Path Tracing in 5 minutes

シングルファイルで拡張しやすいパストレーサーを作る

@yumcyawiz

2020/08/15

OSK 夏休み LT 会 2020

Intro

「レイトレ」してみたいなあ
「レイトレすごい !」
「時代はレイトレーシング」

最近こんな言葉をよく聞く気がします

Intro

でも・・・

「レイトレしてみたいけど難しそう」
「C++ よくわからん」

レイトレしたくても参入障壁は大きいようです

Intro

そこで今回は見るだけで C++ でレイトレーサーを書いた気持ちになれる LT をします

Goal

後で拡張しやすいパストレーサーをシングルファイルで作る

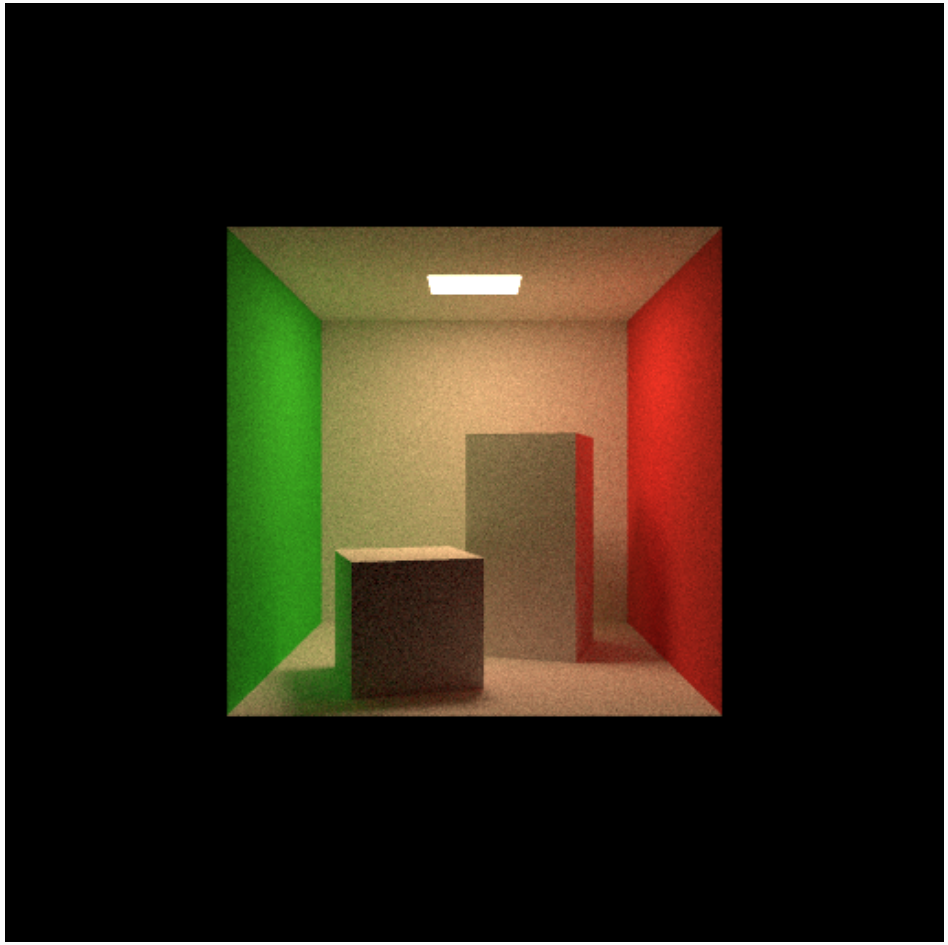

Figure 1: 今回のレンダラーで出た絵

Project Setup

Project Setup

今回使うのは

- C++17
- CMake
- OpenMP

CMake

```
cmake_minimum_required(VERSION 3.12)

project(pathtracing_in_5minutes LANGUAGES C CXX)

add_executable(main main.cpp)

target_compile_features(main PUBLIC cxx_std_17)
set_target_properties(main PROPERTIES CXX_EXTENSIONS OFF)

find_package(OpenMP)
if(OpenMP_CXX_FOUND)
    target_link_libraries(main PUBLIC OpenMP::OpenMP_CXX)
endif()
```

