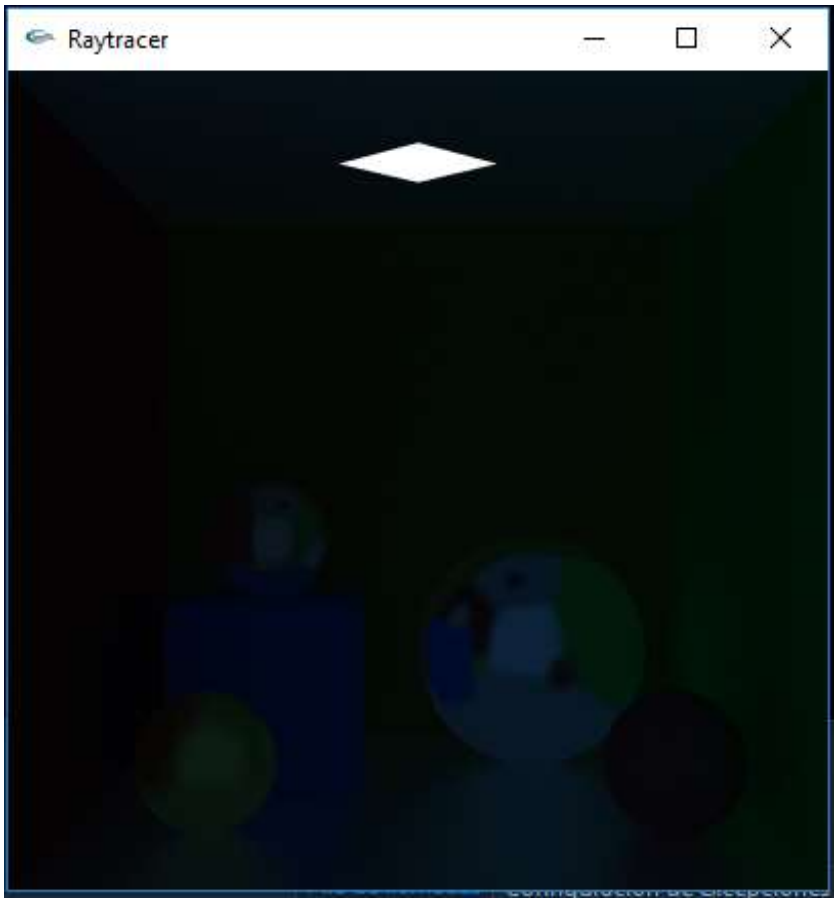
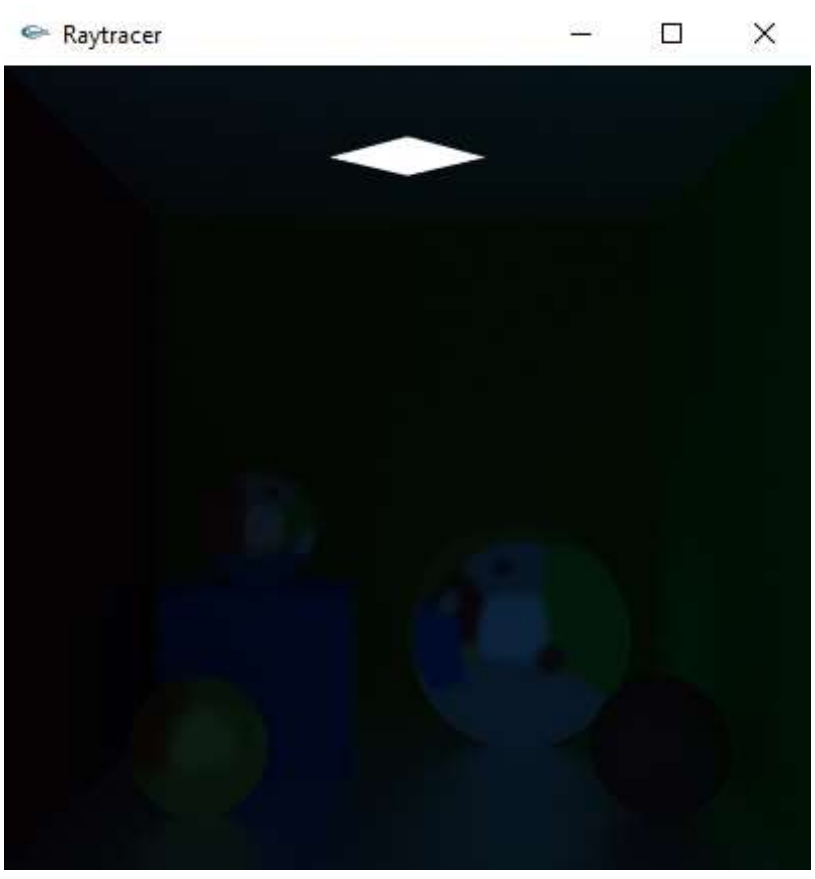
Idea 1: Create the scene using indirect lighting only (i.e. no importance sampling of lights) How many rays do you need to get a reasonable result?

