Fondamenti di Computer Graphics M (8 cfu)

Lab 4 - Ray tracing

Michele Righi

Settembre 2023

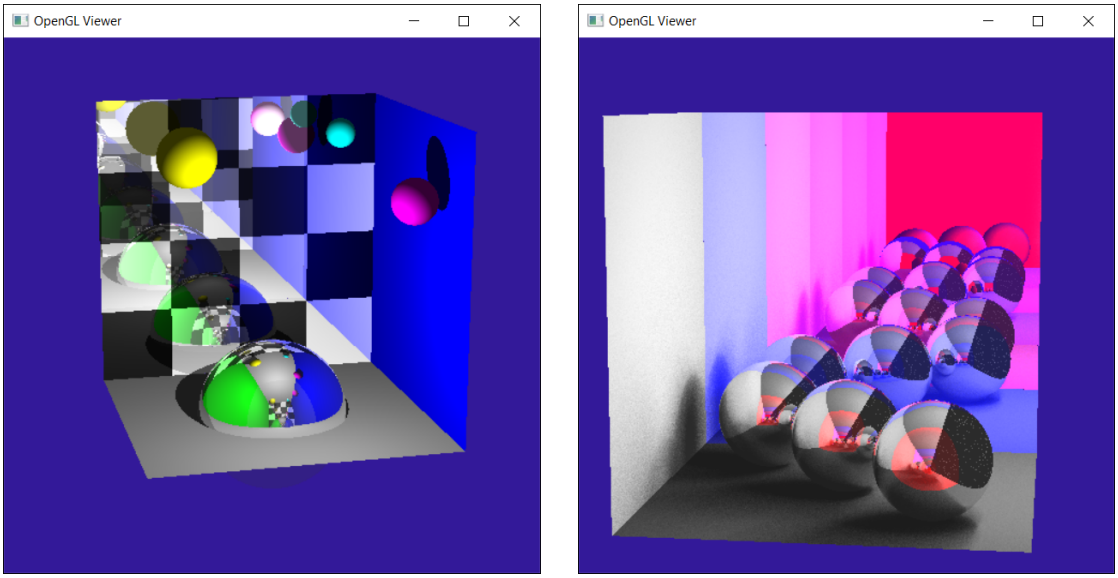


Figura 1: Demo Ray Tracing

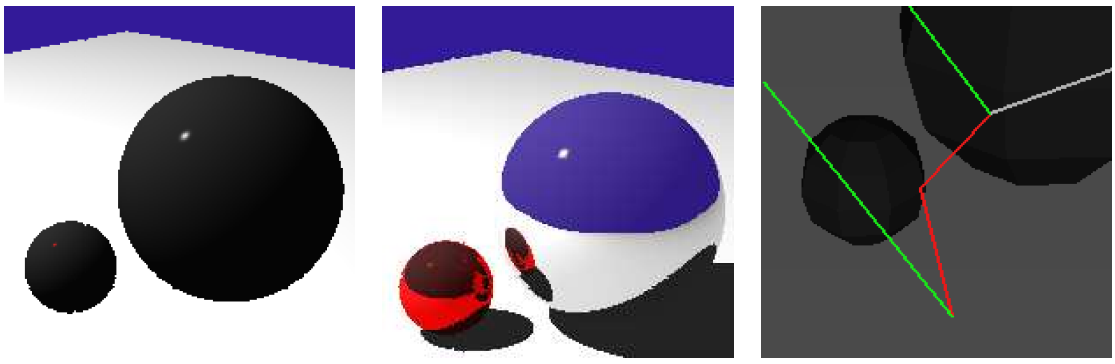
Traccia

L'obiettivo dell'esercitazione, una volta familiarizzato con l'ambiente ed esplorato le potenzialità dell'algoritmo, è di:

1. sviluppare la gestione degli shadow rays per la generazione di hard shadows;
2. sviluppare la gestione ricorsiva dei reflection rays;
3. implementare una strategia per la gestione di soft shadows mediante risorse luminose ad area anziché puntiformi, da attivare da linea di comando con

```
-soft_shadow
```

   Nella versione attuale le luci in scena sono già area lights implementate come quads con emissione non-zero, ma se ne considera solo il baricentro per gestirle come puntiformi. Si devono considerare raggi multipli verso più punti della risorsa luminosa area light. Discutere la strategia scelta per selezionare i punti sull'area light.



