Rendering & ray tracing

Implementazione e applicazione pratica

Che cos'e CG(computer graphics)?

- Quando si parla di CG si intendono immagini, o video creati al computer.
- CG e' una branca dell'informatica che comprende molti rami, ed è impossibile trattarli tutti in uns sola lezione.
- In particolare tratteremo **grafica 3D**.

3D Graphics

- La **grafica 3D** è un ramo della CG che basa la creazione di immagini statiche o in movimento, sull'elaborazione di modelli tridimensionali da parte di un computer.
- Essa viene utilizzata nella creazione di opere o parti di opere per il cinema o la televisione, nei videogiochi, in architettura, in ingegneria e in svariati ambiti scientifici.

3D Graphics - Rendering

Il metodo di produzione della grafica 3D è composto da due elementi:

- Una descrizione di ciò che si intende visualizzare (scena), composta di rappresentazioni matematiche di oggetti tridimensionali, detti "modelli".
- Un motore di render che si fa carico di tutti i calcoli necessari per la creazione dell’immagine 3D partendo da quella descritta nella scena.

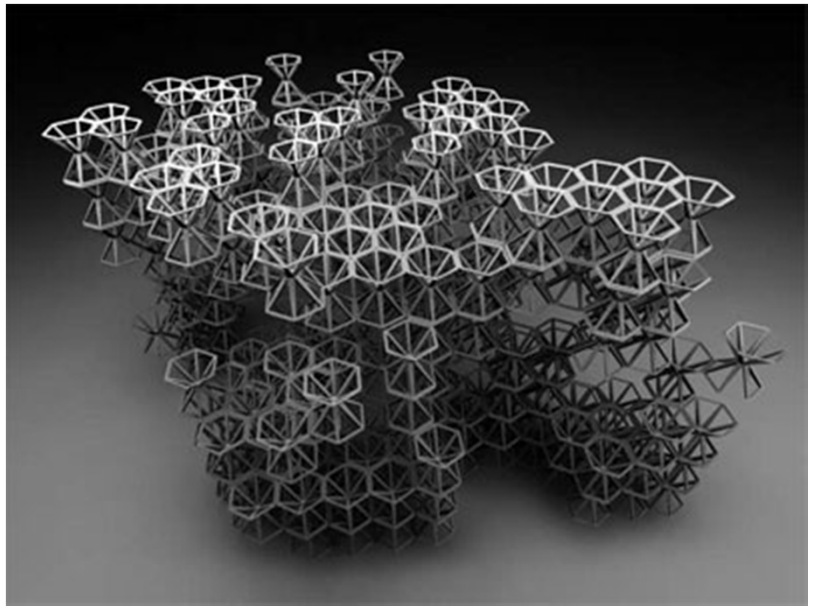
Scena

- Oggetti tridimensionali semplici possono essere rappresentati con equazioni operanti su un sistema di riferimento cartesiano tridimensionale: per esempio, l'equazione $x^2+y^2+z^2=r^2$ descrive una sfera di raggio r.

```
4  aamx 1
5  maxt 6000
6  sphr 900 400 -300 120 6
7  sphr 900 600 -200 100 2
8  sphr 800 500 100 250 4
9  poly 1000 100 0 1400 600 -500 1000 1100 0 5
10 poly 600 100 0 600 1100 0 200 600 -500 5
11 lght 400 600 -400
12 lght 800 600 -1000
13 matr 1 0.1 0.1 0.1 0.5 0 0.9 1.3
14 matr 2 1 0 1 0.5 0.5 0 1
15 matr 3 1 1 1 1 0.1 0 1
```

Scena

- Oggetti piu’ complessi si ottengono dalla unione di quelli semplici(detti anche primitivi).

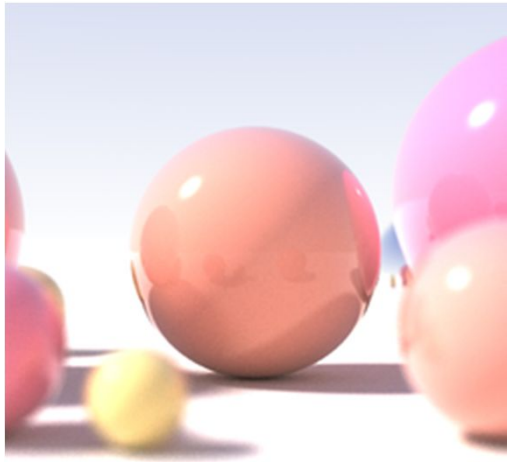


Motore grafico

- Il rendering e' il processo di trasformazione di una scena nell'immagine finale.
- Il motore grafico prende come input la scena, ed esegue una elaborazione, secondo un algoritmo, per ottenere l'immagine finale.
- Vi sono numerosi algoritmi di rendering, ciauscuno con vantaggi e svantaggi.
- Visto elevato numero di algoritmi, oggi ne prenderemo in esamo soltanto uno: il **ray tracing**.

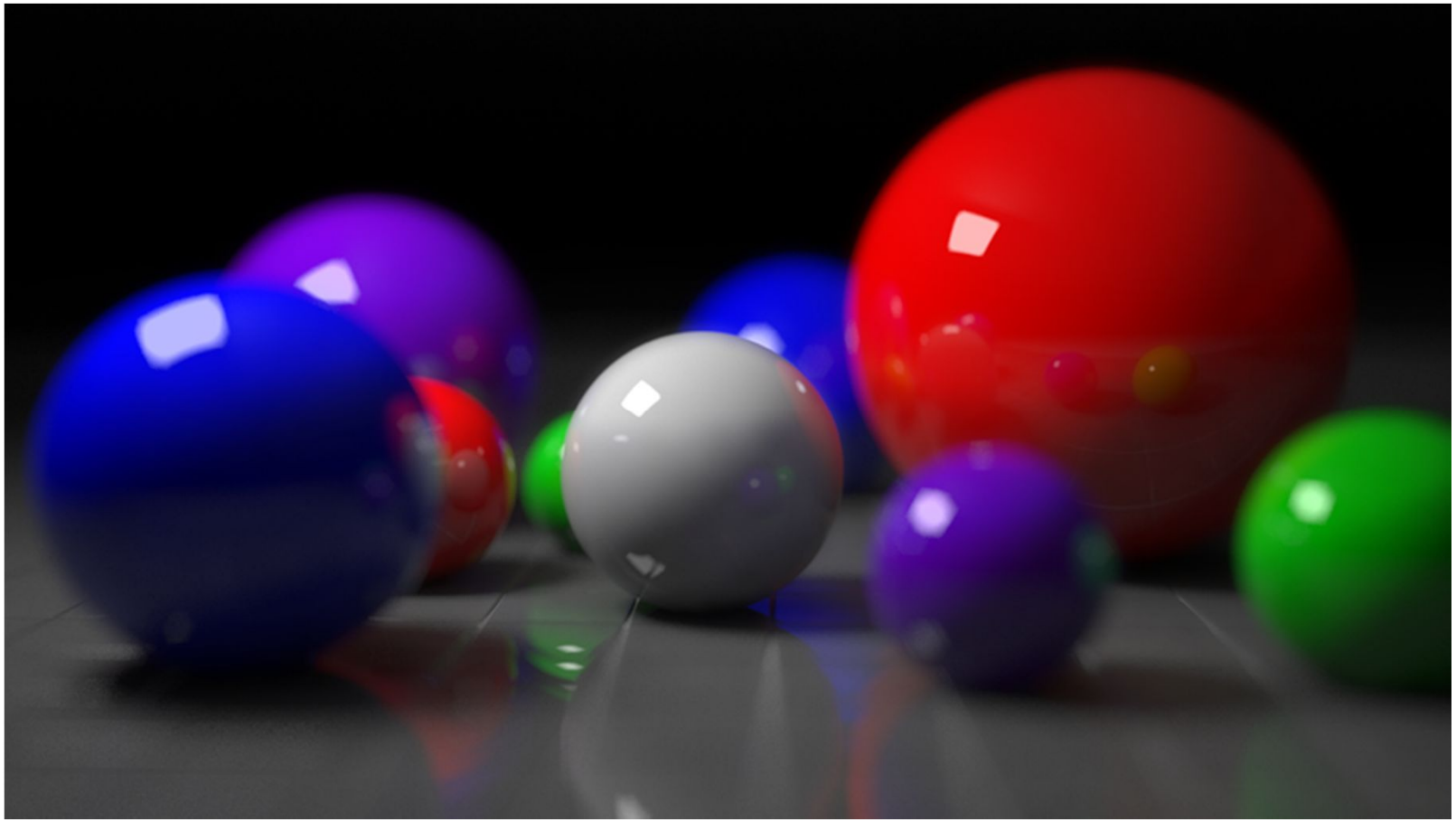
Che cos’e il raytracing?

- Il Ray tracing è una tecnica generale di geometria ottica basata sul calcolo del percorso fatto dalla luce, seguendone i raggi attraverso l'interazione con le superfici.



