Fejlesztői dokumentáció

Vincze Csongor

December 10, 2025

Contents

1 A program ismertetése, áttekintése

Ez a program egy nagyon alapvető sugárkövetőt (ray tracer) implementál le. A lényege, a virtuális kamerából (ahonnét mi a képet látjuk) sugarakat küld, és ezeknek a sugaraknak a gömbökkel való ütközöését követi. Miután a felhasználó megadja a megfelelő paramétereket a program elkezdi a képkockák generálását. A képkockák generálásának előrehaladását közben a felhasználó egy folyamatjelző sávon követheti. Amint az összes képkocka generálása megtörtént a szoftver összefűzi a képkockákat egy videóvá, és ezt el is indítja.

2 Nyelv, használt csomagok, kiegészítők

A teljes projekt C nyelvben íródott, az alapvető C könyvtárak felhasználásával, minden külső csomag nélkül. A program nagy vonalakban a `https://raytracing.github.io/books/RayTracingInOneWeekend.html#overview` dokumentációt követi (természetesen C nyelvre átírva).

3 Az arhitektura áttekintése, a program kezelése

A `video_render.exe` futtatásakor a program bekéri a videó elkészítéséhez szükéges adatokat. Ezek sorba:

1. Milyen felbontást szeretnél? (szélesség pixelekben) (elfogadható értékek: 2^n, ahol n egész és $4 < n < 11$)
2. Hány gömbot szeretnél? (ajánlott: 2-10)
3. Milyen hosszú videót szeretnél? (a videó 30fps-en fut, 1-10 közötti egész szám elfogadott)

Az első bemenet a kép vízszíntes pixelszámát adja meg (a képarány 16:9). Itt 1024-nél már jelentősen lelassul a program. 256-nál viszont még elég alacsony lesz a készített videó minősége. Érdemes lehet ezt az értéket 512-re állítani. Ezzel az adattal a program már el tudja készíteni az ún. nézőportált (viewport), ami meghatározza, hogy mit, hogy fogunk látni. A többi paraméter fix. A `viewport_creator` függvény végzi ezt a folyamatot, és a `video_render.c` hívja meg a `render.h`-ból. A gömbök száma egészen 0-tól 50-ig terjedhet. Ennek ellenére nem ajánlott 10-nél többet megadni mivel ez is jelentősen lelassítja a programot. A gömbökre van egy külön struktúra (sphere), egy ilyen tömböt hozunk létre. A a videó készítés közben mozgatni fogjuk a gömböket. Mivel csak gömb alakú objektumokkat dolgozunk, így érdemes ennek a tömbnek az elejére egy előre beállított nagy gömböt rakni, hogy legyen egy földünk a videókhoz. Így valójában egyel több gömbbel operálunk mintamennyit bekértünk. A harmadik bekérésnél hasonló okokból kiindulva 5 körüli értéket érdemes megadni. Mivel konstans 30 fps-el dolgozunk így csak simán visszaszámoljuk, hogy hány képkockát kell legeneráljunk.

4 Fájlok és azok leírásaik

A program egy fő .c fájlból és 10 header fájlból áll. A header fájlok főleg a program egyes részeihez biztosítanak függvényeket. Minden fájl ugyan abba n a mappában van.

4.1 vec_3.h

Mivel 3 dimenzióban dolgozunk szükségünk van vektoroperációkra. Ez a fájl biztosítja ezek meglétét. A fájlban minden függvény mellé fel van tüntetve, hogy mit csinál, így ezt nem részletezem itt. Ez a fájl tartalmazza a `vec_3.h` struktúrát is ami, lényegében egy 3 dimenziós vektor típus. Ez a struktúra nagyon sokat könnyít a vekorok kezelésén.

4.2 ray.h

Ez a rész csupán a sugár (ray) struktúrát és egz sugárkövetőt tartalmazza. A sugár egy kiinduló pontból és egy irányvektorból áll. a sugárkövető pedig visszaadja egy t paraméter függvényében, hogy hol van a sugár.

4.3 intervals.h

Ez igazából csak megkönnyíti a program írását, átláthatóságát, ha valahol intervallumosan kell dolgozni. Például a a gömbök eltalálásánál használjuk ezt ki.

4.4 sphere.h

Ez a fájl a gömb struktúrát hozza létre. A gömb egy középpontból és egz sugárból áll. Ezután a `hit_sphere` függvény következik ami, lényegileg eldönti, hogy az adott sugár eltalálta-e az adott gömböt. Ez a függvény egy bináris értéket ad vissza. Emellett kiszámolja, hogy hol történt a találat, milyen hosszú volt a sugár, és a találatnál a gömb normálvektorát. Ezeket elmenit egy pointerrel inputként megadott `hit_rec` struktúrába. Ezt két függvény követi; az `_array_spheres` és a `_rand_spheres`. Előbbi egy vízszintes sorba rak le gömböket. Ez inkább a teszteléshez, paraméterek állításához használandó. Útóbbi pedig (bizonyos korlátok kötött) random paraméterekkel rakja le a gömböket.