(a) (b) (c)

Figura 2: (a) l'output dell'applicazione con rendering tramite ray casting; (b) il rendering è eseguito tramite ray tracing, dopo aver implementato la gestione degli shadow rays e dei reflection rays; (c) la modalità di debug con visualizzazione dei raggi generati.

1 Introduzione

1.1 Ray Tracing

Il ray tracing si basa su un algoritmo di ray casting secondo cui, al fine di assegnare un colore per ogni pixel della viewport/finestra, vengono fatto partire un raggio per ciascun pixel e ne viene tracciato il percorso. Il raggio può non colpire nulla (in questo caso al pixel verrà assegnato il colore dello sfondo), oppure colpire una sorgente luminosa o degli oggetti, eventualmente riflettenti. Nel caso di oggetti riflettenti, viene fatto partire un raggio dal punto in cui l'oggetto è stato colpito, nella direzione riflessa, e il procedimento si ripete.

Dunque, il colore degli oggetti in scena è dato dai raggi luminosi che lo colpiscono e vengono riflessi verso la camera, e dai raggi indiretti che rimbalzano dagli altri oggetti.

1.2 Progetto

Per questa esercitazione è stato fornito un progetto con diversi file e classi di utilità per il ray tracing. Il file `glCanvas.cpp` contiene la classe che si occupa di disegnare la scena sul canvas, in particolare, il metodo `TraceRay(float i, float j)` si occupa di ottenere il colore di ciascun pixel, richiamando l'algoritmo di ray tracing:

```
Vec3f GLCanvas::TraceRay(float i, float j) {
    int max_d = max2 (args->width, args->height);
    float x = (i + 0.5 - args->width / 2.0) / float (max_d) + 0.5;
    float y = (j + 0.5 - args->height / 2.0) / float (max_d) + 0.5;

    Ray r = camera->generateRay (Vec2f (x, y));
    Hit hit;
    Vec3f color = raytracer->TraceRay (r, hit, args->num_bounces);
return color;
}
```

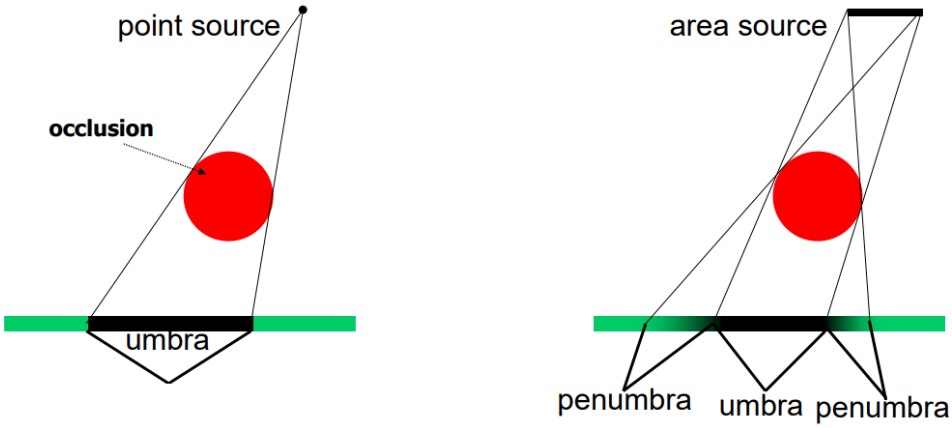
Il codice che implementa la logica di ray tracing, e dunque la soluzione richiesta, è contenuto nel file `raytracer_es.cpp`:

```
// Casts a single ray through the scene geometry and finds the closest hit
bool CastRay(Ray& ray, Hit& h, bool use_sphere_patches) const;

// Does the recursive work
Vec3f TraceRay(Ray& ray, Hit& hit, int bounce_count = 0) const;
```

