

LAB 4 – Ray Tracing & Digital Art
Fondamenti di Computer Graphics M
Erika Gardini – Ingegneria Informatica A.A. 2016/2017

Parte 1 – Ray Tracing

Per prima cosa è stato scaricato, compilato ed eseguito il programma fornito.
Al momento dell'esecuzione del programma si ha la seguente schermata:

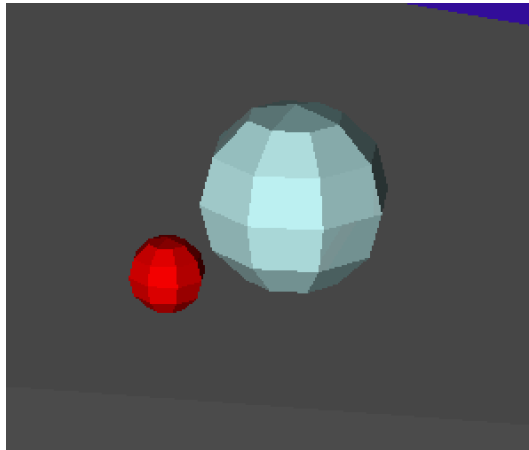


Figura 1: esecuzione del programma iniziale senza raytracing

La funzionalità di base già fornita è la seguente:

- avviamento del raytracing dalla posizione corrente della camera mediante la pressione del tasto 'r';
- modalità di debug che permette di visualizzare i vari raggi in scena mediante la pressione del tasto 't'.

A default sono realizzate soltanto le funzionalità di raycasting. In particolare, viene applicato a ciascun punto della scena il colore di background:

```
Vec3f answer = args->background_color;
```

Si aggiungono poi i contributi luminosi per tutti quei punti intersecati dai raggi della camera (anche detti raggi primari, uno per ogni pixel).

In particolare, si aggiunge il contributo della luce ambiente:

```
answer = args->ambient_light * m->getDiffuseColor ();
```

e tutti i contributi delle altre luci della scena:

```
int num_lights = mesh->getLights().size ();
for (int i = 0; i < num_lights; i++){
    Face *f = mesh->getLights ()[i];
    Vec3f pointOnLight = f->computeCentroid ();
    Vec3f dirToLight = pointOnLight - point;
    dirToLight.Normalize ();

    if (normal.Dot3 (dirToLight) > 0){
        Vec3f lightColor =
            0.2 * f->getMaterial()->getEmittedColor()*f->getArea ();
        answer += m->Shade(ray,hit,dirToLight,lightColor, args);
    }
}
```

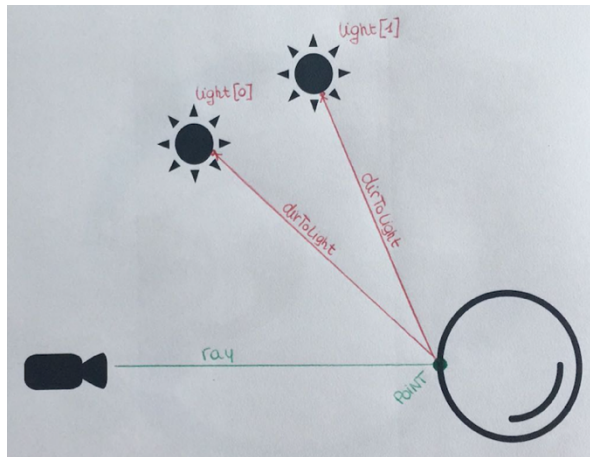


Figura 2: raycasting

A default, quindi, non vengono calcolate le ombre e le caratteristiche riflettenti dei vari oggetti nella scena. Eseguendo il programma si ottiene il seguente risultato:

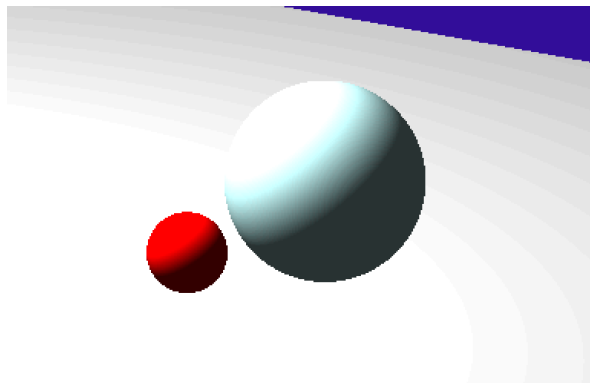
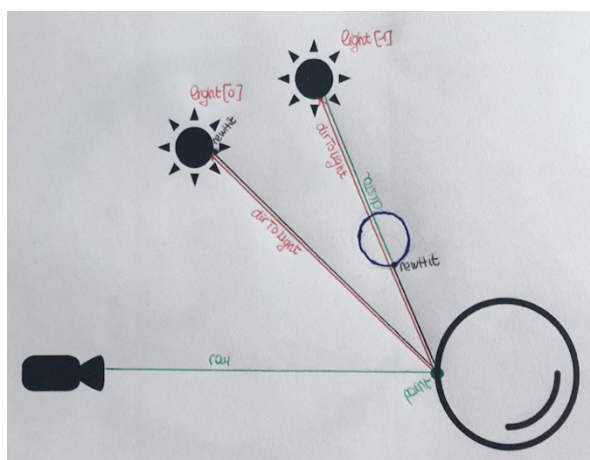


Figura 3: risultato ottenuto applicando il raycasting

Il primo punto dell'esercitazione richiedeva di realizzare e gestire gli shadow rays per la generazione degli hard shadows.

A tal scopo, è necessario aggiungere la logica riguardante il calcolo delle ombre.

In particolare, i contributi luminosi vanno aggiunti solo se tra una luce ed un oggetto non vi sono altri "ostacoli" nel mezzo:



Nell'immagine, per esempio, la luce "light[1]" non ha contributi sul punto "point", mentre la luce "light[0]" fornisce un contributo sul colore di point.

Un punto in ombra, quindi, è un punto che non viene colpito da nessun raggio di luce.

Con questa logica viene realizzato l'algoritmo per il calcolo delle ombre riportato qui di seguito:

```
int num_lights = mesh->getLights().size ();
for (int i = 0; i < num_lights; i++){
    Face *f = mesh->getLights ()[i];
    Vec3f pointOnLight = f->computeCentroid ();
    Vec3f dirToLight = pointOnLight - point;
    dirToLight.Normalize ();
    Ray *shadowRay = new Ray(point, dirToLight);
    Hit *newHit = new Hit();
    bool colpito = CastRay(*shadowRay, *newHit, 0);
    if(colpito){
        Vec3f n_point = shadowRay->pointAtParameter(newHit->getT());
        Vec3f dista;
        dista.Sub(dista, n_point, pointOnLight);
        if(dista.Length() < 0.01){
            if (normal.Dot3 (dirToLight) > 0){
                Vec3f lightColor =
                    0.2 * f->getMaterial()->getEmittedColor() *
                    f->getArea ();
                answer += m->Shade (ray, hit, dirToLight,
                    lightColor, args);
            }
        }
    }
}
```

Eseguendo il programma è stato quindi ottenuto il seguente risultato:

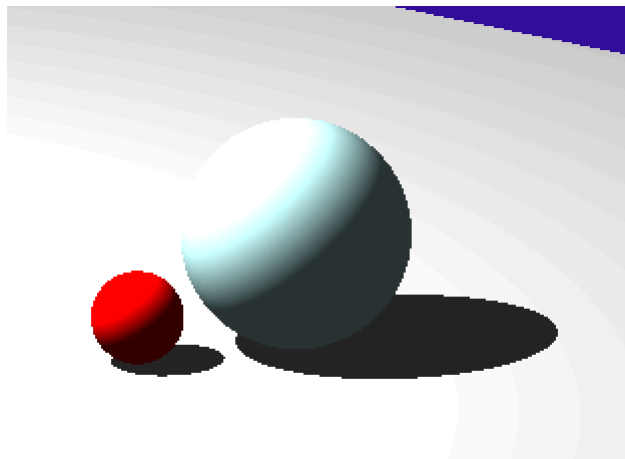
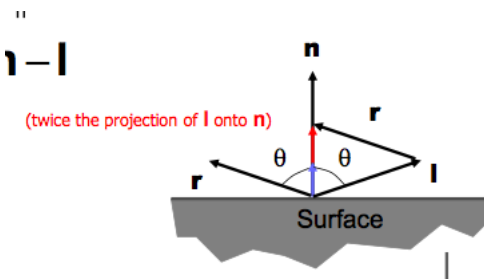


