轮。lookat=点3（0,0,0）；

cam.vup = vec3(0,1,0);  
 cam.vup=vec3(0,1,0)；

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**14.2. Next Steps**  
 14.2.后续步骤



**}**

**cam.defocus\_angle = 0.6;**  
 cam.defocus\_angle=0.6；

**cam.focus\_dist = 10.0;**  
 cam.focus\_dist=10.0；

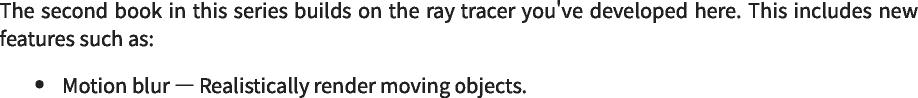
**cam.render(world);**  
 凸轮。渲染（世界）；

[main.cc]  
 [main.cc]

**samples\_per\_pixel**  
 每像素样本数

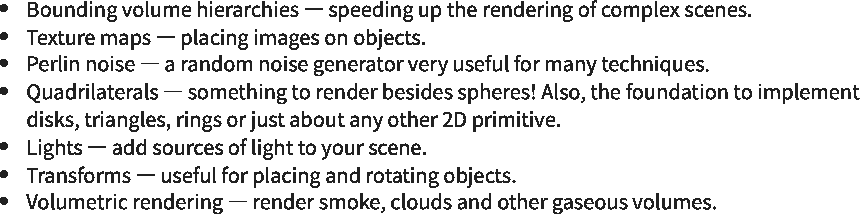


**14.2.1 Book 2: *Ray Tracing: The Next Week***  
 14.2.1第二册：光线追踪：下周

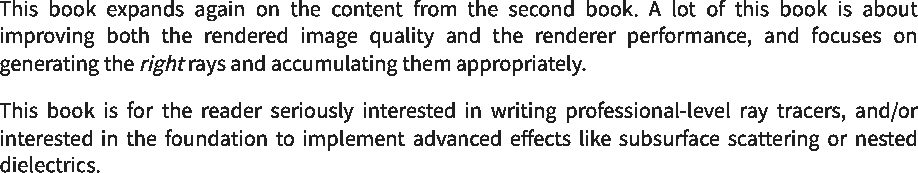


https://raytracing.github.io/books/RayTracingInOneWeekend.html 116/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 116/120

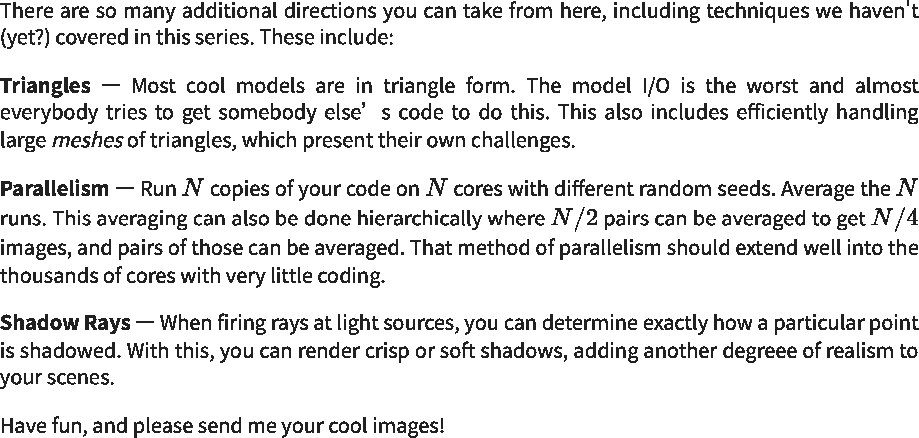
2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



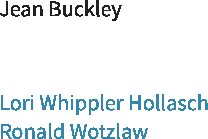
**14.2.2 Book 3: *Ray Tracing: The Rest of Your Life***  
 14.2.2第三册：光线追踪：你的余生