1.3 Hard shadow e Soft shadow

La differenza fra *hard* shadow e *soft* shadow è dovuta alla conformazione della sorgente luminosa:

- hard shadows, sono generate da fonti luminose **puntiforme** (Fig. 3a);
- soft shadows, sono generate da fonti luminose **ad area** (Fig. 3b);



(a) Hard shadow

(b) Soft shadow

Figura 3: Tipi di Ombre

2 Implementazione

2.1 Shadow Rays per Hard Shadows

Le luci in scena contribuiscono al colore e l'intensità di una superficie se e solo se ne raggiungono l'oggetto, ma questo potrebbe essere occluso da altri oggetti in scena. In tal caso, viene generata un'ombra.

Per disegnare un'ombra, ci interessa sapere qual è il **contributo luminoso di ciascuna luce** in scena per ogni punto.

Dunque, quando il raggio castato tramite `RayTracer::TraceRay()` colpisce un qualsiasi oggetto in un punto, è necessario calcolare i contributi di ciascuna sorgente luminosa per quel determinato punto: se una fonte luminosa contribuisce all'illuminazione di tale punto, allora viene considerata e aggiunta nel calcolo della luminosità.

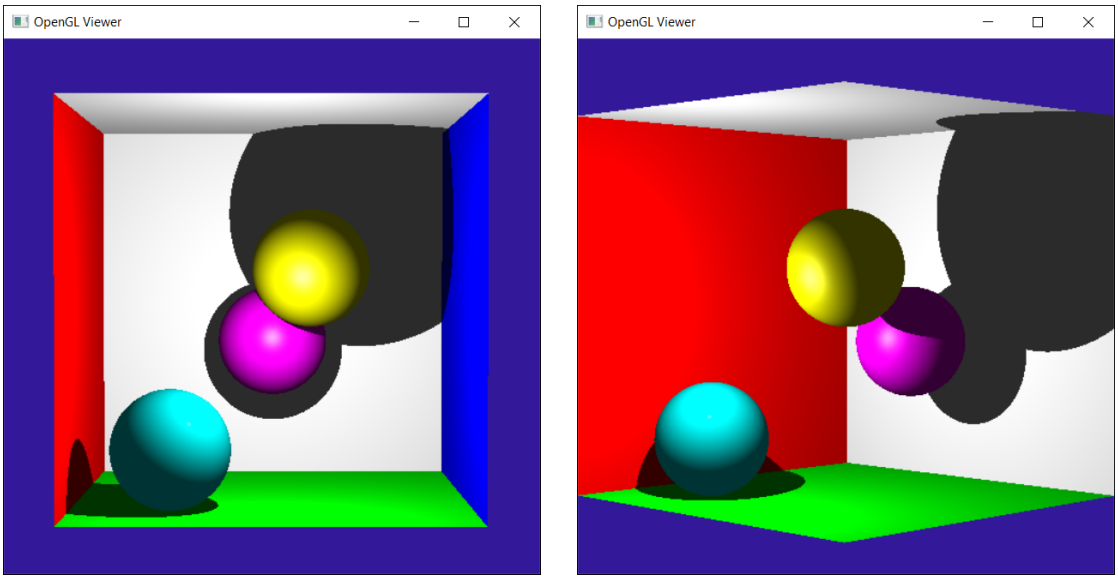
Contributi luminosi Come facciamo a capire quali fonti luminose contribuiscono all'illuminazione di un punto? Ci basta castare un raggio (*shadow ray*) dal punto colpito tramite ray tracing, verso la fonte luminosa. Se il primo oggetto che tale raggio colpisce è la fonte luminosa, allora il punto vede in linea diretta la sorgente luminosa, dunque questa contribuisce all'illuminazione. Altrimenti no, e potrebbe (se il punto non è illuminato da altre sorgenti) essere generata un'ombra.