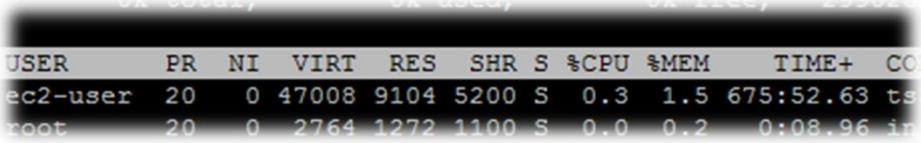
Vantaggi del raytracing

- Livelli di realismo molto elevati, sopratutto per quanto riguarda l'illuminazione.
- La corrispondenza al modello fisico di propagazione di luce, presente nella realta'.
- Relativa semplicità d'implementazione.
- L'indipendenza dei pixel garantisce un parallelismo quasi perfetto durante i calcoli.



Svantaggi del raytracing

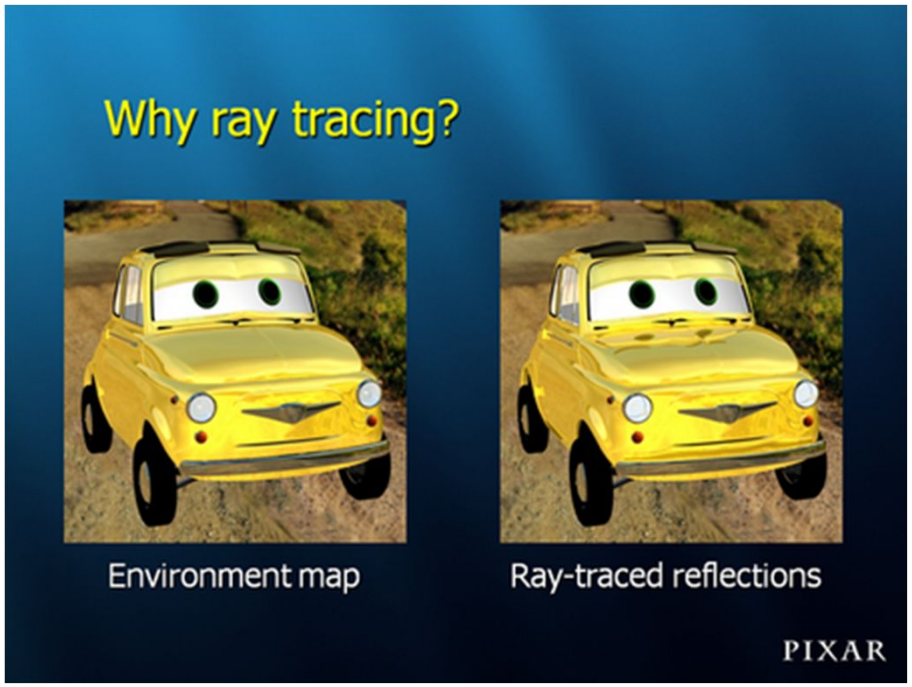
- La velocita' di rendering è inferiore ad altri algoritmi (e.g. scanline), che condividono le informazioni tra i pixel, aumentando notevolmente la velocita'.
- Aggiungendo opzioni come l'AntiAliasing si può diminuire la velocita' di esecuzione.



Applicazioni pratiche

- Può essere utilizzato nel caso in cui si voglia ottenere un livello di realismo maggiore.
- Può essere combinato con altri algoritmi per costruire ombre, riflessioni e rifrazioni.

Cars

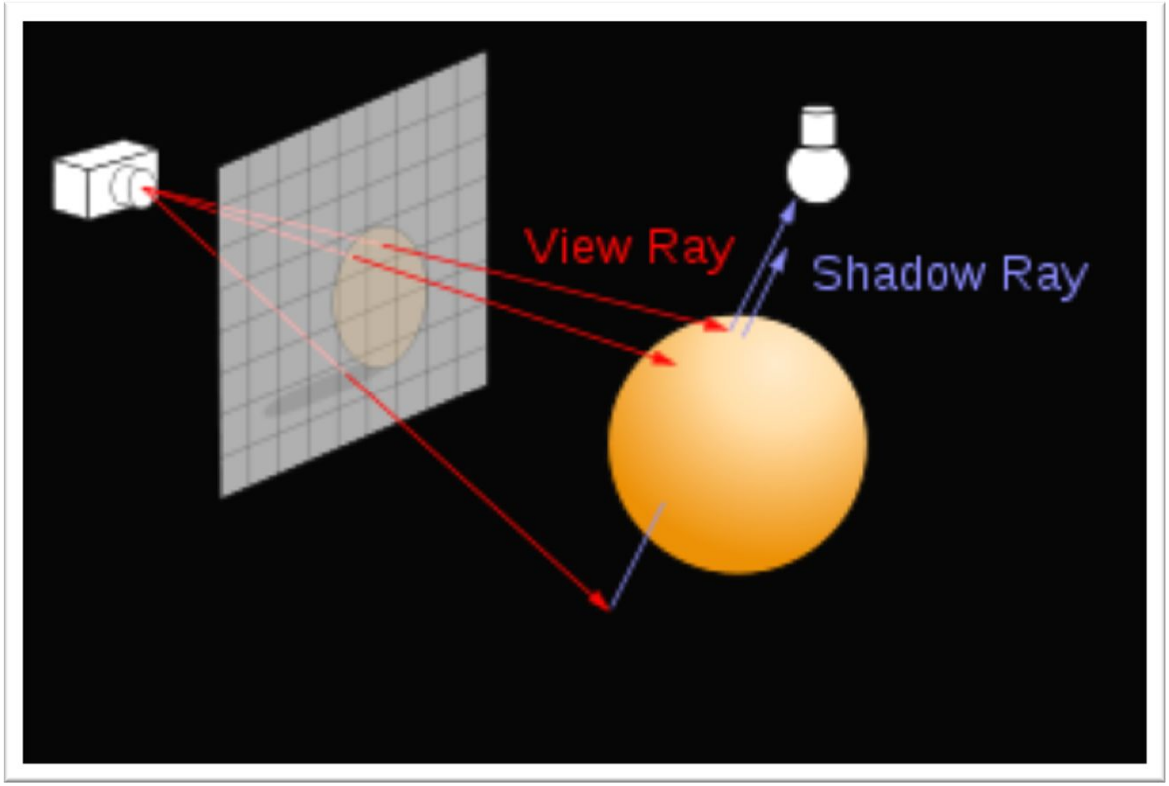

Why ray tracing?
Environment map
Ray-traced reflections
PIXAR

Cosa accade in natura

- In natura, una sorgente di luce emette un raggio di luce che viaggia fino a raggiungere una superficie che ne interrompe il tragitto.
- La superficie può riflettere tutto o parte il raggio in una o più direzioni,può però anche assorbire una parte del raggio, causando una perdita di intensità della luce riflessa e/o rifratta.
- Partendo da qui, i raggi riflessi e/o rifratti possono colpire altre superfici, dove verranno assorbiti, riflessi e/o rifratti nuovamente. Alcuni di questi raggi, colpiscono il nostro occhio, permettendoci di vedere la scena e contribuendo al disegno dell'immagine finale.

Algoritmo di raytracing

- Il Raytracing fa la stessa cosa, ma al contrario.
- Il raggio viene mandato dalla camera (occhio) attraverso un pixel dello schermo virtuale.



Algoritmo di raytracing

- Si trova l'intersezione piu' vicina.
- Dal punto d'intersezione vengono mandati raggi aggiuntivi per determinare se l'oggetto e':
  - In ombra
  - Componente riflessa della luce
  - Componente rifratta della luce
- Colore risultante e' dato dalla somma dei contibuti di tutti i raggi.

Project 0 - Architettura

bal Scope)

```
/*
 *
 *    ______             _              ____
 *   (_____ \           (_)          _ (____)
 *    _____) )___ ___   _ ____  ____| |_ _  __ _
 *   |  ____// ___) _ \ | |/ _  )/ ___)  _) |/ /| |
 *   | |    | |  | |_| || ( (/ /( (___| |_|   /_| |
 *   |_|    |_|   \___/_| |\____)\____)\___)\_____/
 *                    (__/
 *
 *   Project0 - simple C/C++ raytracer
 *   created by Danil Tumaykin, Alessandro Carlin, Marco Cauzzi, Marco Begnozzi
 *
 *   P0 was intended as learning project, so feel free to modify and reuse our code :P
 *
 *   want to contibute - email me - d.tumaykin@gmail.com
 *
 */


#include <iostream>
#include <math.h>
#include <time.h>

#include "structures.h"
#include "util.h"
```

Strutture dati

- Vettore, punto, colore – vengono rappresentati da 3 double
- Raggio – punto di origine + vettore di direzione
- Sfera – centro(punto), raggio
- Triangolo – 3 vertici(punto)
- Materiale – struttura contenente colore, coefficienti di diffusione, riflessione e refrazione
- Scena – struttura parametri di scena(eg. risoluzione), tutti primitivi e materiali

Esecuzione

- Essenzialmente tracer deve fare 3 cose:
  - Leggere la scena da un file di configurazione
  - Eseguire il tracing
  - Scrivere il risultato in nel file di output

Lettura di configurazione

int loadConfig(char * path, scene_t &scene);

- Una funzione che prende dentro il file di configurazione, esegue il parsing e ritorna la scena gia' completa, nel caso di errore di lettura, ritorna il numero della riga.

```
53  int loadConfig(char * path, scene_t &scene)
54  {
55      std::ifstream ifs(path);
56      int line = 0;
57
58      if(ifs.bad()) return 0;
59
60      std::vector<prim_t> tempPrim;
61      std::vector<light_t> tempLight;
62      std::vector<material_t> tempMat;
63
64      while(!ifs.eof())
65      {
66          line++;
67          switch(getOpcode(ifs))
68          {
69          case RESO:
70              ifs >> scene.screenResX;
71              ifs >> scene.screenResY;
72              break;
73          case SIZE:
74              ifs >> scene.screenSizeX;
```

Formato di file di configurazione

Una stringa di configurazione e' composta da un opcode di 4 caratteri e i suoi parametri. Alcuni opcode disponibili:

- reso <resX> <resY>
- size <sizeX> <sizeY>
- sphr <centerX> <centerY> <centerZ> <radius> <matId>
- poly <aX> <aY> <aZ> <bX> <bY> <bZ> <cX> <cY> <cZ> <matId>
- lght <posX> <posY> <posZ>
- matr <Id> <colR> <colG> <colB> <cfDiff> <cfRefl> <cfQRefr> <cfRefr>

```
aamx 1
maxt 6000
sphr 900 400 -300 120 6
sphr 900 600 -200 100 2
sphr 800 500 100 250 4
poly 1000 100 0 1400 600 -500 1000 1100 0
poly 600 100 0 600 1100 0 200 600 -500 5
lght 400 600 -400
lght 800 600 -1000
matr 1 0.1 0.1 0.1 0.5 0 0.9 1.3
matr 2 1 0 1 0.5 0.5 0 1
matr 3 1 1 1 1 0.1 0 1
```

“Tracing”

Il processo di effettivo tracciamento dei raggi si puo suddividere in vari fasi:

- Determinazione di raggio
- Tracciamento di raggio primario
- Tracciamento di raggi secondari(ricorsione)
- Somma dei risultati

Determinazione del raggio

Nel main() con 2 cicli passiamo per tutti pixel dell'immagine futura:

```
53      for(int i = 0; i < scene.screenResX; i++)
54      {
55          for(int j = 0; j < scene.screenResY; j++)
56          {
57              color_t r = getColor(i, j, scene);
```

A seguito, nel getColor(), vengono determinate le coordinate "reali" del pixel, e viene creato il raggio:

```
319     for(double i = pixelSizeX * x; i < pixelSizeX * (x + 1); i += AAShiftX)
320         for(double j = pixelSizeY * y; j < pixelSizeY * (y + 1); j += AAShi
321         {
322             point_t pixelPos = { i, j, 0.0f};
323             vector_t dst = pixelPos - scene.cam;
324             ray_t ray = { scene.cam, dst};
325             //ray_t ray = {{ i, j, 0}, {i, j, 0.0f}};
326             norm(ray.dst);
327
328             cAcc += trace(ray, scene, 0, 1.0f);
329         }
```

trace()

Tutto il progetto gira attorno alla funzione trace().

- Inizialmente testa l'intersezione con tutti primitivi nella scena(algoritmo varia a secondo del tipo di primitivo).
- Se non c'e nessuna intersezione viene ritornato il colore di background.
- Altrimenti dal punto d'intersezione vengono generati altri raggi(raggi secondari).

```
//get nearest intersection
for(int i = 0; i < scene.primCount; i++)
    if(getIntersection(scene.prim[i], ray, t))
        p = &scene.prim[i];

if(!p) return background; // no intersections
```

Raggi secondari

- Verso ogni fonte di luce, per determinare se il punto d'intersezione e' in ombra o meno.
- Viene calcolato il raggio riflesso, e viene ripassato alla funzione trace(). Ricorsione.
- Viene calcolato il raggio refratto, sempre ripassato a trace().

Il colore risultante del pixel e' dato dalla somma dei contributi dei raggi.

```
1494 |
1495         refrRay.dst = ray.dst * n + normal * refr;
1496         c += trace(refrRay, scene, depth + 1, currRefr) * m->coefRefract;
1497     }
```

Esempio di un algoritmo di intesezione

- Una sfera e' data dall'equazione: (Pi-Pc)^2 = r^2
- Un raggio e' data dall'equazione: Pi = Po + t * vD

Due equazioni vengono messi in sistema, che dopo viene ridotta a un'equazione di secondo grado:

$$t^2 * vD^2 + 2 * t * vD * (Po - Pc) + (Po - Pc)^2 - r^2 = 0$$

Questa "semplice" equazione si risolve in t :P

Spaventooo!!!!1111(non scrivete mai cosi!)

```
/*
//kill me please
if (fabs(n.x) >= fabs(n.y) && fabs(n.x) >= fabs(n.z))
{
    if ((n.x*((intrPoint.z-p.polygon.ptA.z)*(p.polygon.ptB.y-p.polygon.ptA.y)-(intrPoint.y-p.polygon.ptA.y)*(p.
        && (n.x*((intrPoint.z-p.polygon.ptB.z)*(p.polygon.ptC.y-p.polygon.ptB.y)-(intrPoint.y-p.polygon.ptB.y)*
        && (n.x*((intrPoint.z-p.polygon.ptC.z)*(p.polygon.ptA.y-p.polygon.ptC.y)-(intrPoint.y-p.polygon.ptC.y)*
        t = plT;
    if ((n.x*((intrPoint.z-p.polygon.ptA.z)*(p.polygon.ptB.y-p.polygon.ptA.y)-(intrPoint.y-p.polygon.ptA.y)*(p.
        && (n.x*((intrPoint.z-p.polygon.ptB.z)*(p.polygon.ptC.y-p.polygon.ptB.y)-(intrPoint.y-p.polygon.ptB.y)*
        && (n.x*((intrPoint.z-p.polygon.ptC.z)*(p.polygon.ptA.y-p.polygon.ptC.y)-(intrPoint.y-p.polygon.ptC.y)*
        t = plT;
}
else if (fabs(n.y) >= fabs(n.x) && fabs(n.y) >= fabs(n.z))
{
    if ((n.y*((intrPoint.z-p.polygon.ptA.z)*(p.polygon.ptB.x-p.polygon.ptA.x)-(intrPoint.x-p.polygon.ptA.x)*(p.
        && (n.y*((intrPoint.z-p.polygon.ptB.z)*(p.polygon.ptC.x-p.polygon.ptB.x)-(intrPoint.x-p.polygon.ptB.x)*
        && (n.y*((intrPoint.z-p.polygon.ptC.z)*(p.polygon.ptA.x-p.polygon.ptC.x)-(intrPoint.x-p.polygon.ptC.x)*
        t = plT;
    if ((n.y*((intrPoint.z-p.polygon.ptA.z)*(p.polygon.ptB.x-p.polygon.ptA.x)-(intrPoint.x-p.polygon.ptA.x)*(p.
        && (n.y*((intrPoint.z-p.polygon.ptB.z)*(p.polygon.ptC.x-p.polygon.ptB.x)-(intrPoint.x-p.polygon.ptB.x)*
        && (n.y*((intrPoint.z-p.polygon.ptC.z)*(p.polygon.ptA.x-p.polygon.ptC.x)-(intrPoint.x-p.polygon.ptC.x)*
        t = plT;
}
else
{
    if ((n.z*((intrPoint.y-p.polygon.ptA.y)*(p.polygon.ptB.x-p.polygon.ptA.x)-(intrPoint.x-p.polygon.ptA.x)*(p.
        && (n.z*((intrPoint.y-p.polygon.ptB.y)*(p.polygon.ptC.x-p.polygon.ptB.x)-(intrPoint.x-p.polygon.ptB.x)*
        && (n.z*((intrPoint.y-p.polygon.ptC.y)*(p.polygon.ptA.x-p.polygon.ptC.x)-(intrPoint.x-p.polygon.ptC.x)*
        t = plT;
    if ((n.z*((intrPoint.y-p.polygon.ptA.y)*(p.polygon.ptB.x-p.polygon.ptA.x)-(intrPoint.x-p.polygon.ptA.x)*(p.
        && (n.z*((intrPoint.y-p.polygon.ptB.y)*(p.polygon.ptC.x-p.polygon.ptB.x)-(intrPoint.x-p.polygon.ptB.x)*
```

Output

- Come output abbiamo scelto formato BMP, come uno dei piu' semplici.
- Per la creazione del file BMP e' stata usata la libreria easyBMP.

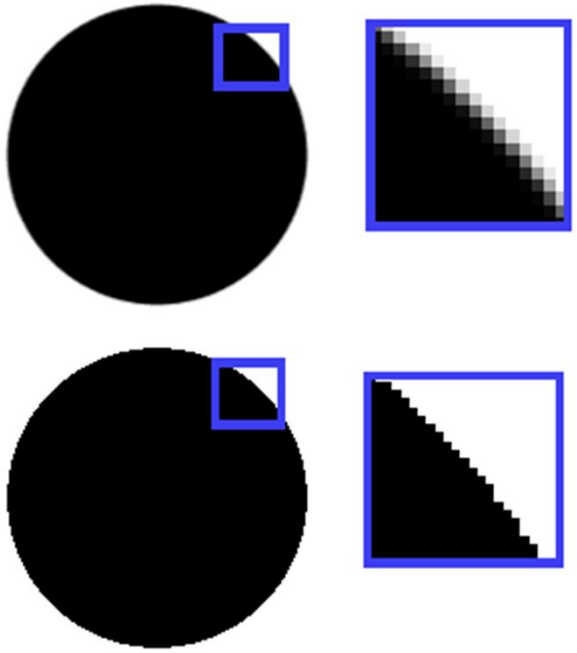
```
48
49      BMP output;
50      output.SetSize(scene.screenResX, scene.screenResY);
51
```

```
73              //write pixels to buffer
74              output(i, j)->Red   = ebmpBYTE(r.r);
75              output(i, j)->Green = ebmpBYTE(r.g);
76              output(i, j)->Blue  = ebmpBYTE(r.b);
```

- http://easybmp.sourceforge.net/

AntiAliasing

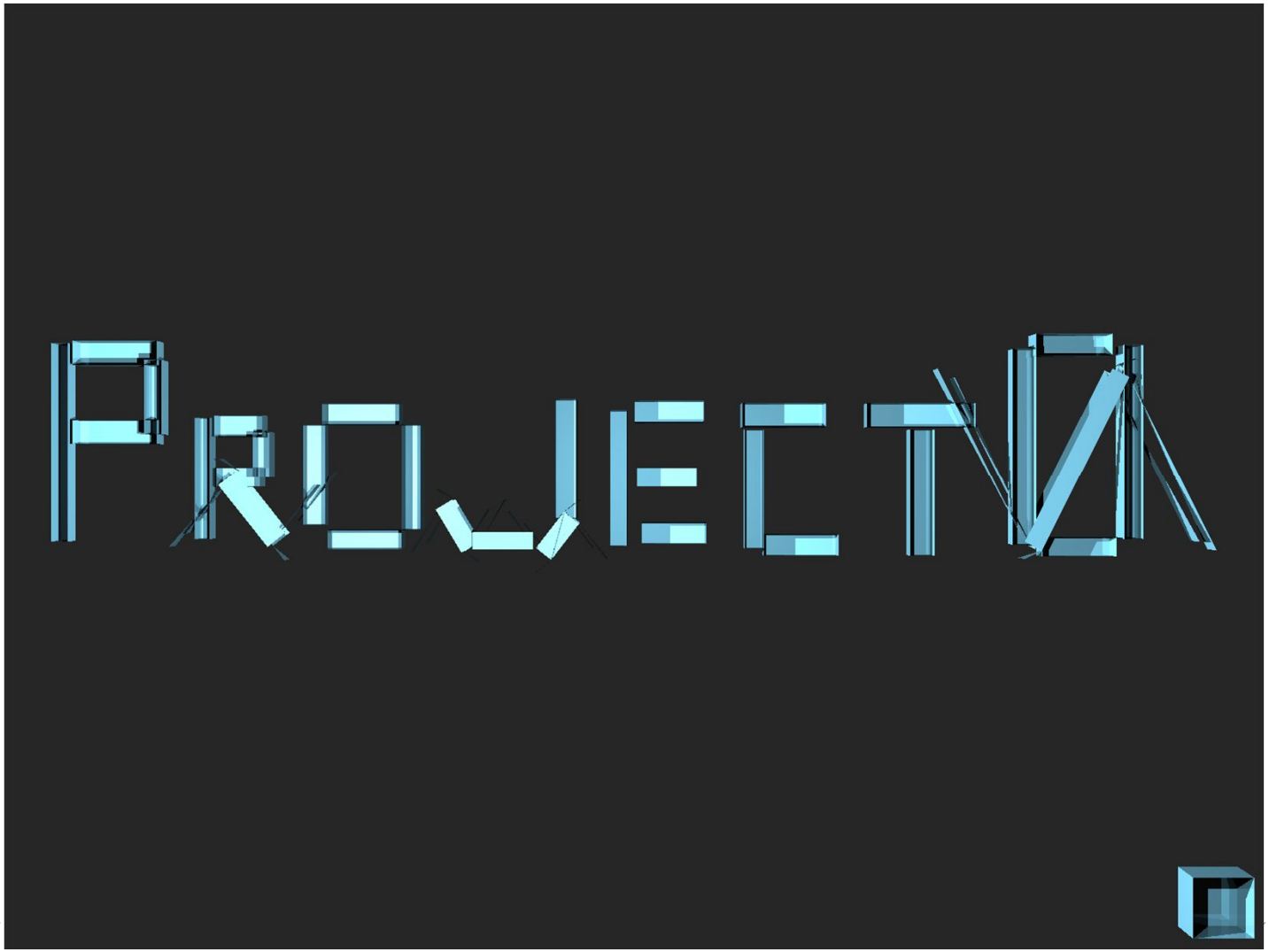
- AA e’ una tecnica che permette di ridurre effetto di aliasing(che caso).



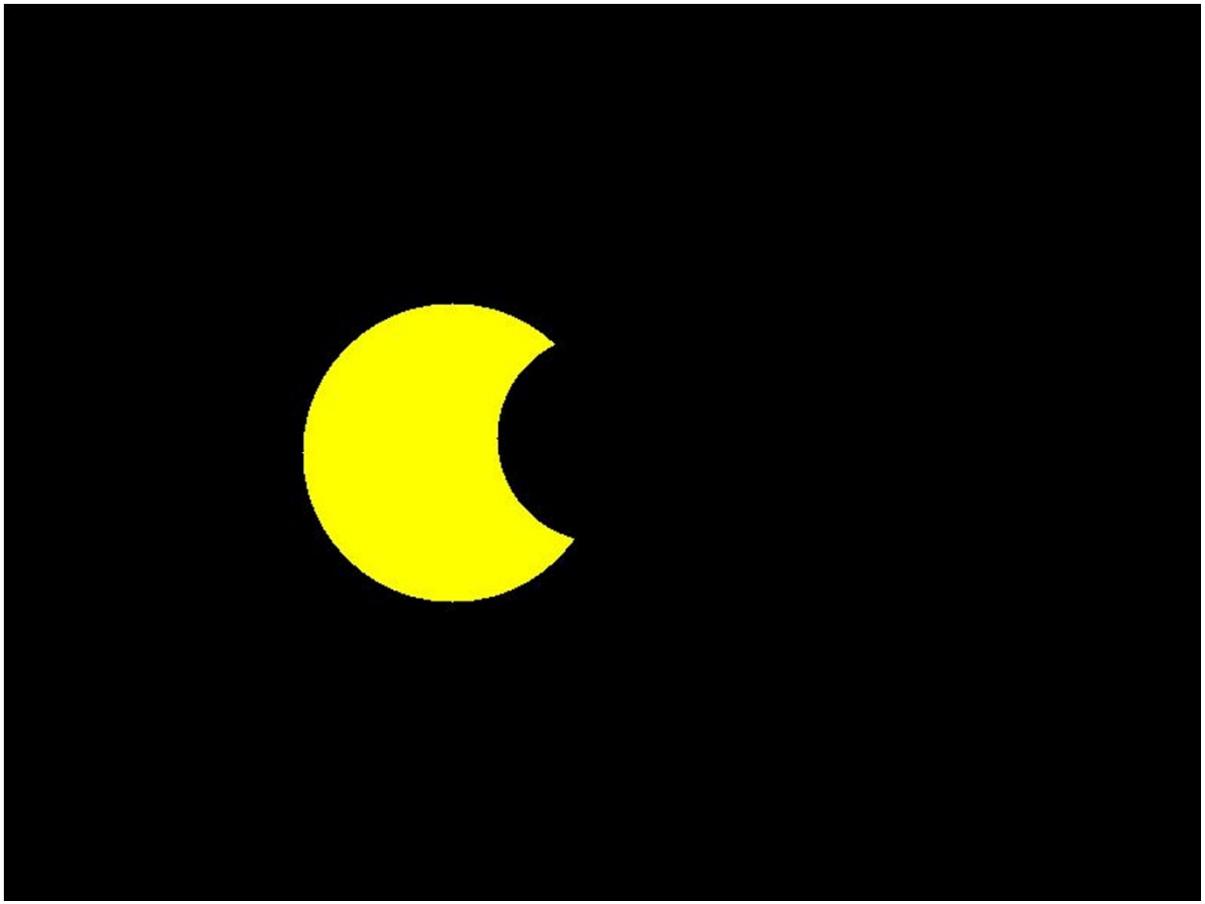
Strumentazione utilizzata

- gedit
- gcc(g++)
- Notepad++
- AutoDesk 3Ds Max
- VS 2010
- GIMP
- git(via github)

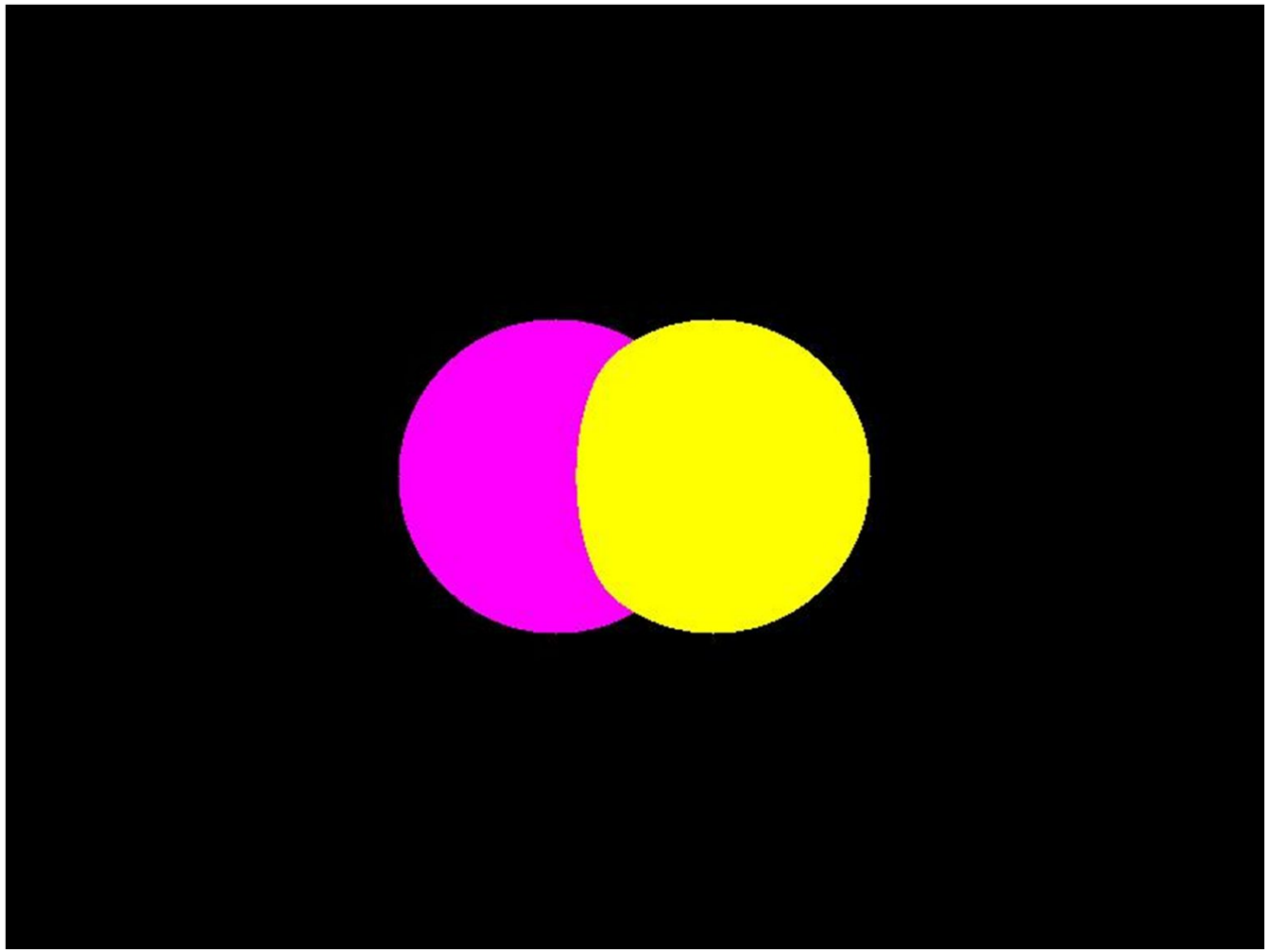
Project0 - Timeline



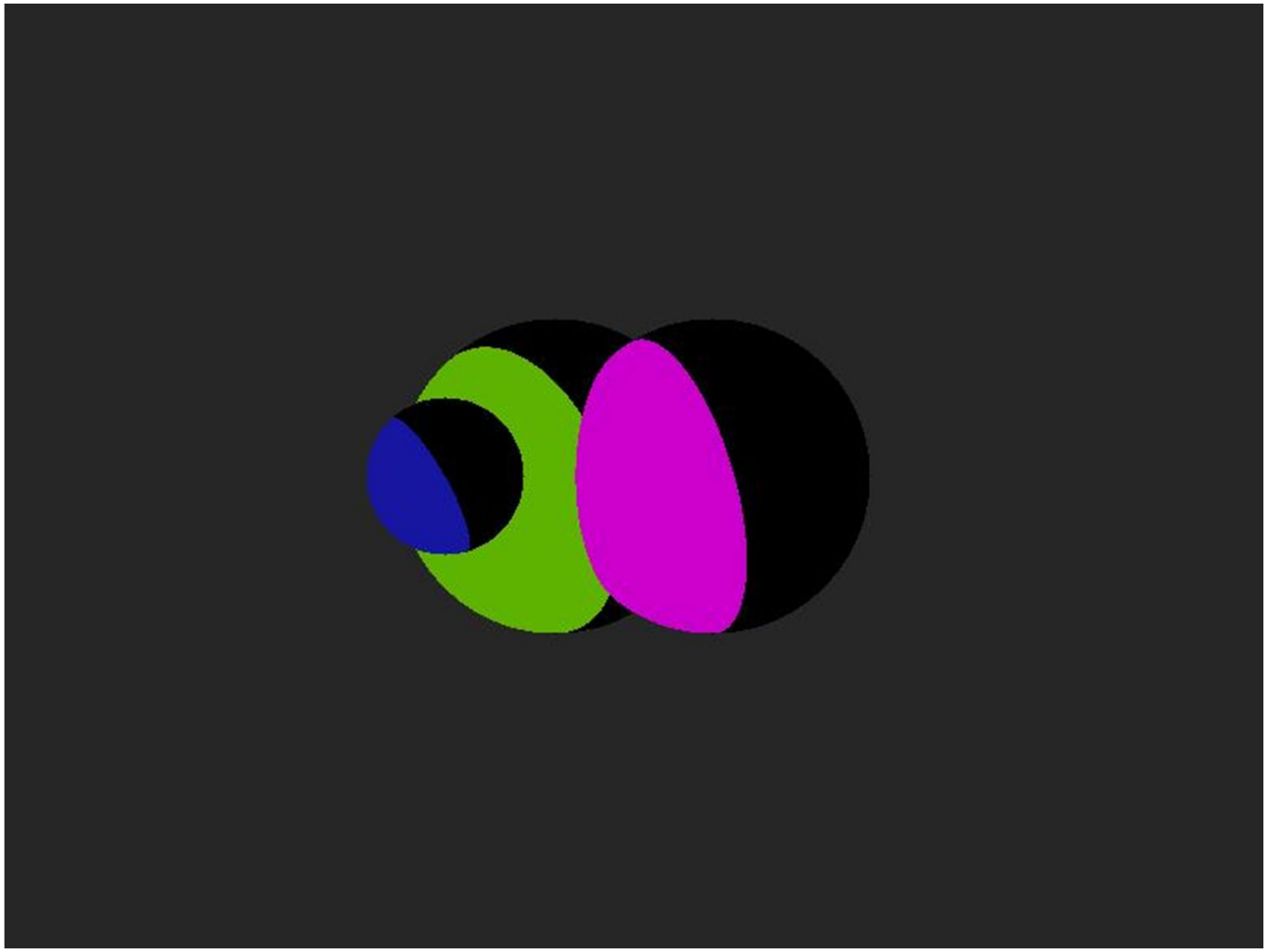
Primo risultato