Figura 4: raytracing con shadow rays

Il secondo punto dell'esercitazione, invece, richiedeva di sviluppare la gestione ricorsiva dei reflection rays. Per realizzare questo punto occorre considerare che, se un oggetto è di materiale riflettente, quando la luce colpisce un suo punto, da quest'ultimo parte un ulteriore raggio di luce (raggio secondario) che potrebbe apportare contributi ad altri oggetti della scena.

Per calcolare il raggio riflesso occorre considerare la seguente regola:

$$r = 2(n \cdot l)n - l$$


Il raggio riflesso, potrebbe a sua volta colpire un oggetto di materiale riflettente creando un ulteriore raggio con contributi luminosi. Il procedimento è quindi ricorsivo e prosegue fino a che l'ultimo raggio riflesso non colpisce un oggetto non riflettente.

Nel caso dell'esercitazione, però, si è stabilito un certo numero di "bounce" da considerare per ciascun raggio riflesso, oltre il quale il procedimento ricorsivo veniva interrotto.

Qui si seguito si riporta la logica dell'algoritmo:

```
Vec3f reflectiveColor = m->getReflectiveColor ();
if(reflectiveColor.Length() != 0 && bounce_count > 0){
    Vec3f rayVector = ray.getOrigin();
    Vec3f reflection = (2*normal.Dot3(rayVector)*normal)-rayVector;
    reflection.Normalize();
    Ray ray = Ray(point, reflection);
    answer += TraceRay(ray, hit, bounce_count - 1);
    answer = answer * reflectiveColor;
}
```

Eseguendo il programma si ottengono i seguenti risultati:

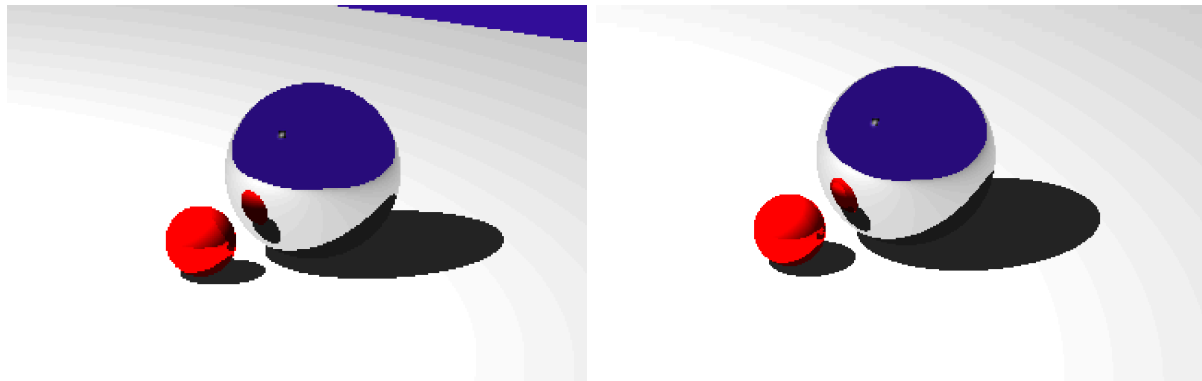


Figura 5: raytracing con reflection rays

Le immagini sopra riportate sono state ottenute rispettivamente con 1 e poi 2 bounce.