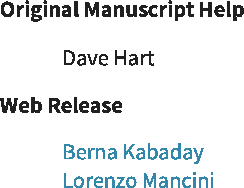


**14.2.3 Other Directions**  
 14.2.3其他方向



**15. Acknowledgments**  
 15.致谢

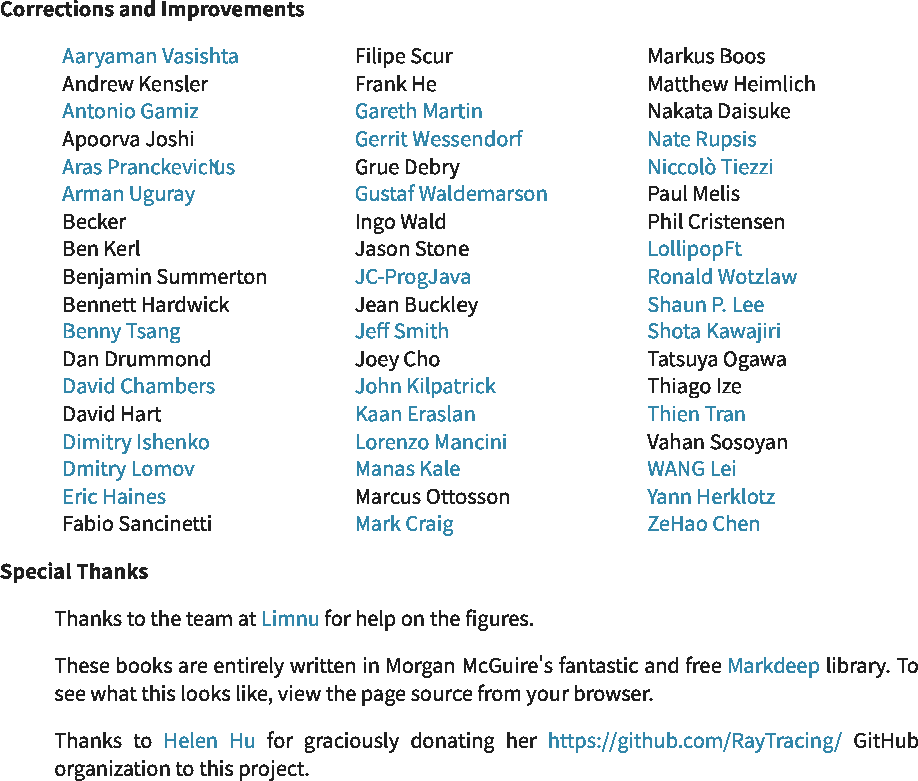




ı   
 我

https://raytracing.github.io/books/RayTracingInOneWeekend.html 117/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 117/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪



**16. Citing This Book**  
 16.引用本书



**16.1. Basic Data**  
 16.1.基本数据



**16.2. Snippets**  
 16.2.代码片段

https://raytracing.github.io/books/RayTracingInOneWeekend.html 118/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 118/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**16.2.1 Markdown**  
 16.2.1降价

**[\_Ray Tracing in One Weekend\_]**  
 [\_一个周末的光线追踪\_]

**(https://raytracing.github.io/books/RayTracingInOneWeekend.html)**   
 (https://raytracing.github.io/books/RayTracingInOneWeekend.html)



**16.2.2 HTML**  
 16.2.2 HTML

**<a href='https://raytracing.github.io/books/RayTracingInOneWeekend.html'>**  
 <a href='https://raytracing.github.io/books/RayTracingInOneWeekend.html'>

**<cite>Ray Tracing in One Weekend</cite>**  
 <cite>一个周末的光线追踪</cite>

**</a>**  
 </a>



**16.2.3 LaTeX and BibTex**  
 16.2.3乳胶和BibTex

**-\cite{Shirley2025RTW1}**  
 -\引用{Shirley2025RTW1}

**@misc{Shirley2025RTW1,**  
 @misc{Shirley2025RTW1,

**title = {Ray Tracing in One Weekend},**  
 title={一个周末的光线追踪},

**author = {Peter Shirley, Trevor David Black, Steve Hollasch},**  
 author={Peter Shirley，Trevor David Black，Steve Hollasch}，

**year = {2025},**  
 年份={2025}，

**month = {April},**  
 月={四月},

**note = {\small**  
 注={\small

**\texttt{https://raytracing.github.io/books/RayTracingInOneWeekend.html}},**  
 \texttt{https://raytracing.github.io/books/RayTracingInOneWeekend.html}}，

**url = {https://raytracing.github.io/books/RayTracingInOneWeekend.html}**  
 url={https://raytracing.github.io/books/RayTracingInOneWeekend.html}

**}**



**16.2.4 BibLaTeX**  
 16.2.4 BibLaTeX

**\usepackage{biblatex}   
-\cite{Shirley2025RTW1}**  
 \usepackage{biblatex} -\引用{Shirley2025RTW1}

**@online{Shirley2025RTW1,**  
 @online{Shirley2025RTW1,

**title = {Ray Tracing in One Weekend},**  
 title={一个周末的光线追踪},

**author = {Peter Shirley, Trevor David Black, Steve Hollasch},**  
 author={Peter Shirley，Trevor David Black，Steve Hollasch}，

**year = {2025},**  
 年份={2025}，

**month = {April},**  
 月={四月},

**url = {https://raytracing.github.io/books/RayTracingInOneWeekend.html}**  
 url={https://raytracing.github.io/books/RayTracingInOneWeekend.html}

**}**



https://raytracing.github.io/books/RayTracingInOneWeekend.html 119/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 119/120

2025/8/3 22:02 Ray Tracing in One Weekend  
 2025/8/3 22:02一个周末的光线追踪

**16.2.5 IEEE**  
 16.2.5 IEEE

**“Ray Tracing in One Weekend.” raytracing.github.io/books/RayTracingInOneWeekend.html (accessed MMM. DD, YYYY)**  
 “射线追踪在一个周末。”raytracing.github.io/books/RayTracingInOneWeekend.html（访问日期：年月日）



**16.2.6 MLA:**  
 16.2.6 MLA：

**Ray Tracing in One Weekend. raytracing.github.io/books/RayTracingInOneWeekend.html Accessed DD MMM. YYYY.**  
 一个周末的光线追踪。raytracing.github.io/books/RayTracingInOneWeekend.html访问了DD嗯。YYYY.





https://raytracing.github.io/books/RayTracingInOneWeekend.html 120/120  
 https://raytracing.github.io/books/RayTracingInOneWeekend.html 120/120