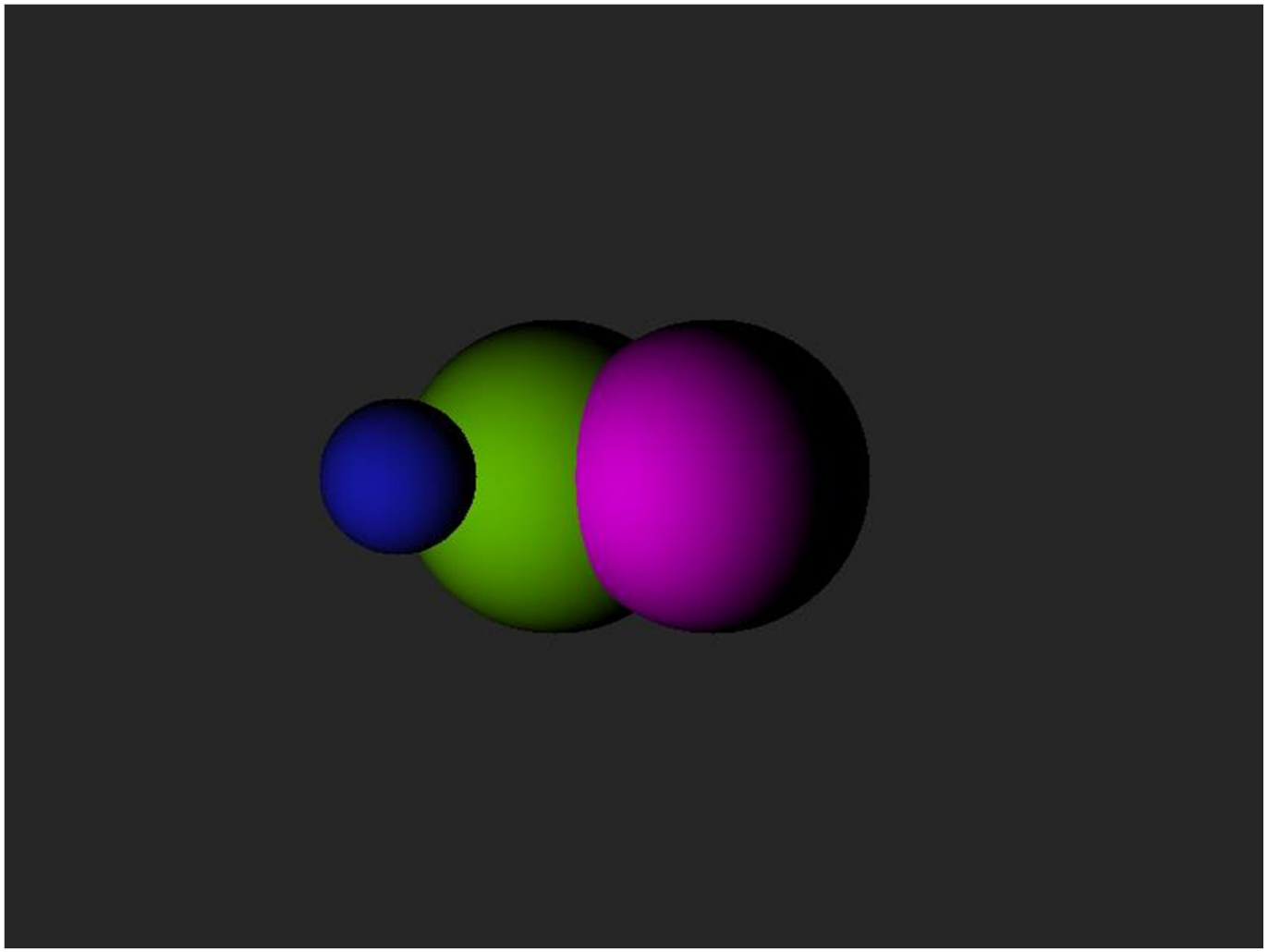
Intersezione tra 2 sfere



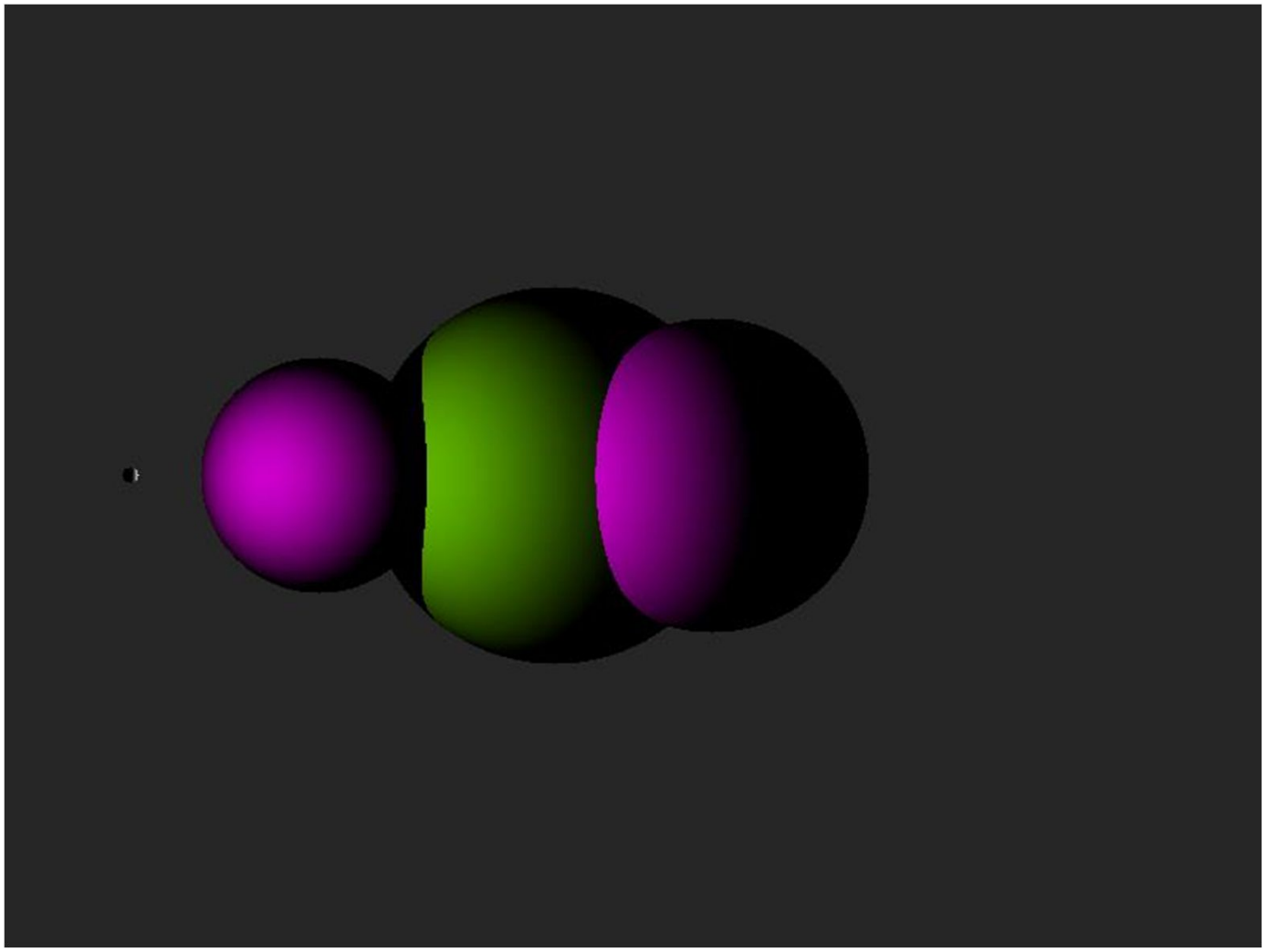
Illuminazione “hard”