Passando in modalità debug si ottiene il seguente risultato:

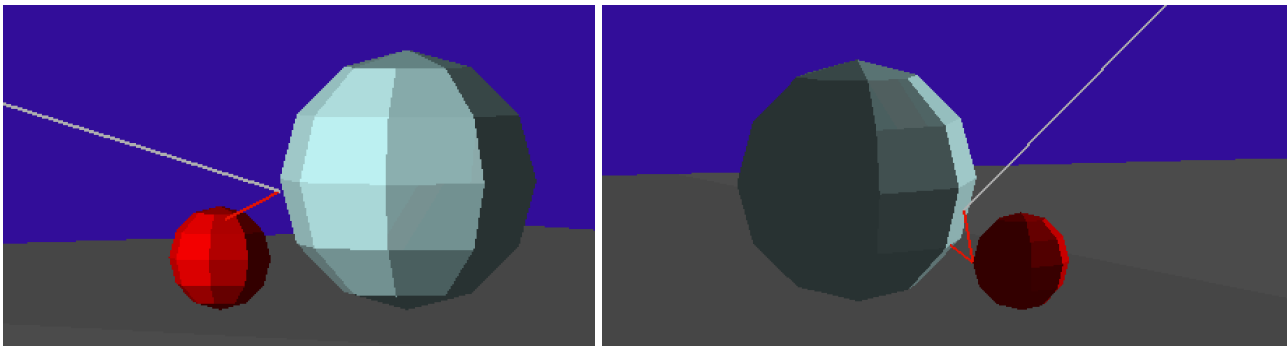


Figura 6: raggio bianco = raggio della luce, raggi rossi = raggi riflessi

I parametri di ingresso utilizzati sono i seguenti:

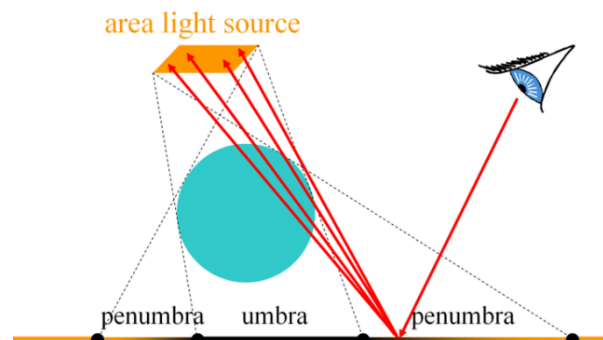
```
✓ -size 200 200
✓ -input "/Users/erika/Desktop/MAGISTRA...ratorio/lab4/lab4/reflective_spheres.obj"
✓ -background_color 0.2 0.1 0.6
✓ -num_shadow_samples 1
✓ -num_bounces 2
```

Il terzo punto dell'esercitazione, infine, richiedeva di implementare un meccanismo per la gestione di soft shadows mediante risorse luminose ad area anziché puntiformi.

Finora le luci in scena erano gestite come puntiformi.

Nel caso in cui la luce non fosse puntiforme, ma avesse un'area, si potrebbero creare più shadow rays, in modo tale da avere un'ombra più accurata e non un passaggio netto fra la zona in ombra e quella non in ombra.

In particolare, per ciascuna area di luce, partono più raggi non più dal centro ma da punti casuali all'interno dell'area stessa. Il contributo luminoso viene calcolato come media dei contributi luminosi di ciascun raggio.



Nell'esercitazione, la scelta di avere un'area di luce o meno veniva indicata attraverso il parametro di input "num_shadow_samples".