Implementazione Iteriamo su tutte le luci presenti in scena (`mesh->getLights()`) e per ciascuna di queste creiamo uno `shadow_ray` che parte dal punto sull'oggetto colpito ed è diretto verso la luce i-esima. Dunque controlliamo a che distanza, rispetto alla fonte luminosa, si trova il primo oggetto colpito da tale raggio:

- se la distanza è circa zero (`< 0.01`), si tratta della fonte luminosa i-esima, e dobbiamo considerare il contributo luminoso di questa (il punto considerato "vede" la fonte luminosa);
- altrimenti, significa che tra la luce e il punto si trova un oggetto terzo, dunque il contributo luminoso della luce i-esima viene scartato per questo punto.

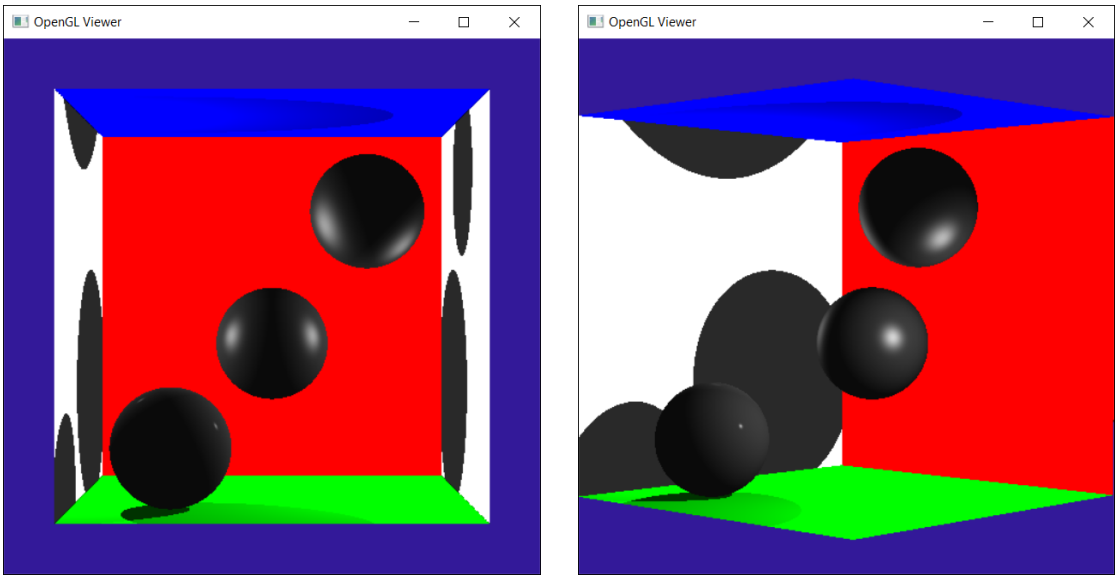
```
1  Vec3f RayTracer::TraceRay(Ray& ray, Hit& hit, int bounce_count) const
2  {
3      // [...]
4
5      Hit* new_hit;
6      bool colpito;
7      Ray* shadow_ray;
8      Vec3f n_point, dista, pointOnLight;
9      int num_lights = mesh->getLights().size();
10
```

```
    // Iteriamo su ciascuna luce in scena
    for (int i = 0; i < num_lights; i++)
    {
        Face *f = mesh->getLights()[i];
        pointOnLight = f->computeCentroid();
        Vec3f dirToLight = pointOnLight - point;
        dirToLight.Normalize();

        // Creiamo uno shadow ray verso il punto luce
        shadow_ray = new Ray(point, dirToLight);

        // Casto il raggio e controllo il primo oggetto colpito
        new_hit = new Hit();
        colpito = CastRay(*shadow_ray, *new_hit, false);
        if (colpito)
        {
            n_point = shadow_ray->pointAtParameter(new_hit->getT());

            // Controllo se l'oggetto colpito è la fonte luminosa i-esima
            dista.Sub(dista, n_point, pointOnLight);
            if (dista.Length() < 0.01)
            {
                // Calcolo il contributo locale della luce i-esima sul punto
                if (normal.Dot3(dirToLight) > 0)
                {
                    // Aggiungiamo il contributo della luce
                    Vec3f lightColor = 0.2 * f->getMaterial()->getEmittedColor() * f->getArea();
                    answer += m->Shade(ray, hit, dirToLight, lightColor, args);
                }
            }
            // else: la luce i-esima non contribuisce alla luminosità di point.
        }
    }
}
```



(a) Vista frontale (b) Vista prospettica

Figura 4: Hard shadows di sfere, con singola fonte luminosa frontale, puntiforme.



(a) Vista frontale (b) Vista prospettica

Figura 5: Hard shadows di sfere, con doppia fonte luminosa (una a sinistra e una a destra). Possiamo notare che per ciascun oggetto vengono generate due ombre.

2.2 Gestione Ricorsiva dei Reflection Rays

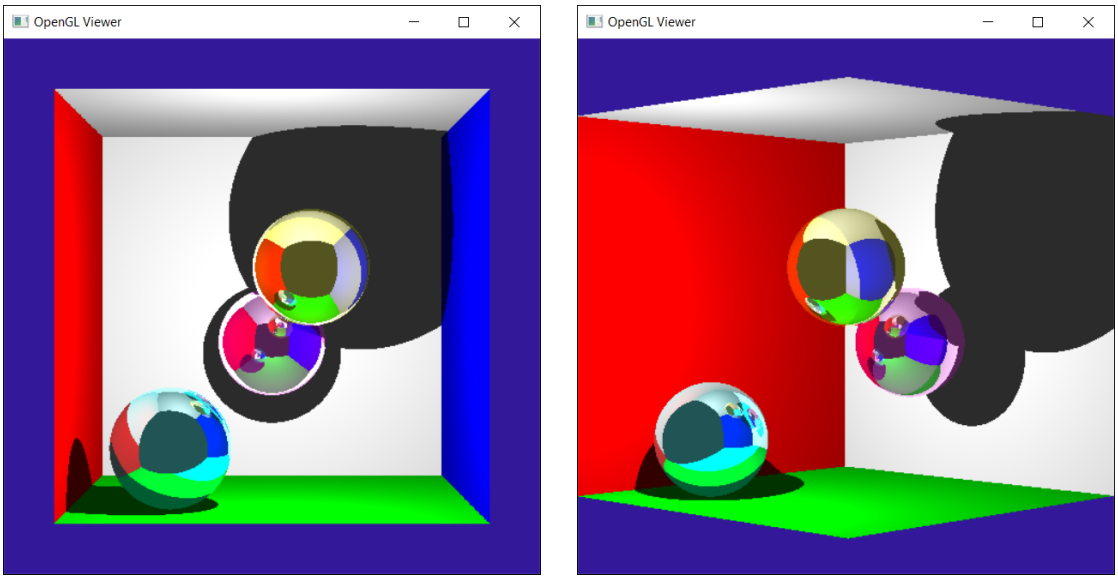
La luce può arrivare al campo visivo, oltre che da fonti luminose, anche tramite percorsi più complessi, ad esempio rimbalzando su oggetti con superfici speculari. Per considerare i contributi luminosi che arrivano da questi oggetti, è necessario (nel ray tracing) utilizzare dei *reflection rays*.

La logica di reflection viene applicata prima di quella delle ombre: quando il raggio castato tramite ray tracing colpisce un oggetto, se questo è riflettente (ovvero il suo materiale possiede il vettore della proprietà riflettente non nullo), viene creato un sotto-raggio. Tale raggio avrà origine nel punto in cui l'oggetto è stato colpito dal raggio precedente, e una direzione data dalla *reflection direction*, calcolabile tramite la seguente formula:

$$r_v = v - 2(n \cdot v)n$$

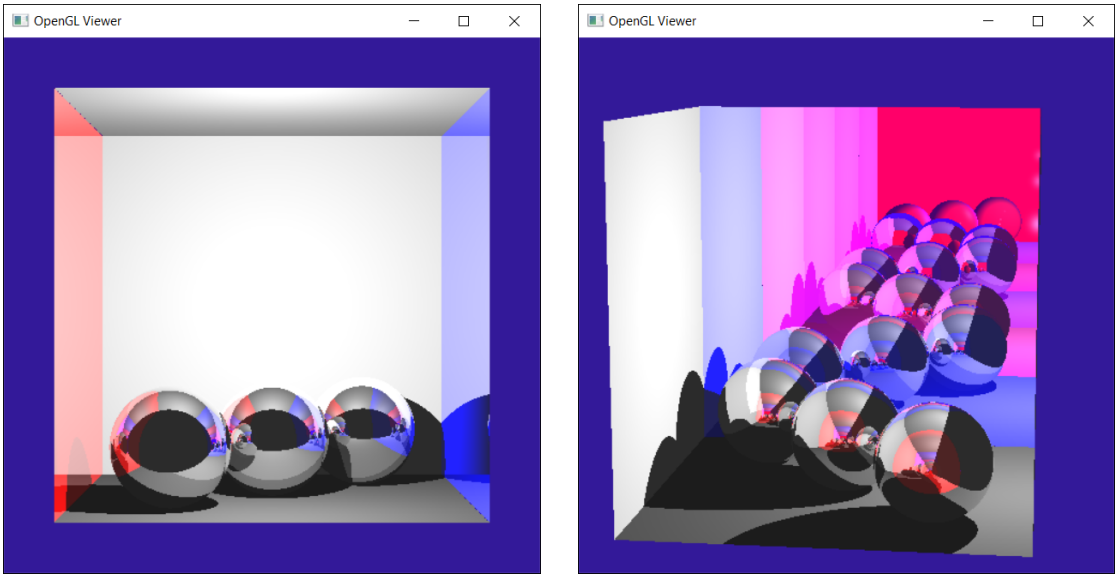
Implementazione Aggiungere la logica dei reflection rays non è complessa: è necessario controllare se l'oggetto colpito possiede proprietà riflettenti e se non abbiamo già raggiunto il numero massimo di rimbalzi da considerare (`bounce_count`). In questo caso, facciamo una chiamata ricorsiva al metodo di ray tracing nella nuova direzione, decrementando il numero di rimbalzi di uno e aggiorniamo il contributo (in `answer`) con il suo risultato.

```
Vec3f RayTracer::TraceRay(Ray& ray, Hit& hit, int bounce_count) const
{
    // [...]

    // Se la superficie è riflettente e non abbiamo raggiunto il limite di rimbalzi
    if (reflectiveColor.Length() != 0 && bounce_count > 0)
    {
        Vec3f VRay = ray.getDirection();

        // Calcolare il reflection ray
        Vec3f reflectionRay = VRay - (2 * VRay.Dot3(normal) * normal);
        reflectionRay.Normalize();
        Ray* new_ray = new Ray(point, reflectionRay);

        // Aggiunta del contributo in modo ricorsivo
        answer += TraceRay(*new_ray, hit, bounce_count - 1) * reflectiveColor;
    }

    // Shadow ray logic
}
```



(a) Vista frontale (b) Vista prospettica

Figura 6: Reflection rays su sfere colorate. Possiamo notare che il colore riflesso è influenzato dal colore degli oggetti riflettenti (azzurro, magenta, giallo).



(a) Vista frontale (b) Vista laterale

Figura 7: Reflection rays su sfere e pareti laterali colorate. In questo esempio in particolare, possiamo notare il progredire dei rimbalzi dei raggi, fino all'ultimo livello (pari a 5).