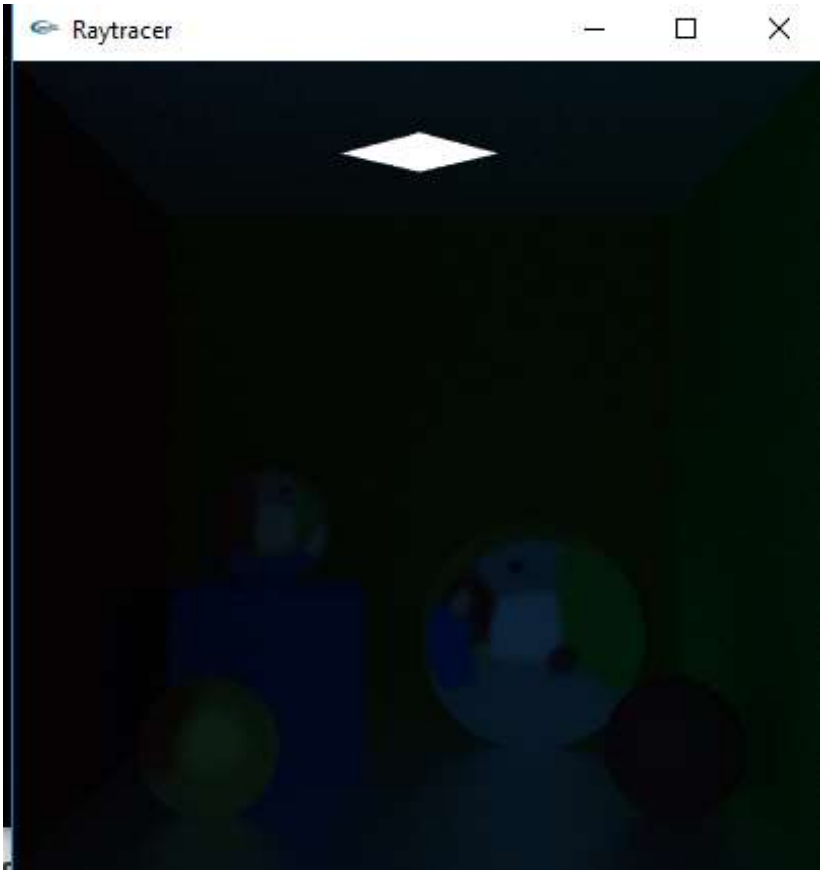


Con 100 rayos se ve así

Tiempo aprox. 40 min



Esta imagen es con 50 rayos tiempo aprox. 22 min.



Con 15 rayos se ve así

Tiempo aprox. 2min

Con 5 rayos

Tiempo aprox. menos de un minuto

Para hacer esta parte del código he añadido un booleano en un if que mira si calculamos o no el color difuso.

Como podemos ver con 5 rayos y 15 aún no se ve del todo bien la imagen. Por lo contrario con 50 y 100 se ven casi iguales, la diferencia es que con 100 rayos tardamos el doble en conseguir la imagen. Para conseguir un podríamos poner un valor entre 15 y 50 rayos por pixel ya que desde 15 se ve de una manera aceptable y tarda muy poco y con 50 tarda unos 20 minutos pero se ve mejor, y el valor que pondríamos dependería de si queremos que se vea mejor se acercara a 50 o que salga más rápido se acercara a 15. Intente probar 1000 rayos por pixel pero iba muy muy lento puede que tardara varios días en acabar y el esfuerzo para el ordenador era muy grande.

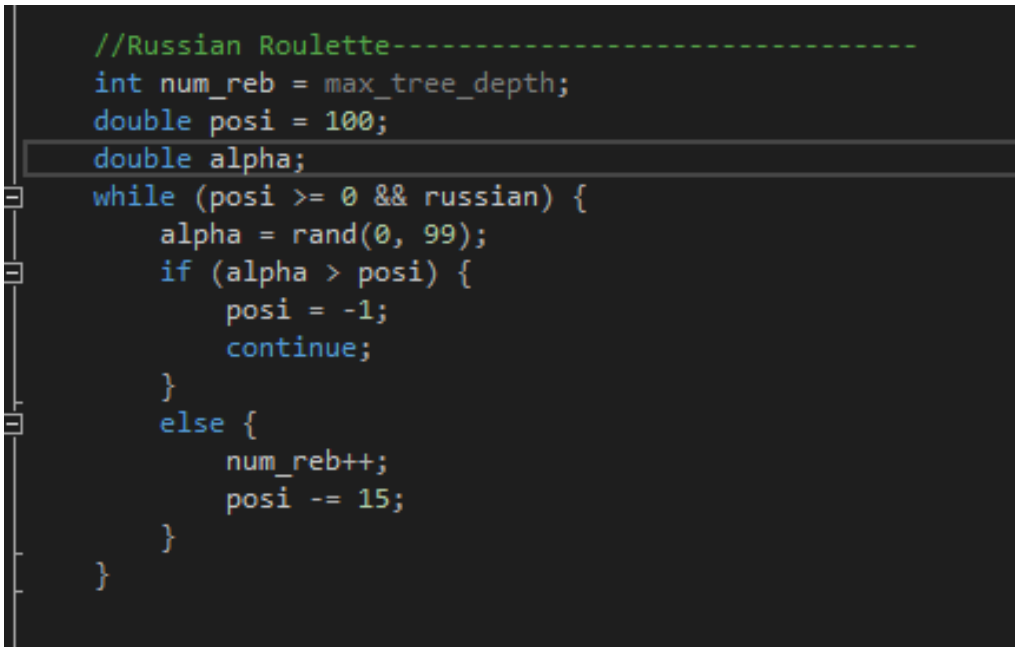
Idea 2: Given a uniform random variable u $\in$[0,1], material diffuse color d k and material specular color s k .

- u $<$ kd : take a diffuse sample and compute its contribution
- kd $\leq$ u $<$ kd $+$ ks : take a specular sample and compute its contribution
- kd $+$ ks $\leq$ u : the contribution is 0

```
    }
    Color indirect;
    if ((contriD + contriS) > u) {
        Sample S1 = SampleProjectedHemisphere(N);
        Ray rayo;
        rayo.no_emitters = false;
        rayo.origin = hit.geom.point + N*Epsilon;
        rayo.direction = S1.P;
        Color indirect_diff;
        if ((u < contriD)) {
            indirect_diff = S1.w * hit.material.m_Diffuse / Pi * Trace(rayo, scene, num_reb);
        }

        Color indirect_spec;

        Sample S2;
        if ((hit.material.m_Phong_exp > 0) && (contriD <= u < (contriD + contriS))) {
            Ray rayo1;
            Vec3 ref = Reflection(V, N);
            rayo1.no_emitters = false;
            rayo1.origin = hit.geom.point + Epsilon*N;
            S2 = SampleSpecularLobe(ref, hit.material.m_Phong_exp);
            rayo1.direction = S2.P;
            indirect_spec = S2.w*hit.material.m_Specular*((hit.material.m_Phong_exp + 2) / (2 * Pi))*Trace(rayo1, scene,
        }


        indirect = indirect_diff + indirect_spec;
    }
```

Creamos una variable u que sea aleatoria entre 0 y 1, luego calculamos la contribución de los colores en la indirecta diffusa y especular. Se calcula sumando la contribución del rgb de la difusa y especular y dividiéndola entre 3. Entonces se la u es más pequeña que las suma de las contribuciones se calcula la indirecta. Si la u es más pequeña que la contribución de la difusa se suma la difusa y si la u es más grande que la difusa y más pequeña que la suma de la difusa y la especular se suma la especular.

Idea 4: Use “Russian Roulette” to compute tree depth. Assign ‘weight’ to each ray, – only continue tracing rays iff weight < const α. Then divide weight by alpha after each trace.



```
//Russian Roulette---------------------------------
int num_reb = max_tree_depth;
double posi = 100;
double alpha;
while (posi >= 0 && russian) {
    alpha = rand(0, 99);
    if (alpha > posi) {
        posi = -1;
        continue;
    }
    else {
        num_reb++;
        posi -= 15;
    }
}
```

Implementamos el Russian Roulette (o más bien lo intentamos), ponemos el num_reb = max_tree_depth, la posibilidad la ponemos al 100%(posi = 100). En el alpha hacemos el random que va de 0 a 99, no llega a 100 para asegurar que el primero rebote.

Entonces si la posibilidad es positiva entramos en un while para ver cuántas veces rebota el rayo (el bool russian sirve para desactivar o activar el russian roulette). Si alpha es más grande que random no rebotara más por lo que salimos del bucle. En caso contrario restaremos 15 a la posibilidad de rebote y volveremos a hacer el bucle. Luego en vez de pasar el max_tree_depth a las funciones, le pasamos el num_reb.

El problema que tuvimos es que a veces rebotaba 13 veces cosa que no es posible ya que como máximo rebotara 6 veces (porque restamos 15 cada iteración)