

Sviluppare la gestione degli shadow rays per la generazione di hard shadows e soft shadow

Dal punto colpito dal raggio principale, si crea uno shadow ray in direzione di ogni luce presente in scena, se questo colpisce per primo una luce, e il punto colpito è in direzione della luce, si aggiunge il contributo fornito dalla luce. `point` è il punto colpito dal raggio principale.

```

Face* f = mesh->getLights()[i];
Vec3f pointOnLight = f->computeCentroid();
Vec3f dirToLight = pointOnLight - point;
dirToLight.Normalize();

// creare shadow ray verso il punto luce, dal punto colpito
Ray* shadowRay = new Ray(point, dirToLight);
// controllare il primo oggetto colpito da tale raggio e verificare se e' una luce
Hit* hitShadow = new Hit();
bool colpito = CastRay(*shadowRay, *hitShadow, false);
if (colpito) {
    Vec3f puntoColpito = shadowRay->pointAtParameter(hitShadow->getT());
    // distanza dal punto colpito al punto della luce
    float dist = (puntoColpito - pointOnLight).Length();
    // se ho colpito la luce la distanza deve essere 0
    if (dist < 0.01)
        if (normal.Dot3(dirToLight) > 0)
    {
        Vec3f lightColor = 0.2 * f->getMaterial()->getEmittedColor() * f->getArea();
        answer += m->Shade(ray, hit, dirToLight, lightColor, args);
    }
}

```

Per ottenere un effetto *soft shadow*, invece di utilizzare un unico punto di destinazione per la luce, vengono utilizzati più punti. I punti vengono scelti in modo da formare una griglia sulla luce, per ogni punto di destinazione viene lanciato un raggio come fatto precedentemente, e il contributo aggiunto viene diminuito di un fattore $\frac{1}{\text{NUMERO DI PUNTI}}$.

```

Face *f = mesh->getLights ()[i];

int rowColumn = sqrt(NUM_RAY_TO_LIGHT);

for (int rowIndex = 0; rowIndex < rowColumn; rowIndex++) {
    for (int columnIndex = 0; columnIndex < rowColumn; columnIndex++) {
        Vec3f pointOnLight = f->getPoint(static_cast<float>(rowIndex) / rowColumn,
                                         static_cast<float>(columnIndex) / rowColumn);

```

```

Vec3f dirToLight = pointOnLight - point;
dirToLight.Normalize();
// creare shadow ray verso il punto luce, dal punto colpito
Ray* shadowRay = new Ray(point, dirToLight);
// controllare il primo oggetto colpito da tale raggio e verificare se e'S una luce
Hit* hitShadow = new Hit();
bool colpito = CastRay(*shadowRay, *hitShadow, false);
if (colpito) {
    Vec3f puntoColpito = shadowRay->pointAtParameter(hitShadow->getT());
    // distanza dal punto colpito al punto della luce
    float dist = (puntoColpito - pointOnLight).Length();
    // se ho colpito la luce la distanza deve essere 0
    if (dist < 0.01)
        if (normal.Dot3(dirToLight) > 0)
    {
        Vec3f lightColor = 0.2 * f->getMaterial()->getEmittedColor()
                                * (f->getArea() / (rowColumn * rowColumn));
        answer += m->Shade(ray, hit, dirToLight, lightColor, args);
    }
}
}