Figure 2: CMakeLists.txt

Build

```
~/C++/pathtracing_in_5minutes develop
❯ mkdir build

~/C++/pathtracing_in_5minutes develop
❯ cd build

~/C++/pathtracing_in_5minutes/build develop
❯ cmake ..
-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/yumcyawiz/C++/pathtracing_in_5minutes/build

~/C++/pathtracing_in_5minutes/build develop
❯ make -j4
Scanning dependencies of target main
[ 50%] Building CXX object CMakeFiles/main.dir/main.cpp.o
[100%] Linking CXX executable main
[100%] Built target main
```

Figure 3: Build の様子

Coding

Vec3

Vec3 は 3 次元ベクトルを表すクラス

```
// 3-dimensional vector
class Vec3 {
 public:
  Real x;
  Real y;
  Real z;

  Vec3() { x = y = z = 0; }
  Vec3(Real _x) { x = y = z = _x; }
  Vec3(Real _x, Real _y, Real _z) : x(_x), y(_y), z(_z) {}

  Vec3 operator-() const { return Vec3(-x, -y, -z); }

  Vec3& operator+=(const Vec3& v) {
    x += v.x;
    y += v.y;
    z += v.z;
    return *this;
  }
  Vec3& operator+=(Real k) {
    x += k;
    y += k;
    z += k;
    return *this;
  }
  Vec3& operator-=(const Vec3& v) {
    x -= v.x;
    y -= v.y;
    z -= v.z;
    return *this;
  }
  Vec3& operator-=(Real k) {
    x -= k;
    y -= k;
    z -= k;
    return *this;
  }
  Vec3& operator*=(const Vec3& v) {
    x *= v.x;
    y *= v.y;
    z *= v.z;
    return *this;
  }
  Vec3& operator*=(Real k) {
    x *= k;
    y *= k;
    z *= k;
    return *this;
  }
  Vec3& operator/=(const Vec3& v) {
    x /= v.x;
    y /= v.y;
    z /= v.z;
    return *this;
```

Ray

Ray は始点 (Vec3) と方向 (Vec3) を持つ

```cpp
// Ray
class Ray {
 public:
  Vec3 origin;     // origin of ray
  Vec3 direction;  // direction of ray

  static constexpr Real tmin = 1e-3;
  static constexpr Real tmax = std::numeric_limits<Real>::max();

  Ray() {}
  Ray(const Vec3& _origin, const Vec3& _direction)
      : origin(_origin), direction(_direction) {}

  Vec3 operator()(Real t) const { return origin + t * direction; }
};
```

Figure 5: Ray Class

PCG32 を使って乱数生成

```c
// PCG32 random number generator

// *Really* minimal PCG32 code / (c) 2014 M.E. O'Neill / pcg-random.org
// Licensed under Apache License 2.0 (NO WARRANTY, etc. see website)

typedef struct {
  uint64_t state;
  uint64_t inc;
} pcg32_random_t;

uint32_t pcg32_random_r(pcg32_random_t* rng) {
  uint64_t oldstate = rng->state;
  // Advance internal state
  rng->state = oldstate * 6364136223846793005ULL + (rng->inc | 1);
  // Calculate output function (XSH RR), uses old state for max ILP
  uint32_t xorshifted = ((oldstate >> 18u) ^ oldstate) >> 27u;
  uint32_t rot = oldstate >> 59u;
  return (xorshifted >> rot) | (xorshifted << ((-rot) & 31));
}

```

Figure 6: PCG32 Random Number Generator

Sampler

乱数生成器のラッパー

```cpp
// random number generator
class Sampler {
 public:
  pcg32_random_t state;

  static constexpr uint64_t PCG32_DEFAULT_STATE = 0x853c49e6748fea9bULL;
  static constexpr uint64_t PCG32_DEFAULT_INC = 0xda3e39cb94b95bdbULL;

  Sampler() {
    state.state = PCG32_DEFAULT_STATE;
    state.inc = PCG32_DEFAULT_INC;
  }
  Sampler(uint64_t seed) { setSeed(seed); }

  // set seed of random number generator
  void setSeed(uint64_t seed) {
    state.state = seed;
    uniformReal();
    uniformReal();
  }

  // return random value in [0, 1]
  Real uniformReal() {
    return std::min(static_cast<Real>(pcg32_random_r(&state) * 0x1p-32),
                    ONE_MINUS_EPS);
  }
};
```

