國立臺北科技大學

進階 C 語言實務

Homework 4

老師：蔣政諺

班級：電機碩一

學號：111318133

姓名：魏千竣

日期：112/06/06

1. Program Description

1. 匯入所需的標頭檔，和使用命名空間 std。這邊我有特別加上 include/來使用 include 資料夾裡的標頭檔。

```
#include <cstdio>
#include <vector>
#include <cstdlib>
#include <iostream>
#include "include/BVH.h"
#include "include/Sphere.h"
#include "include/Cube.h"
#include "include/Vector3.h"

using std::vector;

using namespace std;
```

2.這兩個函式分別用於生成隨機數字和隨機向量，後面用於生成球體的隨機位置。

```
// Return a random number in [0,1]
float rand01() {
    return rand() * (1.f / RAND_MAX);
}

// Return a random vector with each component in the range [-1,1]
Vector3 randVector3() {
    return Vector3(rand01(), rand01(), rand01())*2.f - Vector3(1,1,1);
}
```

3. 主程式部分，首先創建一個包含多個球體對象的向量。向量中的每個球體都使用 new Sphere(randVector3(), 0.05f) 進行初始化，即每個球體的中心位置是隨機的（使用前面定義的 randVector3() 函數生成），半徑設為 0.05。這樣的話，所有的球體都會在一個單位立方體內（因為 randVector3() 生成的向量的每個分量都在[-1,1]範圍內）。
接著創建一個立方體對象，並添加到對象向量中。立方體的中心被設置在原點（0,0,0），大小被設置為 1。

```
int main(int argc, char **argv) {

    // Create a million spheres packed in the space of a cube
    const unsigned int N = 1000;
    vector<Object*> objects;
    printf("Constructing %d spheres...\n", N);
    for(size_t i=0; i<N; ++i) {
        objects.push_back(new Sphere(randVector3(), 0.05f));
    }

    // Create a Cube and add it to the objects list
    Vector3 cubeCenter(0, 0, 0);  // Set the cube's center at the origin
    Vector3 cubeSize(1, 1, 1);  // Cube's size is 1
    printf("Constructing 1 spheres...\n");
    objects.push_back(new Cube(cubeCenter, cubeSize));
```

4. 使用這些對象構建一個 BVH 結構。這是一種空間劃分的數據結構，用於高效地進行光線與對象的交集檢測。
接著分配足夠的空間來儲存 1024*1024 個像素的所有數據，用於儲存生成的圖像。
最後設定相機的位置、焦點和向上方向，並計算了相機的方向、右側向量和上側向量。

```
// Compute a BVH for this object set
BVH bvh(&objects);

// Allocate space for some image pixels
const unsigned int width=1024, height=1024;
float* pixels = new float[width*height*3];

// Create a camera from position and focus point
Vector3 camera_position(1.6, 1.3, 1.6);
Vector3 camera_focus(0,0,0);
Vector3 camera_up(0,1,0);

// Camera tangent space
Vector3 camera_dir = normalize(camera_focus - camera_position);
Vector3 camera_u = normalize(camera_dir ^ camera_up);
Vector3 camera_v = normalize(camera_u ^ camera_dir);
```

5. 這段程式將逐步對像素進行光線追蹤。對於每個像素計算一個光線（從相機位置出發，經過像素中心，並延伸到場景中），並檢查這個光線與場景中的任何對象是否相交。如果沒有相交的對象，該像素將被設置為黑色；如果有相交的對象，該像素的顏色將被設置為該對象的顏色。

```
printf("Rendering image (%dx%d)...\n", width, height);
// Raytrace over every pixel
for(size_t i=0; i<width; ++i) {
    for(size_t j=0; j<height; ++j) {
        size_t index = 3*(width * j + i);

        float u = (i+.5f) / (float)(width-1) - .5f;
        float v = (height-1-j+.5f) / (float)(height-1) - .5f;
        float fov = .5f / tanf( 70.f * 3.14159265*.5f / 180.f);

        // This is only valid for square aspect ratio images
        Ray ray(camera_position, normalize(u*camera_u + v*camera_v + fov*camera_dir));

        IntersectionInfo I;
        bool hit = bvh.getIntersection(ray, &I, false);

        if(!hit) {
            pixels[index] = pixels[index+1] = pixels[index+2] = 0.f;
        } else {

            // Just for fun, we'll make the color based on the normal
            // const Vector3 normal = I.object->getNormal(I);
            // const Vector3 color(fabs(normal.x), fabs(normal.y), fabs(normal.z));
            // const Vector3 color(1.f, 0.f, 0.f);  // Red color
            const Vector3& color = I.object->getColor();

            pixels[index  ] = color.x;
            pixels[index+1] = color.y;
            pixels[index+2] = color.z;
        }
    }
}
```