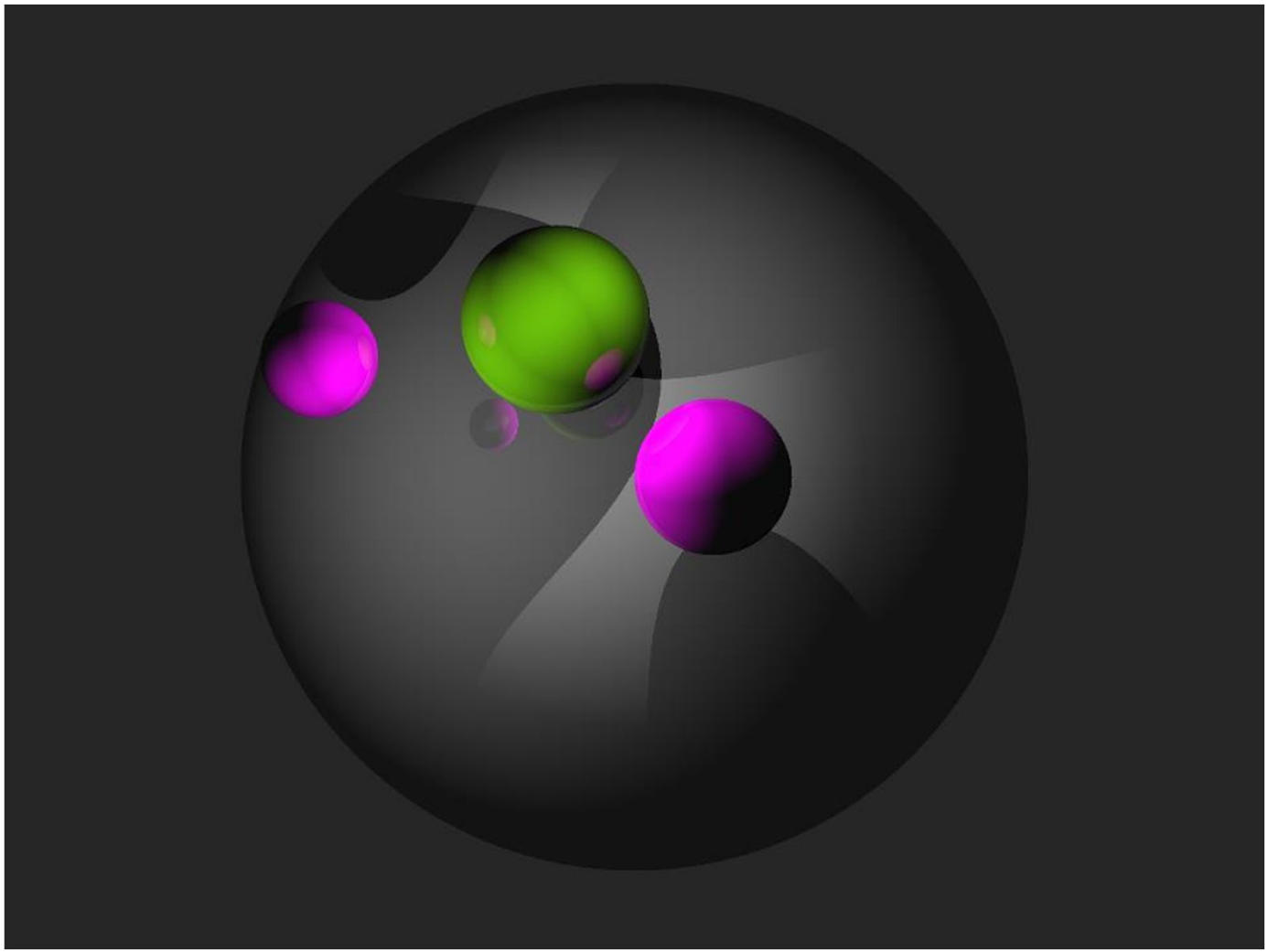
Illuminazione “soft”



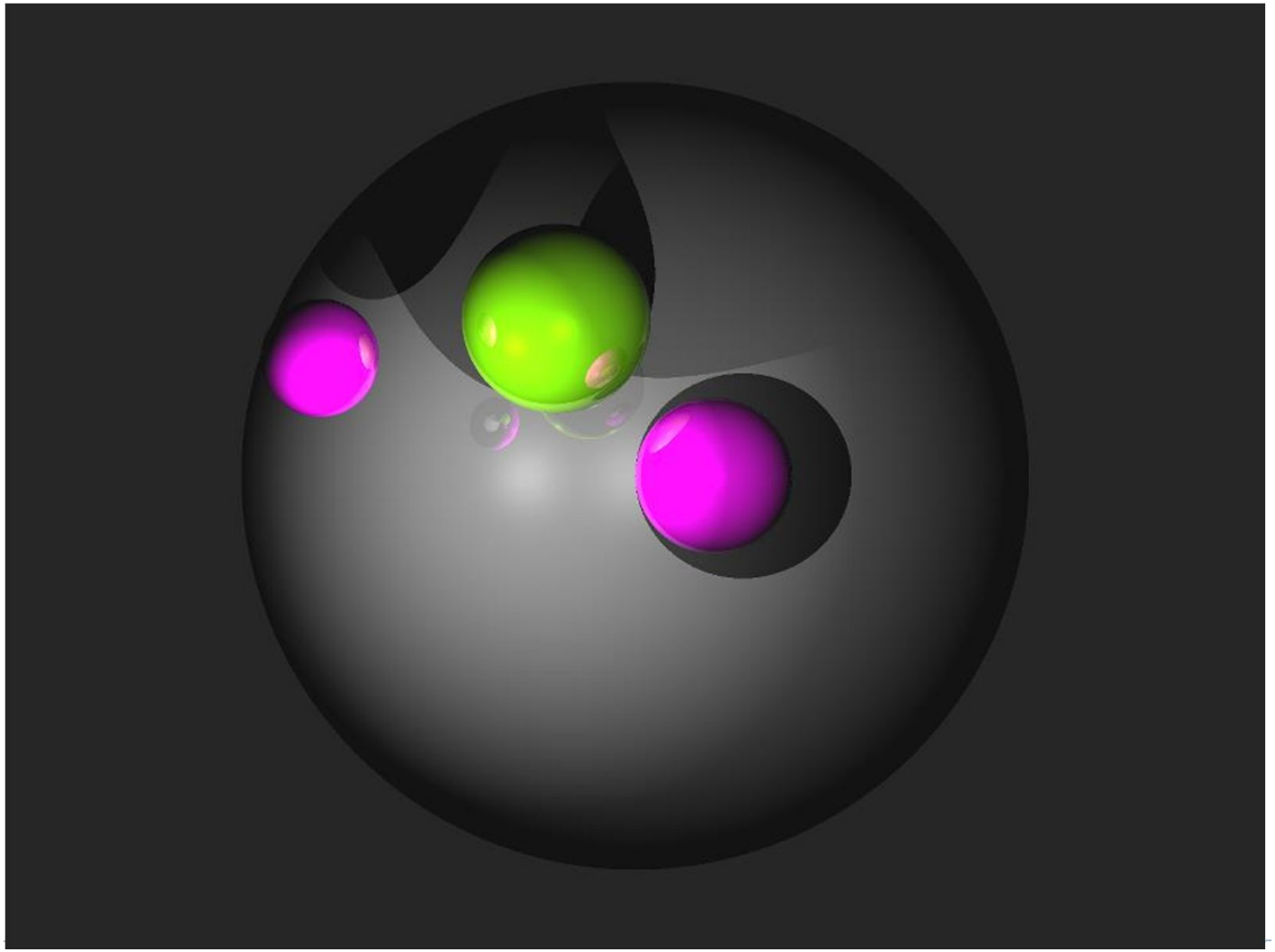
Ombre “soft”