Figure 7: Sampler Class

Film

カメラのフィルム (画像) を表すクラス

```cpp
// Film
class Film {
 public:
  uint32_t width;            // width of image in [px]
  uint32_t height;           // height of image in [px]
  const Real width_length;   // physical length in x-direction in [m]
  const Real height_length;  // physical length in y-direction in [m]

  // pixels are row-major array.
  Vec3* pixels;  // an array contains RGB at each pixel

  Film(uint32_t _width, uint32_t _height, Real _width_length = 0.025,
       Real _height_length = 0.025)
      : width(_width),
        height(_height),
        width_length(_width_length),
        height_length(_height_length) {
    pixels = new Vec3[width * height];
  }
  ~Film() { delete[] pixels; }

  // getter and setter
  Vec3 getPixel(uint32_t i, uint32_t j) const { return pixels[j + width * i]; }
  void setPixel(uint32_t i, uint32_t j, const Vec3& rgb) {
    pixels[j + width * i] = rgb;
  }

  // add RGB to pixel
  void addPixel(uint32_t i, uint32_t j, const Vec3& rgb) {
    pixels[j + width * i] += rgb;
  }

  void divide(uint64_t k) {
    for (int i = 0; i < height; ++i) {
      for (int j = 0; j < width; ++j) {
        pixels[j + width * i] /= k;
      }
    }
  }

```

Figure 8: Film Class

PPM 出力部分

```cpp
// output ppm image
void writePPM(const std::string& filename) const {
  std::ofstream file(filename);
  if (!file) {
    printf("failed to open %s", filename.c_str());
    return;
  }

  // ppm header
  file << "P3" << std::endl;
  file << width << " " << height << std::endl;
  file << "255" << std::endl;

  for (int i = 0; i < height; ++i) {
    for (int j = 0; j < width; ++j) {
      Vec3 rgb = getPixel(i, j);

      // gamma correction
      rgb.x = std::pow(rgb.x, 1 / 2.2);
      rgb.y = std::pow(rgb.y, 1 / 2.2);
      rgb.z = std::pow(rgb.z, 1 / 2.2);

      // convert real to uint
      const uint32_t R =
          std::clamp(static_cast<uint32_t>(255 * rgb.x), 0u, 255u);
      const uint32_t G =
          std::clamp(static_cast<uint32_t>(255 * rgb.y), 0u, 255u);
      const uint32_t B =
          std::clamp(static_cast<uint32_t>(255 * rgb.z), 0u, 255u);

      file << R << " " << G << " " << B << std::endl;
    }
  }

  file.close();
 }
};
```

Figure 9: Film Class

Camera

レイを生成するところ
ピンホールカメラモデル

```cpp
// Camera
// pinhole camera model
class Camera {
 public:
  Vec3 origin;   // origin of camera(position of film center in 3-dimensional
                 // space)
  Vec3 forward;  // forward direction of camera
  Vec3 right;    // right direction of camera
  Vec3 up;       // up direction of camera

  std::shared_ptr<Film> film;  // image sensor of camera

  Real focal_length;  // focal length of pinhole camera

  Camera(const Vec3& _origin, const Vec3& _forward,
         const std::shared_ptr<Film>& _film, Real fov)
      : origin(_origin), forward(normalize(_forward)), film(_film) {
    // create orthonormal basis from forward direction vector
    right = normalize(cross(forward, Vec3(0, 1, 0)));
    up = normalize(cross(right, forward));

    std::cout << "[Camera] origin: " << origin << std::endl;
    std::cout << "[Camera] forward: " << forward << std::endl;
    std::cout << "[Camera] right: " << right << std::endl;
    std::cout << "[Camera] up: " << up << std::endl;

    const Real diagonal_length =
        std::sqrt(film->width_length * film->width_length +
                  film->height_length * film->height_length);
    focal_length = diagonal_length / (2.0 * std::tan(0.5 * fov));

    std::cout << "[Camera] focal length: " << focal_length << std::endl;
  }

  // sample ray from camera
  Ray sampleRay(uint32_t i, uint32_t j, Sampler& sampler) {
    // sample position in pixel with super sampling
    const Real u = film->width_length *
                   (2.0 * (j + sampler.uniformReal()) - film->width) /
                   film->height;
    const Real v = film->height_length *
                   (2.0 * (i + sampler.uniformReal()) - film->height) /
                   film->height;
    const Vec3 p_film = origin + u * right + v * up;

    // sample ray
    const Vec3 p_pinhole = origin + focal_length * forward;
    return Ray(origin, normalize(p_pinhole - p_film));
  }
};
```

Figure 10: Camera Class

Material

物体表面の反射特性を表すクラス
レイを反射させる役割を持つ

```cpp
// Material utils
Vec3 reflect(const Vec3& v, const Vec3& n) { return -v + 2 * dot(v, n) * n; }

// Material
// computation on local coordinate(surface normal is y-axis)
class Material {
 public:
  // BRDF sampling
  virtual Vec3 sampleBRDF(const Vec3& wo, Sampler& sampler, Vec3& direction,
                          Real& pdf_solid) const = 0;
};

class Diffuse : public Material {
 public:
  const Vec3 kd;

  Diffuse(const Vec3& _kd) : kd(_kd) {}

  Vec3 sampleBRDF(const Vec3& wo, Sampler& sampler, Vec3& direction,
                  Real& pdf_solid) const override {
    // sample direction
    direction = sampleCosineHemisphere(sampler.uniformReal(),
                                       sampler.uniformReal(), pdf_solid);

    // compute BRDF
    return INV_PI * kd;
  }
};

class Mirror : public Material {
 public:
  const Vec3 ks;

  Mirror(const Vec3& _ks) : ks(_ks) {}

  Vec3 sampleBRDF(const Vec3& wo, Sampler& sampler, Vec3& direction,
                  Real& pdf_solid) const override {
    direction = reflect(wo, Vec3(0, 1, 0));
    pdf_solid = 1;

    const Real cos = std::abs(direction.y);
    return ks / cos;
  }
};
```

Figure 11: Material Class

Light

光源を表現するクラス

```
423
424   // Light
425   class Light {
426    public:
427     const Vec3 le;
428
429     Light(const Vec3& _le) : le(_le) {}
430
431     Vec3 Le() const { return le; }
432   };
433
```

Figure 12: Light Class

IntersectInfo

交差計算の結果を表現する構造体

```cpp
// IntersectInfo

// prototype declaration of Primitive
class Primitive;

struct IntersectInfo {
  Real t;          // hit distance
  Vec3 hitPos;     // hit potision
  Vec3 hitNormal;  // surface normal at hit position
  Vec3 dpdu;       // derivative of hit position with u(tangent vector)
  Vec3 dpdv;       // derivative of hit position with v(cotangent vector

  std::shared_ptr<Primitive> hitPrimitive;  // hit primitive

  IntersectInfo() : t(std::numeric_limits<Real>::max()) {}
};
```

Figure 13: IntersectInfo Struct

Shape

物体の形を表現する抽象クラス

```
454
455  // Shape
456  class Shape {
457   public:
458    virtual bool intersect(const Ray& ray, IntersectInfo& info) const = 0;
459  };
460
461  // Sphere
```

Figure 14: Shape Class

Sphere

球体を表現するクラス

```cpp
// Sphere
class Sphere : public Shape {
 public:
  const Vec3 center;  // center of sphere
  const Real radius;  // radius of sphere

  Sphere(const Vec3& _center, Real _radius)
      : center(_center), radius(_radius) {}

  // intersect ray with sphere
  bool intersect(const Ray& ray, IntersectInfo& info) const override {
    // solve quadratic equation
    const Real b = dot(ray.direction, ray.origin - center);
    const Real c = length2(ray.origin - center) - radius * radius;
    const Real D = b * b - c;
    if (D < 0) return false;

    // choose closer hit distance
    const Real t0 = -b - std::sqrt(D);
    const Real t1 = -b + std::sqrt(D);
    Real t = t0;
    if (t < ray.tmin || t > ray.tmax) {
      t = t1;
      if (t < ray.tmin || t > ray.tmax) {
        return false;
      }
    }

    info.t = t;
    info.hitPos = ray(t);

    const Vec3 r = info.hitPos - center;
    info.hitNormal = normalize(r);

    info.dpdu = normalize(Vec3(-r.z, 0, r.x));

    // compute local coordinate(phi, theta) of hit position
    Real phi = std::atan2(r.z, r.x);
    if (phi < 0) phi += PI2;
    const Real theta =
        std::acos(std::clamp(r.y / radius, Real(-1.0), Real(1.0)));

    info.dpdv = normalize(Vec3(std::cos(phi) * r.y, -radius * std ::sin(theta),
                               std::sin(phi) * r.y));

    return true;
  }
};
```

Figure 15: Sphere Class

Plane

平面を表現するクラス

```cpp
// Plane
class Plane : public Shape {
 public:
  Vec3 leftCornerPoint;
  Vec3 right;
  Vec3 up;
  Vec3 normal;

  Vec3 center;
  Vec3 rightDir;
  Real rightLength;
  Vec3 upDir;
  Real upLength;

  Plane(const Vec3& _leftCornerPoint, const Vec3& _up, const Vec3& _right)
      : leftCornerPoint(_leftCornerPoint), right(_right), up(_up) {
    normal = normalize(cross(right, up));
    center = leftCornerPoint + 0.5 * right + 0.5 * up;
    rightDir = normalize(right);
    rightLength = length(right);
    upDir = normalize(up);
    upLength = length(up);
  };

  bool intersect(const Ray& ray, IntersectInfo& res) const override {
    const Real t =
        -dot(ray.origin - center, normal) / dot(ray.direction, normal);
    if (t < ray.tmin || t > ray.tmax) return false;

    const Vec3 hitPos = ray(t);
    const Real dx = dot(hitPos - leftCornerPoint, rightDir);
    const Real dy = dot(hitPos - leftCornerPoint, upDir);
    if (dx < 0 || dx > rightLength || dy < 0 || dy > upLength) return false;

    res.t = t;
    res.hitPos = hitPos;
    res.hitNormal = dot(-ray.direction, normal) > 0 ? normal : -normal;
    res.dpdu = rightDir;
    res.dpdv = upDir;
    return true;
  };
};

```

Figure 16: Plane Class

Primitive

物体を表現するクラス

Material, Light, Shape を一緒に持つ

```
554
555  // Primitive
556  class Primitive {
557   public:
558    std::shared_ptr<Shape> shape;
559    std::shared_ptr<Material> material;
560    std::shared_ptr<Light> light;
561
562    Primitive(const std::shared_ptr<Shape>& _shape,
563              const std::shared_ptr<Material>& _material,
564              const std::shared_ptr<Light>& _light)
565        : shape(_shape), material(_material), light(_light) {}
566
567    bool hasLight() const { return light != nullptr; }
568
569    // intersect ray with primitive
570    bool intersect(const Ray& ray, IntersectInfo& info) const {
571      return shape->intersect(ray, info);
572    }
573
574    // sample direction propotional to BRDF
575    Vec3 sampleBRDF(const Ray& ray, const IntersectInfo& info, Sampler& sampler,
576                    Vec3& direction, Real& pdf_solid) const {
577      // convert direction vector from world to local
578      const Vec3 wo =
579          worldToLocal(-ray.direction, info.dpdu, info.hitNormal, info.dpdv);
580
581      Vec3 direction_local;
582      const Vec3 BRDF =
583          material->sampleBRDF(wo, sampler, direction_local, pdf_solid);
584
585      // convert direction vector from local to world
586      direction = normalize(
587          localToWorld(direction_local, info.dpdu, info.hitNormal, info.dpdv));
588
589      return BRDF;
590    }
591  };
592
```

Figure 17: Primitive Class

Sky

空の色を表現するクラス

```
629  // Sky
630  class Sky {
631   public:
632    const Vec3 le;
633
634    Sky(const Vec3& _le) : le(_le) {}
635
636    Vec3 Le(const Ray& ray) const { return le; }
637  };
638
```