Qui di seguito si riporta la logica dell'algoritmo:

```
Vec3f answer1;
for(j = 0; j < num_shadows; j++){
    pointOnLight = f->RandomPoint();
    dirToLight = pointOnLight - point;
    dirToLight.Normalize();
    Ray *shadowRay = new Ray(point, dirToLight);
    Hit *newHit = new Hit();
    bool colpito = CastRay(*shadowRay, *newHit, 0);
    if(colpito){
        Vec3f n_point = shadowRay->pointAtParameter(newHit->getT());
        Vec3f dista;
        dista.Sub(dista, n_point, pointOnLight);
```

```

        if(dista.Length() < 0.01){
            if (normal.Dot3 (dirToLight) > 0){
                Vec3f lightColor = 0.2 *
                    f->getMaterial() ->getEmittedColor() *
                    f->getArea ();
                answer1 +=
                    m->Shade(ray,hit,dirToLight,lightColor, args);
            }
        }
    }
}
answer1 /= num_shadows;
answer += answer1;

```

Eseguendo il programma si ottiene il seguente risultato:

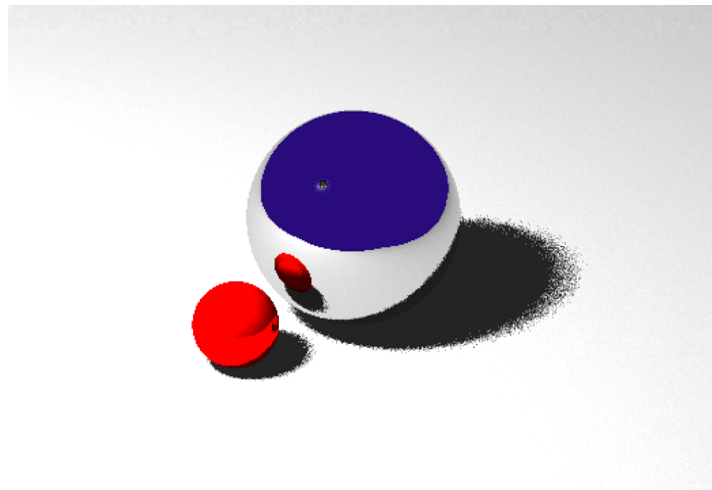
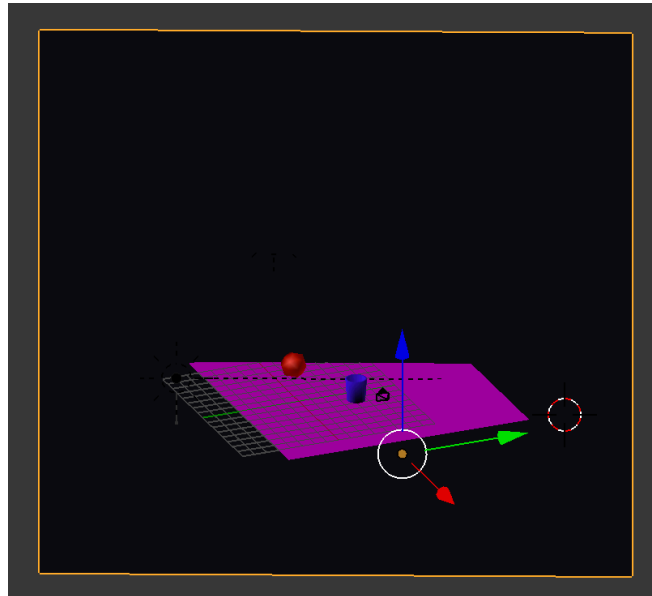


Figura 7: raytracing con soft shadows

Parte 2 – Digital Art: uso di Ray Tracing in Blender

La seconda parte dell'esercitazione richiedeva di utilizzare le tecniche del ray tracing in un ambiente di modellazione/grafica.

Per prima cosa si è scelto di effettuare alcune prime sperimentazioni in una scena semplice, costituita da soli due oggetti e da due piani (uno per il pavimento ed uno per la parete posteriore).