6. 最後程式將渲染結果寫入 PPM 格式的圖像檔，並釋放之前分配的像素。

```
// Output image file (PPM Format)
printf("Writing out image file: \"render.ppm\"\n");
FILE *image = fopen("render1.ppm", "w");
fprintf(image, "P6\n%d %d\n255\n", width, height);
for(size_t j=0; j<height; ++j) {
    for(size_t i=0; i<width; ++i) {
        size_t index = 3*(width * j + i);
        unsigned char r = std::max(std::min(pixels[index  ]*255.f, 255.f), 0.f);
        unsigned char g = std::max(std::min(pixels[index+1]*255.f, 255.f), 0.f);
        unsigned char b = std::max(std::min(pixels[index+2]*255.f, 255.f), 0.f);
        fprintf(image, "%c%c%c", r,g,b);
    }
}
fclose(image);

// Cleanup
delete[] pixels;
```

7.標頭檔更改部分：

- 在 object.h 中新增 color 的成員變數，新增成員函式用於設置和獲取物體的顏色。

```
Vector3 color;  // Add a new member variable for color
```

```
//! Return the color of this object
Vector3 getColor() const {
    return color;
}

//! Set the color of this object
void setColor(const Vector3& newColor) {
    color = newColor;
}
```

- 在 sphere.h 中，由於中間需要賦值給 object，所以需要確保 Sphere 是從 Object 類公開派生，sphere 構造函數的訪問級別改為 public。最後設定顏色都為紅色。

```
class Sphere : public Object {
public:
    Vector3 center; // Center of the sphere
    float r, r2; // Radius, Radius^2

    Sphere(const Vector3& center, float radius)
    : center(center), r(radius), r2(radius*radius) {
        setColor(Vector3(1.f, 0.f, 0.f));  // Red color
    }
```