```

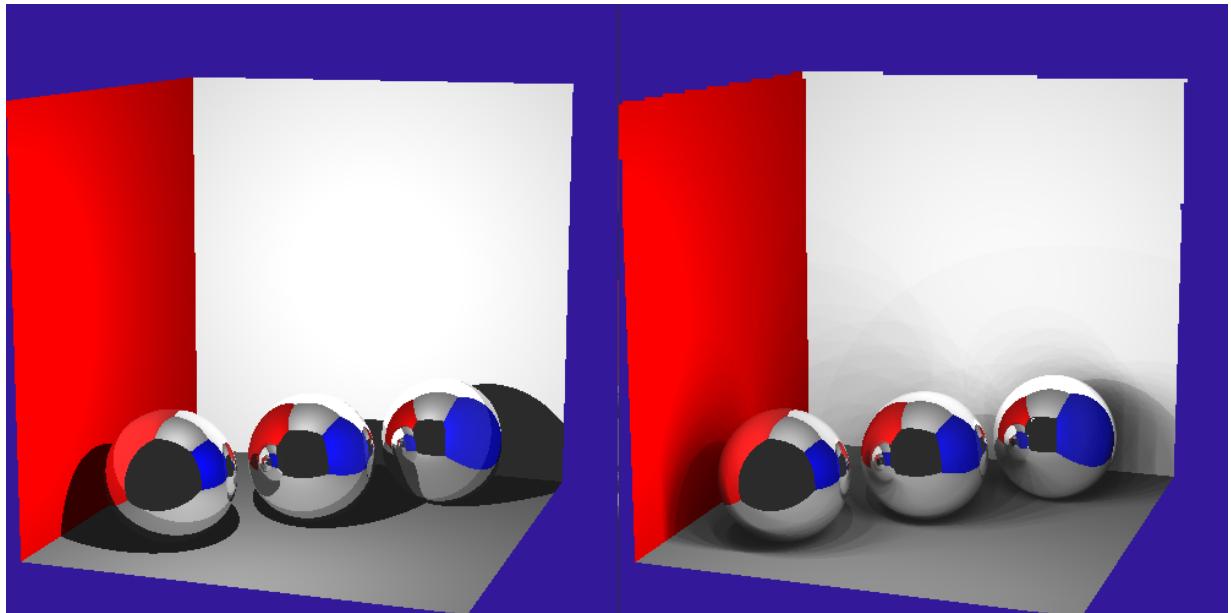


Figura 1: Sulla sinistra *hard shadow*, sulla destra *soft shadow*

Sviluppare la gestione ricorsiva dei reflection rays

Se con il raggio principale colpisco un punto, e il materiale del punto è riflettente, creo un altro raggio e aggiungo il suo contributo. Se con il nuovo raggio colpisco un altro oggetto riflettente la procedura continua fino al raggiungimento del numero di rimbalzi massimo.

```

/* se (il punto sulla superficie e' riflettente & bounce_count>0).
   per vedere se è riflettente guardo se la lunghezza del vettore > 0
   (almeno una componente diversa da 0) */
if (reflectiveColor.Length() > 0 && bounce_count > 0) {
    // calcolare ReflectionRay  R=2(normal,l)n - l
    Vec3f rayVector = ray.getDirection();
    Vec3f reflection = rayVector - (2 * rayVector.Dot3(normal) * normal);
    reflection.Normalize();
    Ray* newRay = new Ray(point, reflection);
    //   invocare TraceRay(ReflectionRay, hit,bounce_count-1)
    // aggiungere ad answer il contributo riflesso [moltiplicare per k_r]
    answer += TraceRay(*newRay, hit, bounce_count - 1) * reflectiveColor;
}

```

Digital Art: uso di Ray tracing in Blender

Come base di partenza è stata utilizzata la scena dell'esercitazione 4. In particolare nella scena sono stati aggiunti i seguenti elementi:

- Per lo scivolo sono è stata aggiunta una texture sulla parte anteriore e sui gradini e tramite il bump mapping si è cercato di far risaltare ulteriormente i dettagli.
- Per il tunnel è stato utilizzato un materiale semi trasparente con una parte diffusiva non visibile per far risaltare maggiormente la luce all'interno.
- La palla posizionata all'interno e all'esterno del tunnel, ha una texture e del bump mapping.
- Una pozza d'acqua realizzato tramite una combinazione di un materiale trasparente e riflettente, con del bump mapping per "simulare" la caduta di una goccia d'acqua. Il bump mapping utilizzata una texture generata tramite un'onda sinusoidale con della distorsione.
- Nella scena sono state posizionate diverse luci, alcune all'interno del tunnel di diversi colori, altre per illuminare lo scivolo, la palla e la giostra.

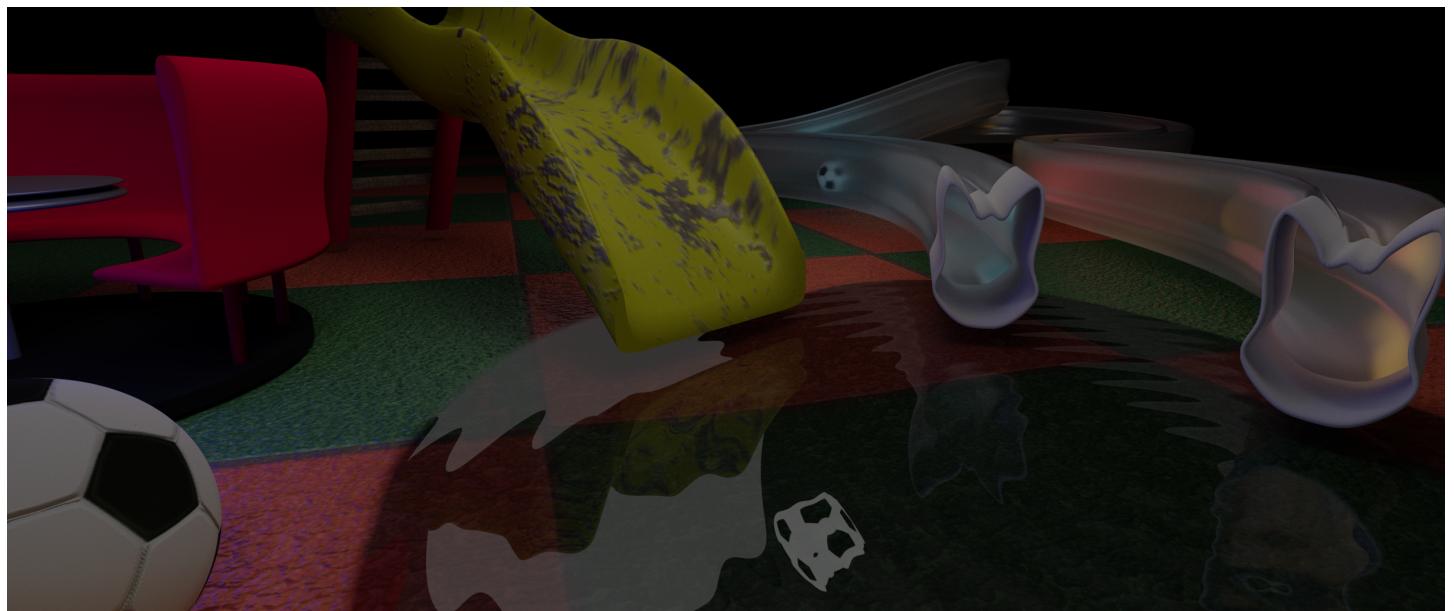


Figura 2: Scena finale