2.3 Shadow Rays per Soft Shadows

La differenza principale tra hard shadows e soft shadows, è dovuta al fatto che le ultime sono generate da sorgenti luminose ad area.

A livello algoritmico, la differenza principale risiede nel fatto che, una volta colpito l'oggetto tramite ray tracing, invece di castare un singolo shadow ray verso la fonte luminosa (come accadeva per le hard shadows), per le soft shadows ne vengono mandati molteplici, verso punti randomici contenuti nell'area della luce.

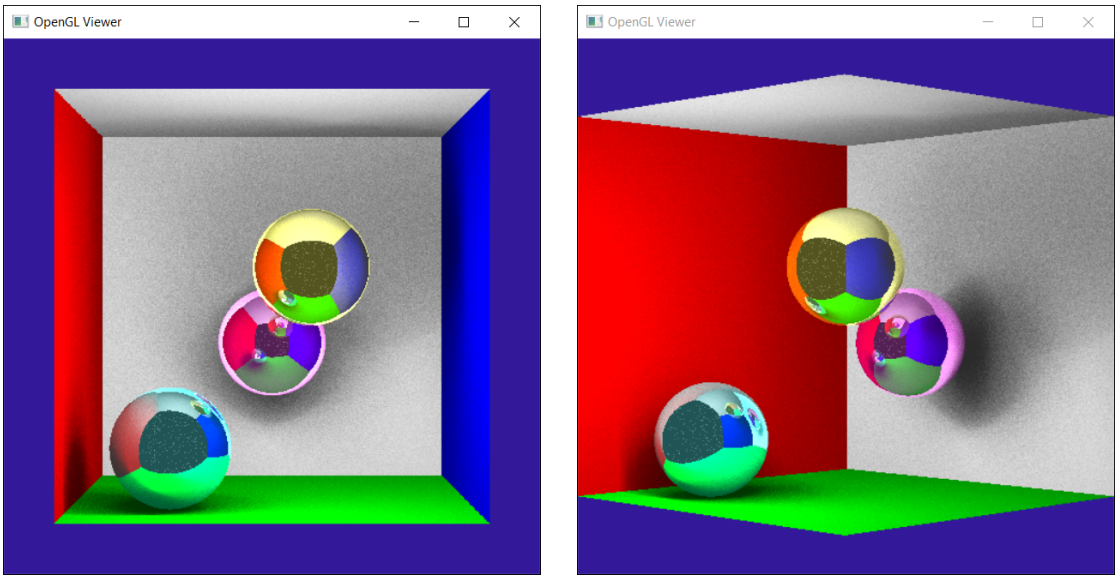
Ovviamente, l'utilizzo di soft shadows calcolate in questo modo comporta un rendering molto più pesante computazionalmente, che richiede più tempo, proporzionalmente al numero di shadow ray castati.

Implementazione Il codice che implementa la logica delle soft shadows differisce da quello delle hard shadows solo per due aspetti:

- Poiché non consideriamo più la fonte luminosa come puntiforme, non usiamo più il metodo `comuteCentroid()`. Al contrario, poiché ci interessa ottenere un insieme di punti randomici, sulla superficie della fonte luminosa, verso cui castare gli shadow rays, utilizziamo il metodo `RandomPoint()`. Per avere un'approssimazione sufficiente, impostiamo il numero di punti randomici pari a 100.
- Poiché gli shadow rays non si occupano solo di disegnare le ombre, ma contribuiscono all'illuminazione degli oggetti, se incrementiamo il numero di shadow rays, aumenta anche l'illuminazione della scena. Per mantenere un'illuminazione adeguata, indipendentemente dal numero di shadow ray castati, dividiamo il coefficiente moltiplicativo nel calcolo del `lightColor` per il numero di shadow rays (dato da `num_points`).