- 在 intersection.h 中 include 進 vector3.h 來利用。

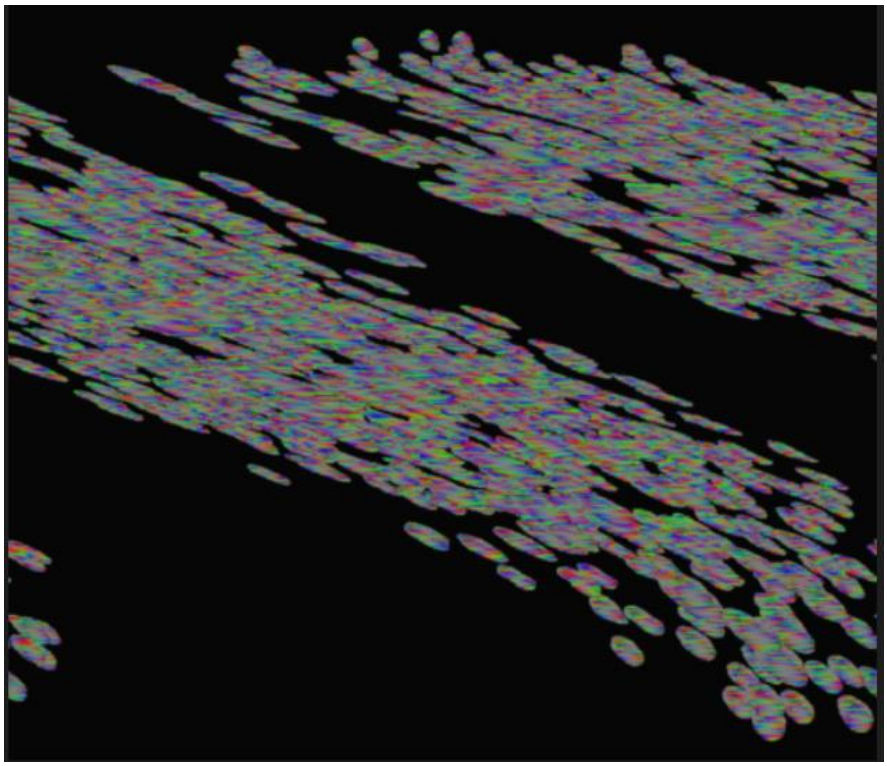
```
#include "Vector3.h"
```

- 另外創建一個 cube.h 檔，來創建正方體形狀，以及設定中心點、半尺寸，及設定顏色等。且設定好光線與物體的交互程式、法相量，以及立方體的邊界框與中心點，下面只展示一部分程式。

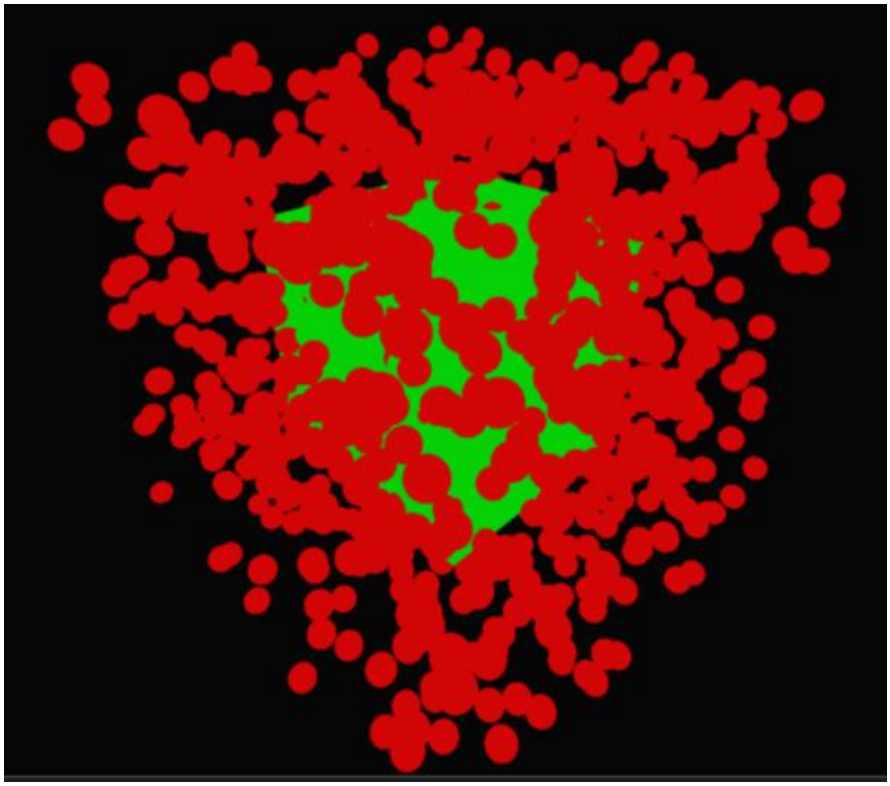
```
class Cube : public Object {
public:
    Vector3 center; // Center of the cube
    Vector3 halfSize; // Half of the size (dimensions) of the cube

    Cube(const Vector3& center, const Vector3& size)
        : center(center), halfSize(size * 0.5f) {
            setColor(Vector3(0.f, 1.f, 0.f));  // Green color
        }
```

2. Result Display

Sample :



增加 cube :



3. Conclusion

會讓每一個物體有各自的顏色，是因為使用原本的表面法線方向來決定顏色時，各種物體很容易因為顏色相近而重疊，導致形狀不易辨認，因此我增加了設定顏色，來使每一個物體有各別的顏色，也確實更好的辨認出形狀。也新增了 cube 形狀。