4.5 hittable.h

Ez a header létrehoz egy `hit_rec` struktúrát, a következő módon;

```
typedef struct{
    point_3 p;              // ebben taroljuk az eltalalt pontot
    vec_3 outward_normal;   // mindig kifele mutato norma
    vec_3 normal;           // ez a norma mindig a sugarral
                            // elenkezo iranyba mutat
    double t;               // mennyi "ido"-be telt a sugarnak
                            // eltalalni a pontot
    bool front_face;        // kivulrol jon-e a sugar
} hit_rec;
```

Ezután létrehozom a `hit_side` függvényt. A függvény lényege, hogy meghatározza, hogy a gömb melyik oldalán vagyunk. (A felhasználó által használt .exe fájl enélkül is működne mivel úgy vannak beállítva a paraméterek, hogy minden gömböt kívülről találjukn el.)

4.6 hit_list.h

Ez a header csak egy függvényből áll, ez viszont egy nagyon hasznos függvény. a `which_hit` Megnézi, hogy egy adott sugár melyik gömböket találja el (meghívja a `hit_sphere` függvényt minden gömbre.), és a legközelebbi találat adatait lementi egy `hit_rec` struktúrába. A függvény egyet ad vissza, ha sikerült eltalálni egy gömböt, nullát ha nem sikerült eltalálni semmit sem.

4.7 color.h

Először csinálunk egy color nevű típust ami egy hasommása a `vec_3` stuktúrának. (Itt kihasználjuk azt, hogy egy RGB-vel megadott színnek pont három komponense van, csakúgy mint egy vektornak is.) Ez a fájl is egy függvényt tartalmaz. A `_color_divider` inputként egy színt és egy fájl pointert kap. A színt szétbontja megfelelő RGB formátumra, és beleírja a fájlba.

4.8 common_headers.h

Ez igazából csak arra jó, hogy ne kelljen folyton az összes header fájlt cipelni, ebbe belerakjuk őket és ezt "icludoljuk". Emellett a program megnyitását követően felugró felirat kiírására itt tároljuk a függvényt. (A többi dolog ami még itt van nem olyan fontos.)

4.9 animation.h

Ez azért felelős, hogy valóban videót kapjunk és ne csak egymás után rakott ugyan olyan képeket. Két opciónk van. Az első a gömböket az y tengely mentén harmónikusan rezgeti (`harm_osc_y`). A másik pedig minden irányban valamilyen kis random értéket ad hozzá a gömbök középpontjának koordinátáihoz, így egyfajta véletlen séta szerűséget kapunk (`random_walk`).

4.10 render.h

Itt először létrehozzuk a kamera és a nézőportál (viewport) paramétereit, a `viewport_creator` fügvénnyel. A fix képarányból és a kép szélességéből kiszámoljuk a kép magasságát. Utána pedig a nézőportált (lényegileg a kiküldött sugaraknak ad egz korlátot hogy az egész térből mit lásson). Ezután kiszámoljuk a kis vektorokat amiket majd hozzáadogatunk a bal felső pixelhez, így jutunk el az összes pixelhez. Végül pedig kiszámoljuk a bal felső pixel (és a bal felső csúcs) koordinátáit.

Ezután jön a `ray_color` függvény. Ez egy rekurzív függvény ami először megnézi hogy az adott irányba indított sugár talált-e (meghívja a `which_hit` függvényt) Ha volt találat akkor meghívja magát ujra (a rekurziós szám 20-ra van állítva) a gömb találati pontjáról a gömbtől elfelé random irányba mutató irányvektorral (itt a `vec_3`-ból hívja meg a `_unit_vec_on_hemisphere` függvényt), és a gömb adott pontjával (ezeket berakja egy `ray_3 struktúrába`), és a végül visszakapott értéket megszorozza a `reflection_number`-el (lényegileg az, hogy a színnek hányad részét adja vissza). Amennyiben nem volt találat a háttár színét adja vissza. Ha elertuk a rekurziós számnyi "pattogást" akkor feketét adunk vissza.

Ezt egy rövid függvény követi; `rand_square()` amit az x-y síkon a (-0.5 — 0.5)x(-0.5 — 0.5) területen visszaad egy random vektort (z komponens mindig 0).

Végezetül a render függvény végigiterál a nézőportál minden pixelén bal fentről kezdve. A legbelső for ciklus azért kell, hogy ne olyan élesek, szögletesek legyenek az objektumok határai

(itt használjuk fel a `rand_square()` függvényt, hogy mindig kicsit más irányba küldjük a sugarakat, és aztán kiátlagoljuk az eredményt). Elküldünk minden pixelhez tartozó színértéket a `_color_divider`-be ami kiszínezi, egy fájlba lementi az RGB értékeket. Amint megvagyunk egy képkockával bezárjuk azt a fájlt.