Figure 18: Sky Class

Scene

シーンを表現するクラス
カメラ, 物体集合, 空を一緒に持つ

```cpp
// Scene
class Scene {
 public:
  const std::shared_ptr<Camera> camera;
  const std::vector<std::shared_ptr<Primitive>> prims;
  const std::shared_ptr<Sky> sky;

  Intersector intersector;

  Scene(const std::shared_ptr<Camera>& _camera,
        const std::vector<std ::shared_ptr<Primitive>>& _prims,
        const std ::shared_ptr<Sky>& _sky)
      : camera(_camera), prims(_prims), sky(_sky) {
    // setup intersector
    intersector = Intersector(prims);
  }

  bool intersect(const Ray& ray, IntersectInfo& info) const {
    return intersector.intersect(ray, info);
  }
};
```

Figure 19: Scene Class

Integrator

レイの持つ放射輝度を計算する
パストレ

```cpp
// Integrator
class Integrator {
 public:
  const uint64_t maxDepth = 100;
  const Real russian_roulette_prob = 0.99;

  Integrator() {}

  // compute given ray's radiance
  Vec3 radiance(const Ray& ray_in, const Scene& scene, Sampler& sampler) const {
    Ray ray = ray_in;
    Vec3 radiance;
    Vec3 throughput(1);

    for (uint64_t depth = 0; depth < maxDepth; depth++) {
      // russian roulette
      if (sampler.uniformReal() >= russian_roulette_prob) break;
      throughput /= russian_roulette_prob;

      // compute intersection with scene
      IntersectInfo info;
      if (scene.intersect(ray, info)) {
        const auto& prim = info.hitPrimitive;

        // Le
        if (prim->hasLight()) {
          radiance += throughput * prim->light->Le();
        }

        // BRDF sampling
        Vec3 next_direction;
        Real pdf_solid;
        const Vec3 BRDF =
            prim->sampleBRDF(ray, info, sampler, next_direction, pdf_solid);

        // cosine term
        const Real cos = std::abs(dot(next_direction, info.hitNormal));

        // update throughput
        throughput *= BRDF * cos / pdf_solid;

        // update ray
        ray = Ray(info.hitPos, next_direction);

      } else {
        radiance += throughput * scene.sky->Le(ray);
        break;
      }
    }

    return radiance;
  }
};
```

Figure 20: Integrator Class

Renderer

レンダリングを行う部分

```
736  class Renderer {
737   public:
738    const Scene scene;
739    const Integrator integrator;
740    Sampler sampler;
741
742    Renderer(const Scene& _scene, const Integrator& _integrator,
743             const Sampler& _sampler)
744        : scene(_scene), integrator(_integrator), sampler(_sampler) {}
745
746    void render(uint64_t n_samples) {
747      for (uint64_t k = 0; k < n_samples; ++k) {
748  #pragma omp parallel for schedule(dynamic, 1)
749        for (uint32_t i = 0; i < scene.camera->film->height; ++i) {
750          for (uint32_t j = 0; j < scene.camera->film->width; ++j) {
751            // sample ray
752            const Ray ray = scene.camera->sampleRay(i, j, sampler);
753
754            // compute radiance
755            const Vec3 radiance = integrator.radiance(ray, scene, sampler);
756
757            // add radiance on pixel
758            scene.camera->film->addPixel(i, j, radiance);
759
760            if (omp_get_thread_num() == 0) {
761              const Real index =
762                  j + i * scene.camera->film->width +
763                  k * scene.camera->film->width * scene.camera->film->height;
764              const Real max = scene.camera->film->width *
765                               scene.camera->film->height * n_samples;
766              std::cout << progressbar(index, max) << " "
767                        << percentage(index, max) << "%"
768                        << "\r" << std::flush;
769            }
770          }
771        }
772      }
773
774      // divide by n_samples
775      scene.camera->film->divide(n_samples);
776
777      // write ppm
778      scene.camera->film->writePPM("output.ppm");
779    }
780  };
781
```