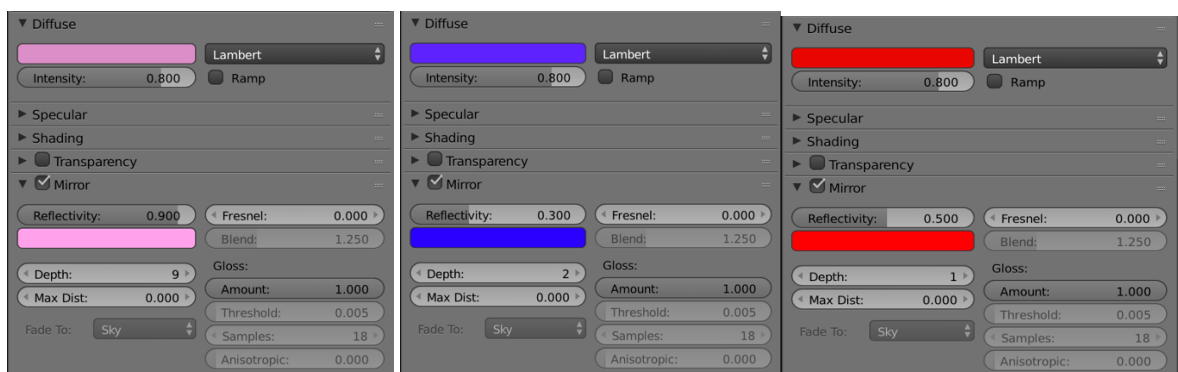
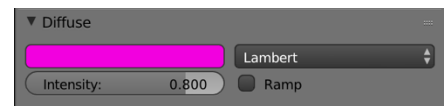
Agli oggetti non è stata applicata nessuna texture, ma sono stati associati solo dei colori.

Per impostare le proprietà del materiale è necessario lavorare sul pannello del materiale.

In particolare, per ciascun materiale è possibile stabilire la componente diffusiva, la componente speculare (o emissiva), la trasparenza e la riflessività.

Qui di seguito si riportano le caratteristiche associate agli oggetti in scena:

- al pavimento è stato semplicemente associato un colore, quindi non riflette gli oggetti in scena ed attraverso esso non passano luci;
- alla parete posteriore alla sfera ed al bicchiere sono stati associati un colore ed un certo valore di riflessività; questo vuol dire che su di essi si riflettono gli altri oggetti in scena. Il parametro "Depth", inoltre, stabilisce il numero di salti che i raggi secondari (che rimbalzano sugli oggetti per via della loro riflessività) devono compiere.



Una volta impostate le proprietà del materiale sono stati effettuati i primi esperimenti sul numero di salti dei raggi secondari.

In particolare:

- modificando il numero di salti della parete posteriore è stato possibile ottenere i seguenti risultati:

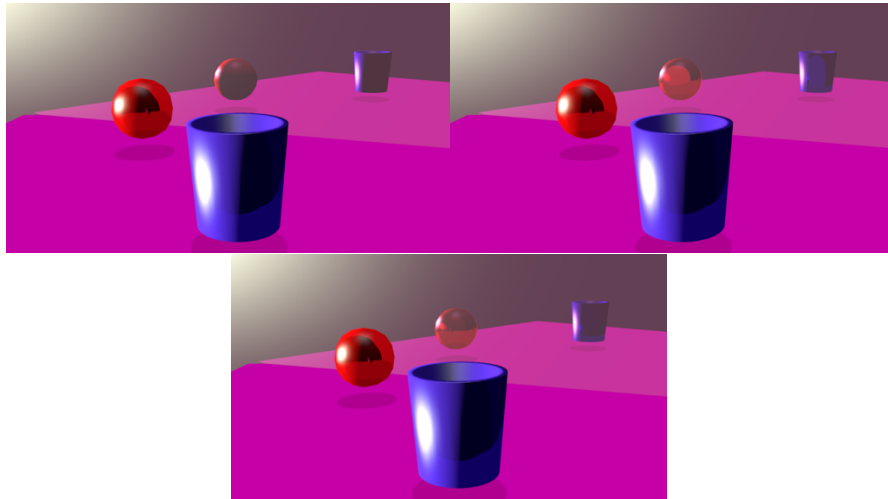


Figura 8: passaggio della parete posteriore da 1 a 3 salti

- modificando il numero di salti della sfera sono stati ottenuti i seguenti risultati:

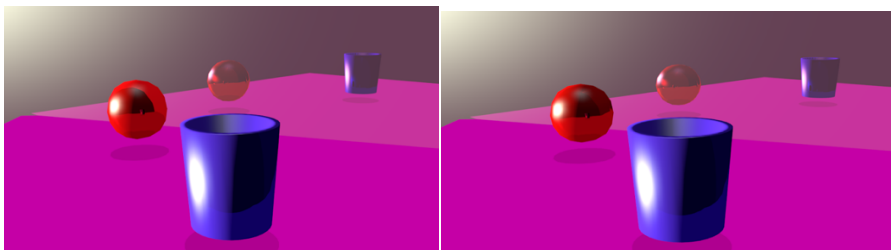
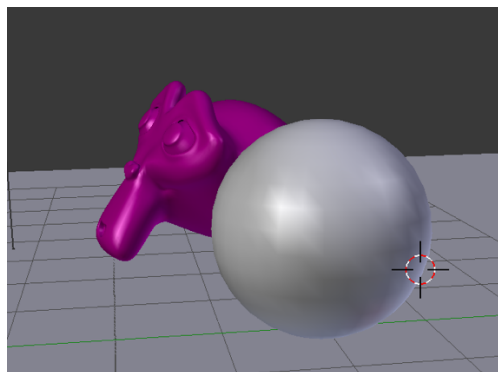
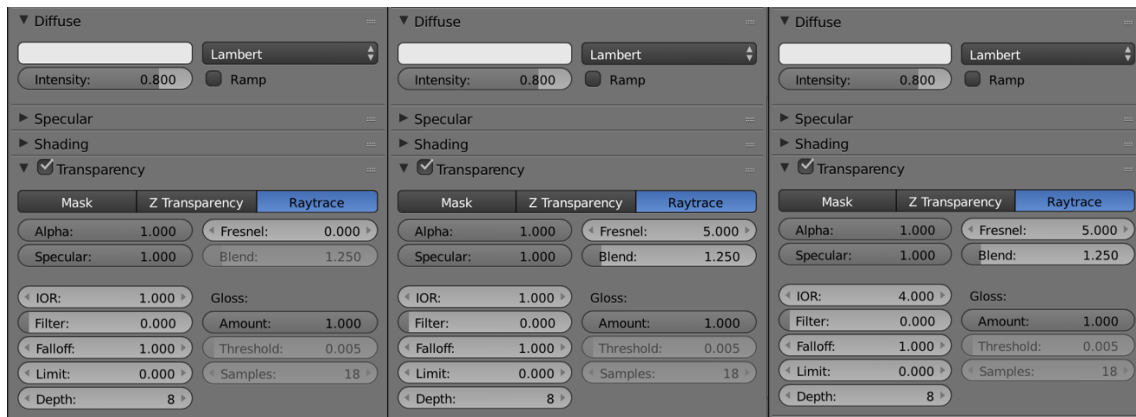


Figura 9: passaggio della sfera da 1 a 2 salti

Sono poi stati effettuati ulteriori esperimenti su un'altra scena molto semplice:



Questa volta però sono state sperimentate le tecniche di trasparenza. Infatti, mentre alla scimmia ed al pavimento sono stati applicati materiali riflettenti, alla sfera sono stati associati rispettivamente i seguenti materiali:



Il parametro “Fresnel” stabilisce il grado di trasparenza. Il parametro “IOR” stabilisce, invece, di quanto deve essere deformato l’oggetto che si trova dietro la sfera.

Effettuando il rendering sono stati ottenuti rispettivamente i seguenti risultati:

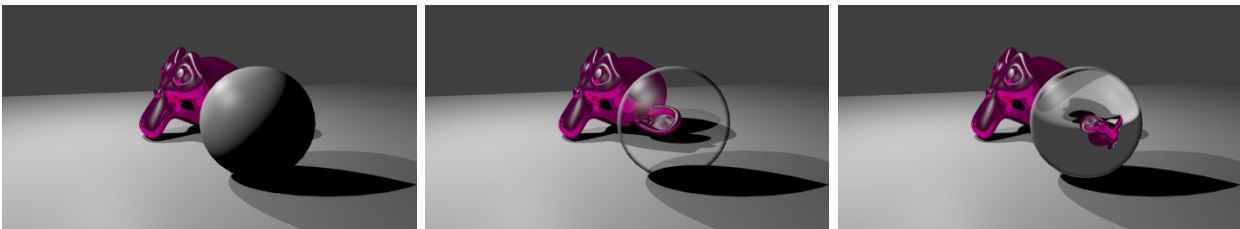


Figura 10: cambiamenti di trasparenza e deformazione

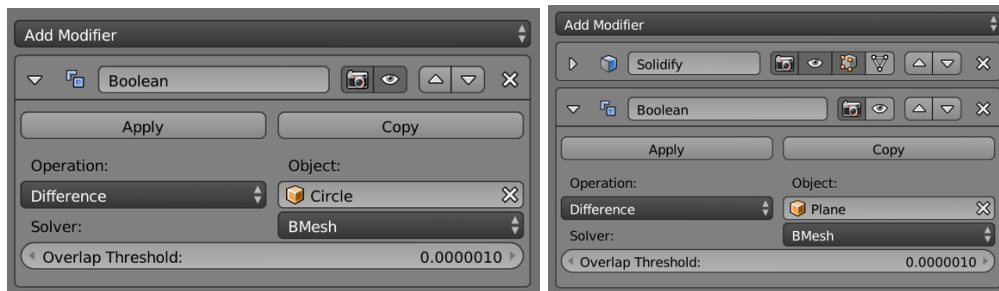
Terminati gli esperimenti con le proprietà dei materiali ed il raytracing in Blender si è deciso di riprendere il modello di Olaf dell’esercitazione precedente e si applicarvi degli effetti attraverso il materiale.

Il risultato finale è stato il seguente:

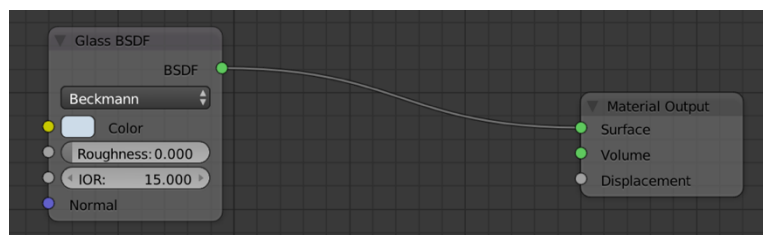


Figura 11: Olaf con Ray Tracing

La pozzanghera è stata realizzata deformando un cerchio ed applicando il modificatore differenza fra essa ed il piano sottostante:

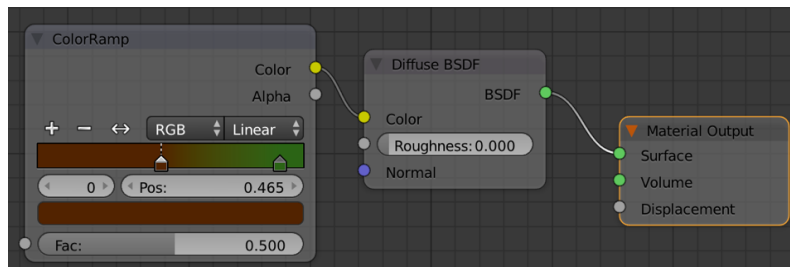


Per realizzare l'effetto "specchio" della pozzanghera è stato utilizzato il seguente materiale:



Non è stato possibile impostare l'effetto mirror come negli esempi precedenti in quanto la modalità utilizzata in questo caso è Cycles Render (e non Blender Render), necessaria per la realizzazione dell'erba.

Per dare sfumatura al terreno, invece, sono stati utilizzati i seguenti nodi:



Sono stati utilizzati due piani per realizzare il terreno, in modo tale da poter lasciare il contorno senza erba intorno alla pozzanghera.

Sul piano superiore è stata applicata la tecnica del Particle System per realizzare i fili d'erba.

Infine, per il cielo è stata utilizzata una texture inserita come proprietà del World:

