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 ütközését 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. A program és a hozzá tartózó dolgok elérhetőek a: `https://github.com/CsongorVincze/Ray_tracing_Csongi_Bongi` linken.

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 architektúra áttekintése, a program kezelése

A `video_render.exe` fájl készen futtatható. De ezt úgy is megkaphatjuk ha egy mappába lementjük a letöltött `.c` és `.h` fájlokat és együtt lefordítjuk őket.: `gcc -o video_render.exe video_render.c vec_3.c sphere.c hittable.c hit_list.c render.c animation.c common.c color.c ray.c intervals.c -lm` (Esetlegesen lehet a Makefile-al is végezni ezt a folyamatot.) A `video_render.exe` futtatásakor a program bekéri a videó elkészítéséhez szükséges adatokat. Ezek sorban:

1. Milyen felbontást szeretnél? (szélesség pixelekben) (elfogadható értékek: 2^n, ahol n egész és $4 < n < 11$)
2. Hány gömböt 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 vízszintes 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 videó készítés közben mozgatni fogjuk a gömböket. Mivel csak gömb alakú objektumokkal 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 mint amennyit 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 Modulok és Fájlok

A program mostantól moduláris felépítésű, azaz különválasztottuk a deklarációkat (header fájlok, .h) és az implementációkat (forrásfájlok, .c). Ez a modern C fejlesztés szabványait követi, és lehetővé teszi a külön fordítást (separate compilation). A fordítást egy `Makefile` vezérli.

4.1 Makefile

Ez a fájl tartalmazza a fordítási szabályokat. A `mingw32-make` (vagy `make`) parancs kiadásával automatikusan lefordulnak a módosított modulok, majd a linkelés után létrejön a `video_render.exe`.

4.2 vec_3 (vec_3.h, vec_3.c)

Mivel 3 dimenzióban dolgozunk szükségünk van vektoroperációkra. A header fájl tartalmazza a függvények prototípusait és a `vec_3` struktúrát. Az implementáció a `.c` fájlban található. Ez biztosítja a műveleteket (összeadás, szorzás, dot product, stb.) és a random vektor generálást.

4.3 ray (ray.h, ray.c)

Ez a modul a sugár (ray) struktúrát és a sugárkövető függvényt (`_pos()`) tartalmazza.

4.4 intervals (intervals.h, intervals.c)

Intervallumokkal végzett műveletek (tartalmazás vizsgálata). Segít az átláthatóságban, például a gömbök eltalálásakor.

4.5 sphere (sphere.h, sphere.c)

A gömb objektum megvalósítása. Ez a modul a gömb struktúrát hozza létre. A gömb egy középpontból és egy sugárból áll. Ezután a `hit_sphere()` függvény következik ami, lényegében 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 elmenti 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. Utóbbi pedig (bizonyos korlátok kötött) random paraméterekkel rakja le a gömböket.

4.6 hittable (hittable.h, hittable.c)

Definiálja a `hit_rec` struktúrát (találati rekord), 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 tartalmazza 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 találjuk el.)

4.7 hit_list (hit_list.h, hit_list.c)

Ez a modul 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.

4.8 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.

4.9 common (common.h, common.c)

Általános segédfüggvények, például a program elején megjelenő felirat (`printCaption()`).

4.10 animation (animation.h, animation.c)

Az animációért felelős modul. Oszcilláció (`harm_osc_y()`) és véletlen mozgás (`random_walk()`) implementációk.

4.11 render (render.h, render.c)

A renderelés magja. 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 egy 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 újra (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 végül a visszakapott értéket megszorozza a `reflection_number`-rel (lényegében az, hogy a színnek hányad részét adja vissza). Amennyiben nem volt találat a háttér színét adja vissza. Ha elértük a rekurziós számnyi "pattogást" akkor feketét adunk vissza.

Ezt egy rövid függvény követi; `rand_square()` ami 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.

4.12 video_render.c

Ez a fő fájl ahol végül maga a renderelés megtörténik. Kiírjuk a kis szöveget, a `printCaption()` függvénnyel. Utána bekérjük a felhasználótól, hogy hány pixelt szeretne látni vízszintesen. Létrehozzuk a nézőportál és a renderelés többi paraméterét, majd ezeket berakjuk a `viewport_creator()`-ba ami átírja a szükséges paraméterek értékét. Ezután megkérdezzük a felhasználót, hogy hány gömböt szeretne. Létrehozunk egy `sphere` dinamikus tömböt aminek a mérete eggyel több lesz mint a felhasználó által megadott érték, mivel a tömbben az első gömb (egy meghatározott paraméterű gömb) a földnek feleltethető meg. Közben ellenőrizzük, hogy sikerült-e létrehozni a dinamikus tömböt. Ha megvan a dinamikus tömb, akkor meghívjuk a `_rand_spheres()` függvényt, hogy a többi elemet feltöltse gömbökkel. Ezután a reflekciós számot is bekérjük a felhasználótól ("Milyen vilagosak legyenek a gombok?"). Végül pedig bekérjük a videó hosszát, és abból kiszámoljuk, hogy hány képkockát kell generáljunk (30 fps-en futnak a videók). Minden bekérésnél ellenőriztük, hogy a felhasználó által megadott adat formátuma helyes volt-e és, hogy a megfelelő határokon belül adta-e meg az értékeket. Amennyiben nem hibaüzenetet küldünk, és várjuk, hogy új adatot adjon meg. A render elkezdése előtt csinálunk egy kis vizuális sávot ahol lehet követni, hogy hogy áll a program.

Ezután egy for ciklussal elindulunk, és minden képkockához nyitunk egy új fájlt. Itt az `sprintf()` függvényt használjuk, hogy a fájlneveknek csak a "sorozatszáma" változzon. Ellenőrizzük, hogy sikerült-e megnyitni a fájlt. Ha nem kilépünk, ha igen minden kép elejére beírunk egy-két paramétert ami a ppm formátum miatt használatos.:

```
fprintf(fp, "P3\n%d␣%d\n255\n", image_width, image_height);
```

Utána meghívjuk a `render()` függvényt, a megfelelő paraméterekkel. A vizuális sávra kiírunk egy '/' jelet ha a program az előző kiíráshoz képest legalább 2 százalékot haladt előre. (Ez itt nem működne rövid videókra, de olyan rövidet nem adhat meg a felhasználó.) Ezután meghívjuk a `harm_osc_y()` függvényt és kicsit odébb rakjuk a gömböket. Ha az egész képgenerálás kész kiírjuk hogy Kész. Utána ffmpeg-gel összefűzzük a képkockákat videóvá, töröljük a képkockákat (ha a felhasználó futás közben megszakítja a programot akkor nem törlődnek a képkockák, újabb teljes futásnál viszont már egyben kitörli a program a képkockákat.). Végül lejátsszuk a videót. Itt

használtuk a `system()` parancsot, hogy el tudjuk végezni a fönt említett folyamatokat. Legutoljára pedig felszabadítjuk a dinamikusan allokált tömböt.

Sok sikert és jó szórakozást a program próbálgatásához!