Figure 21: Renderer Class

main 関数

```
918  int main() {
919      // parameters
920      const uint32_t width = 512;
921      const uint32_t height = 512;
922      const uint64_t samples = 4000;
923
924      // setup image
925      const auto film = std::make_shared<Film>(width, height);
926
927      // setup sampler
928      Sampler sampler;
929
930      // setup integrator
931      Integrator integrator;
932
933      // setup scene
934      Scene scene = cornellBoxScene(film);
935
936      // setup renderer
937      Renderer renderer(scene, integrator, sampler);
938
939      // render
940      renderer.render(samples);
941
942      return 0;
943  }
```

Figure 22: main 関数

Rendering

Test Scene

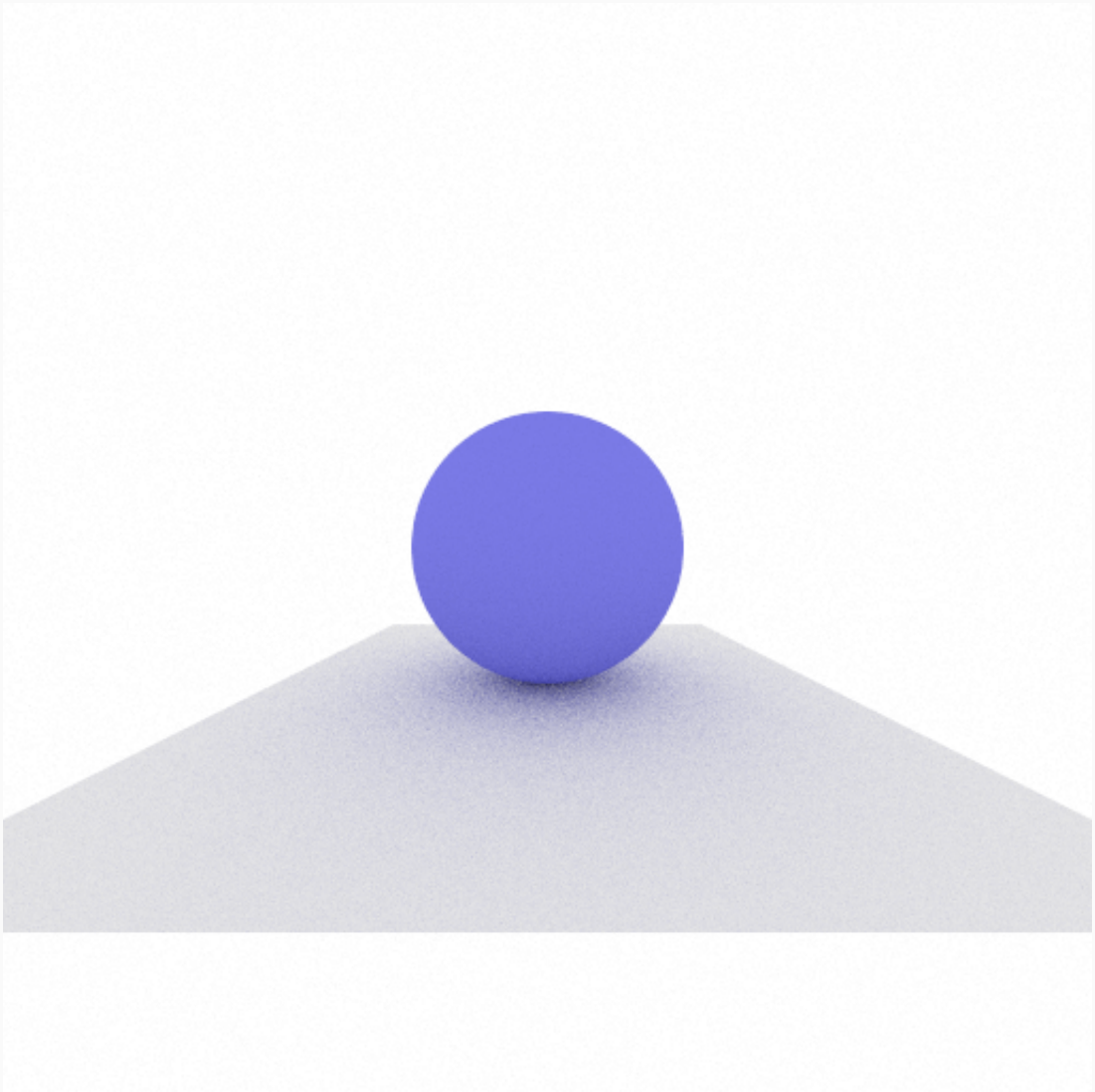

Figure 23: Test Scene

CornellBox Scene

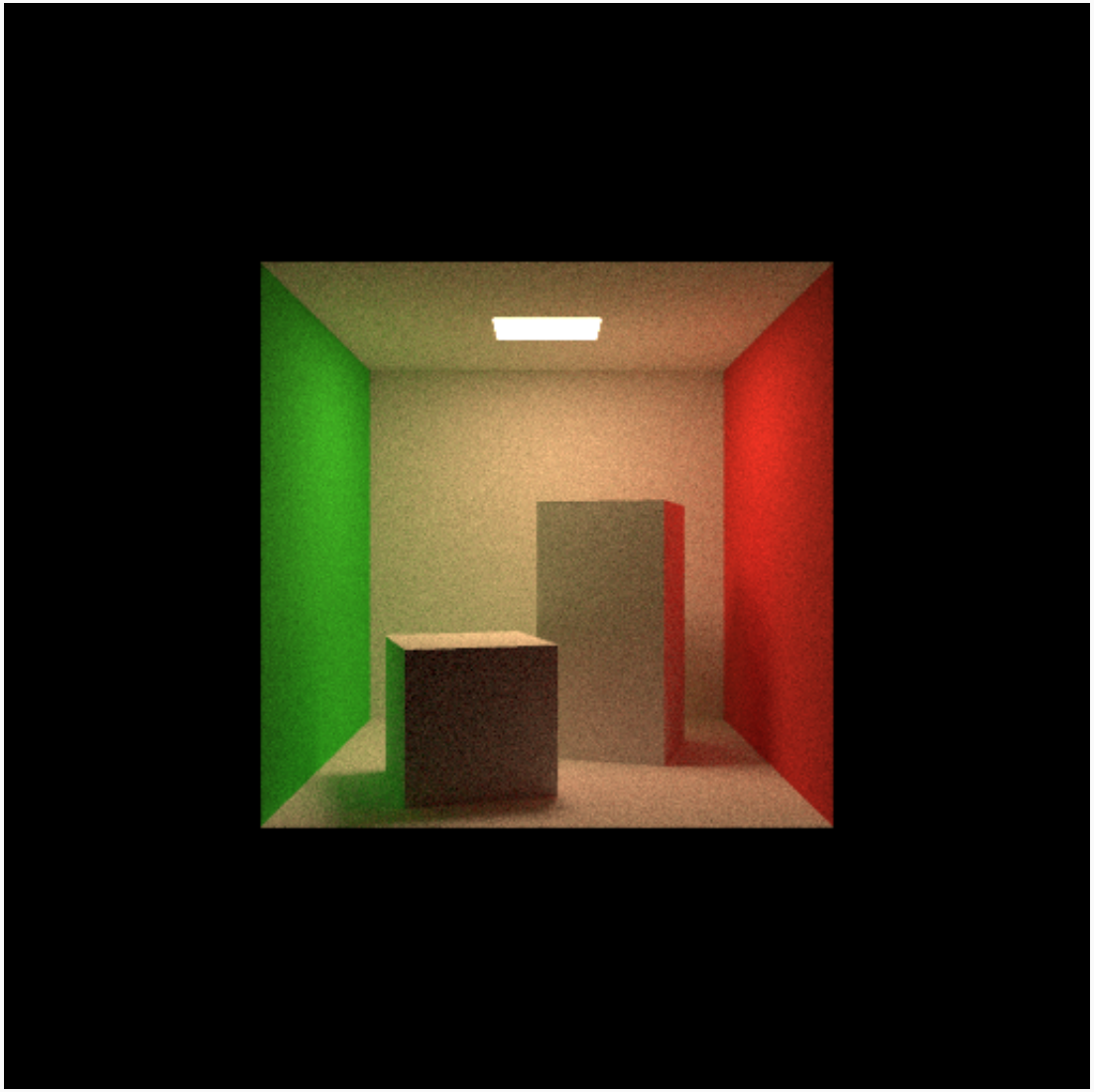

Figure 24: Cornell Box