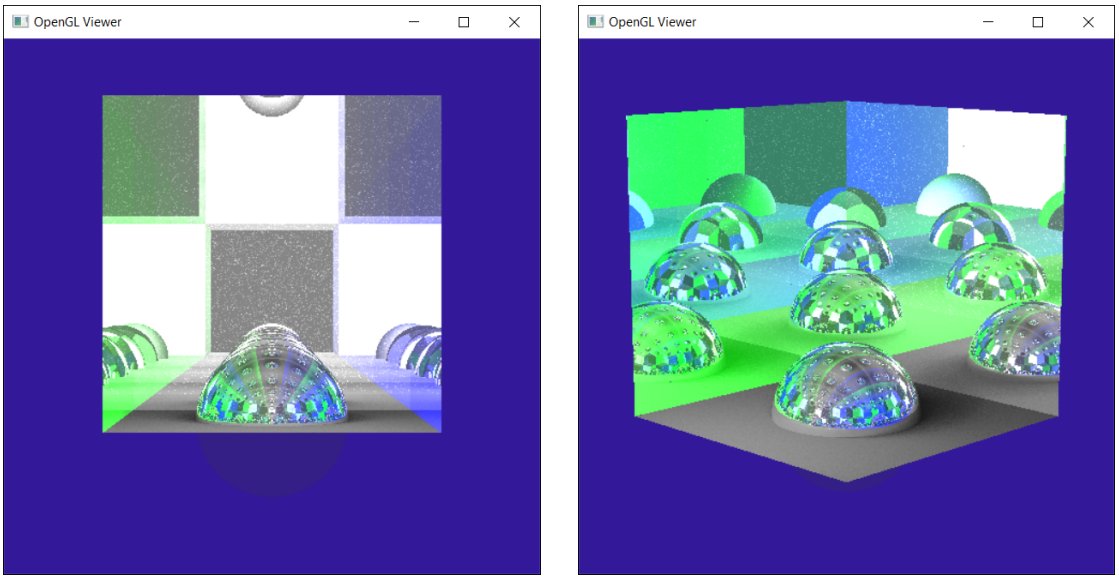
```
1  Vec3f RayTracer::TraceRay(Ray& ray, Hit& hit, int bounce_count) const
2  {
3      // [...]
4
5      if (args->softShadow)
6      {
7          int num_points = 100;
8
9          for (int i = 0; i < num_lights; i++)
10         {
11             Face* f = mesh->getLights()[i];
12
13             std::vector<Vec3f> pointsOnLigh;
14             std::vector<Vec3f> directionsToLight;
15
16             // Random points on area light
```

```
            for (int j = 0; j < num_points; j++)
            {
                pointsOnLigh.push_back(f->RandomPoint());
                Vec3f dirToLight = pointsOnLigh.at(j) - point;
                dirToLight.Normalize();
                directionsToLight.push_back(dirToLight);
            }

            for (int j = 0; j < num_points; j++)
            {
                shadow_ray = new Ray(point, directionsToLight.at(j));
                new_hit = new Hit();
                colpito = CastRay(*shadow_ray, *new_hit, false);

                if (colpito)
                {
                    n_point = shadow_ray->pointAtParameter(new_hit->getT());
                    dista.Sub(dista, n_point, pointsOnLigh.at(j));

                    if (dista.Length() < 0.01 && normal.Dot3(directionsToLight.at(j)) > 0)
                    {
                        Vec3f lightColor = 0.2 / num_points *
                            f->getMaterial()->getEmittedColor() * f->getArea();
                        answer += m->Shade(ray, hit, directionsToLight.at(j), lightColor, args);
                    }
                }
            }
        }
    }
}
```



(a) Vista frontale (b) Vista prospettica

Figura 8: Soft shadows di sfere colorate, con fonte luminosa frontale.



(a) Vista frontale (b) Vista laterale

Figura 9: Soft shadows di una sfera riflettente, riflessa tramite pareti specchiate, con fonte luminosa frontale. È interessante notare come la metà inferiore della sfera non venga colorata, in quanto i raggi della luce non la colpiscono.