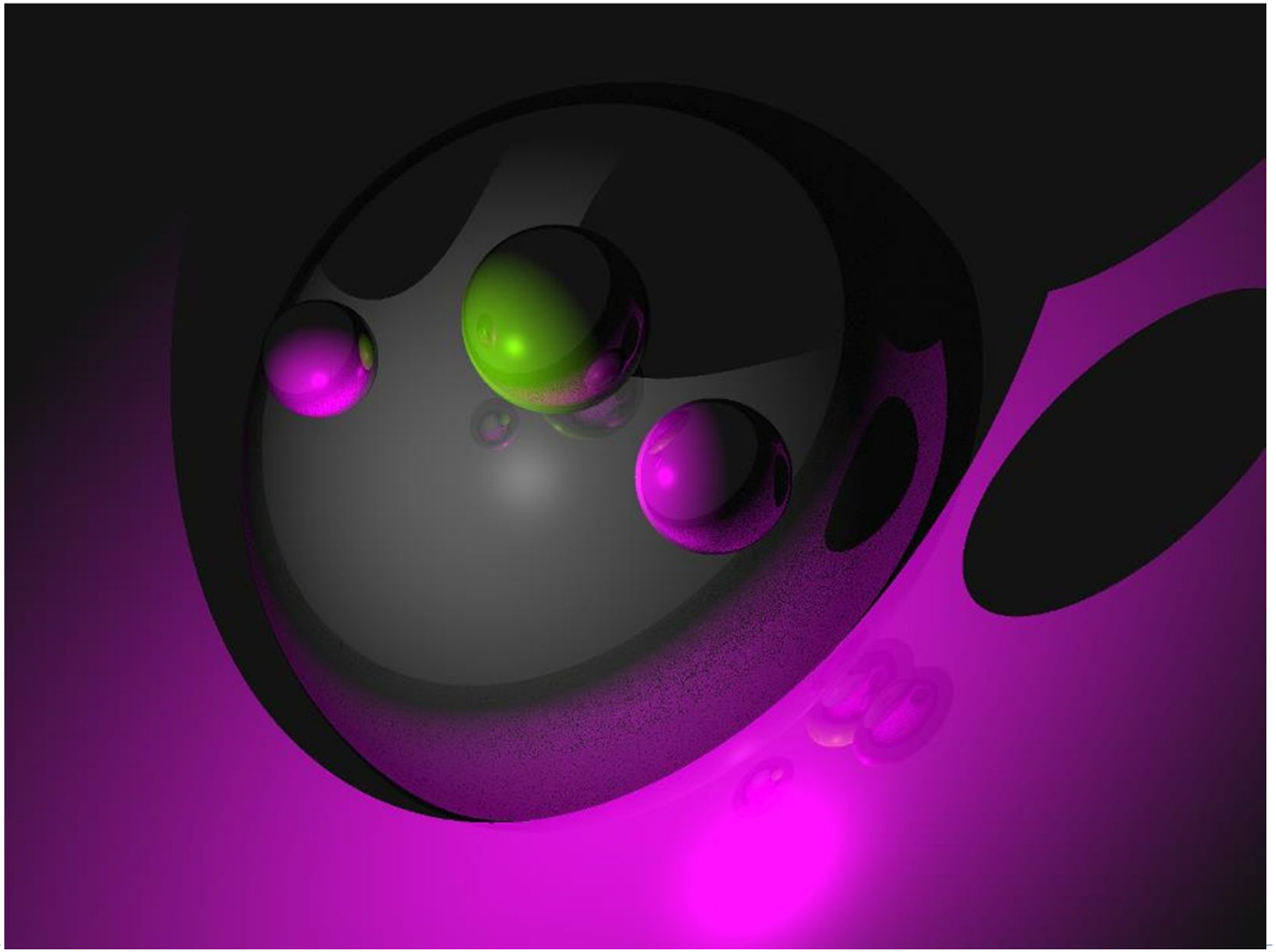
Riflessioni



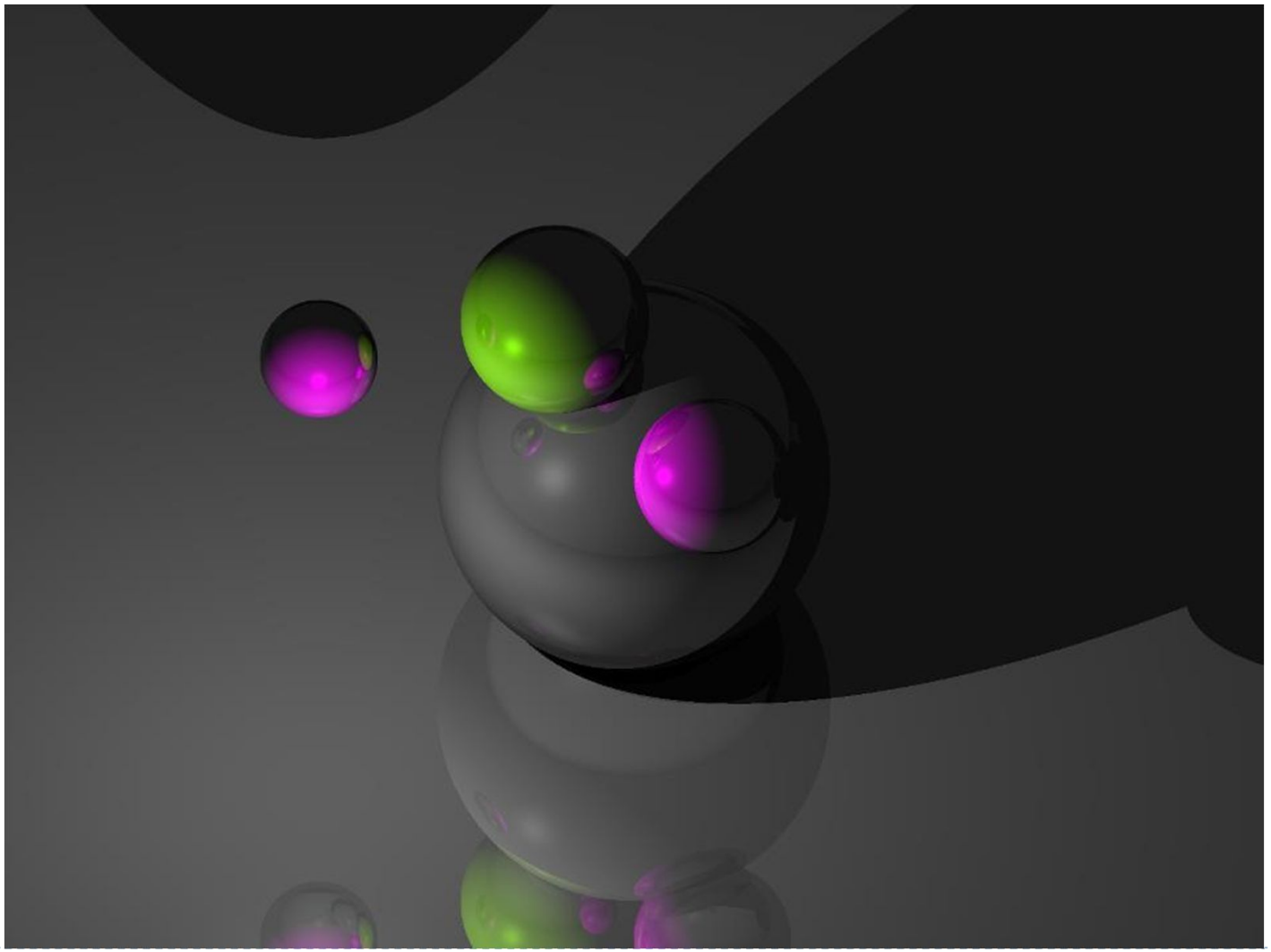
Riflession + illuminazione secondo Blinn



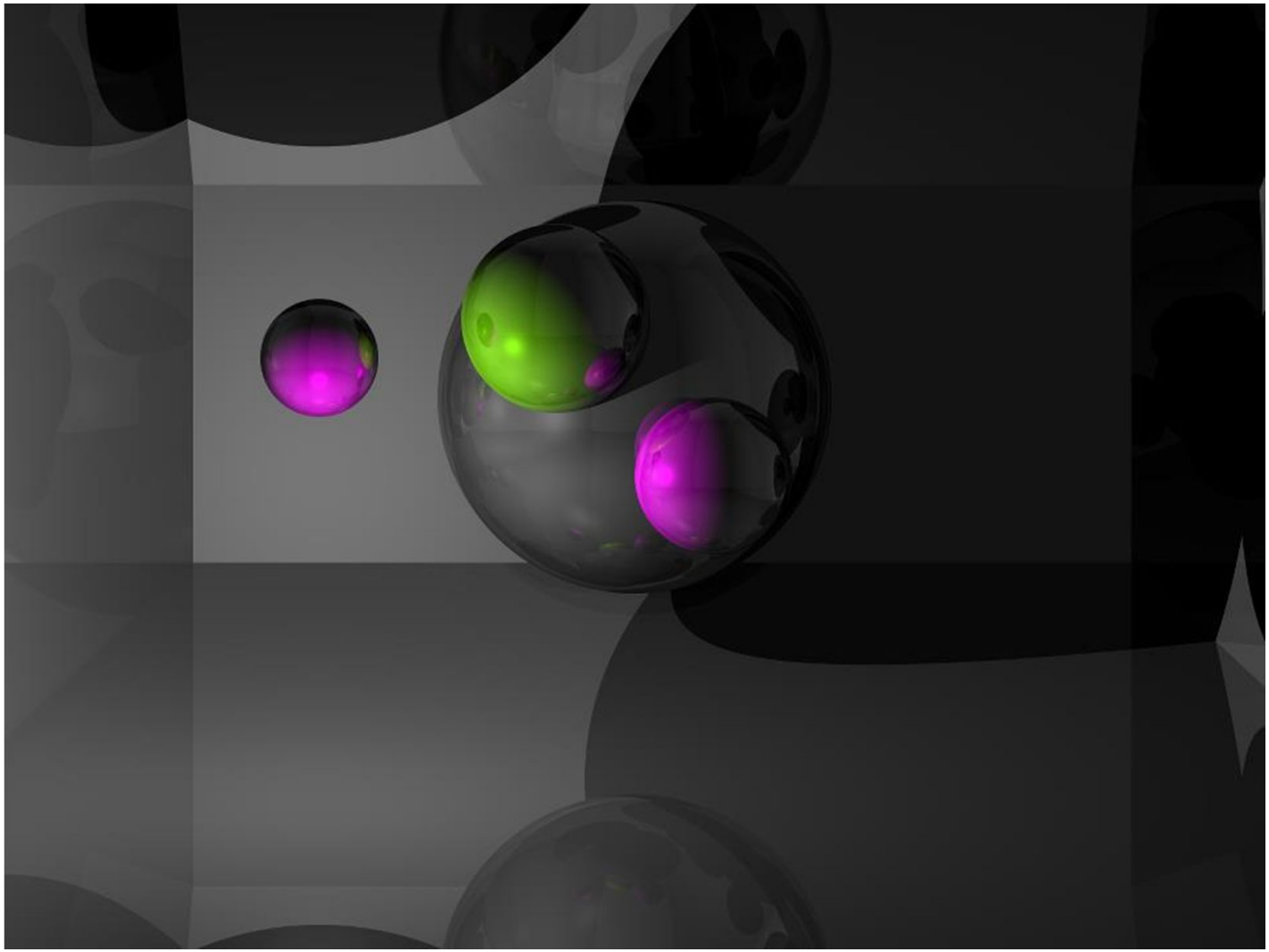
Piani



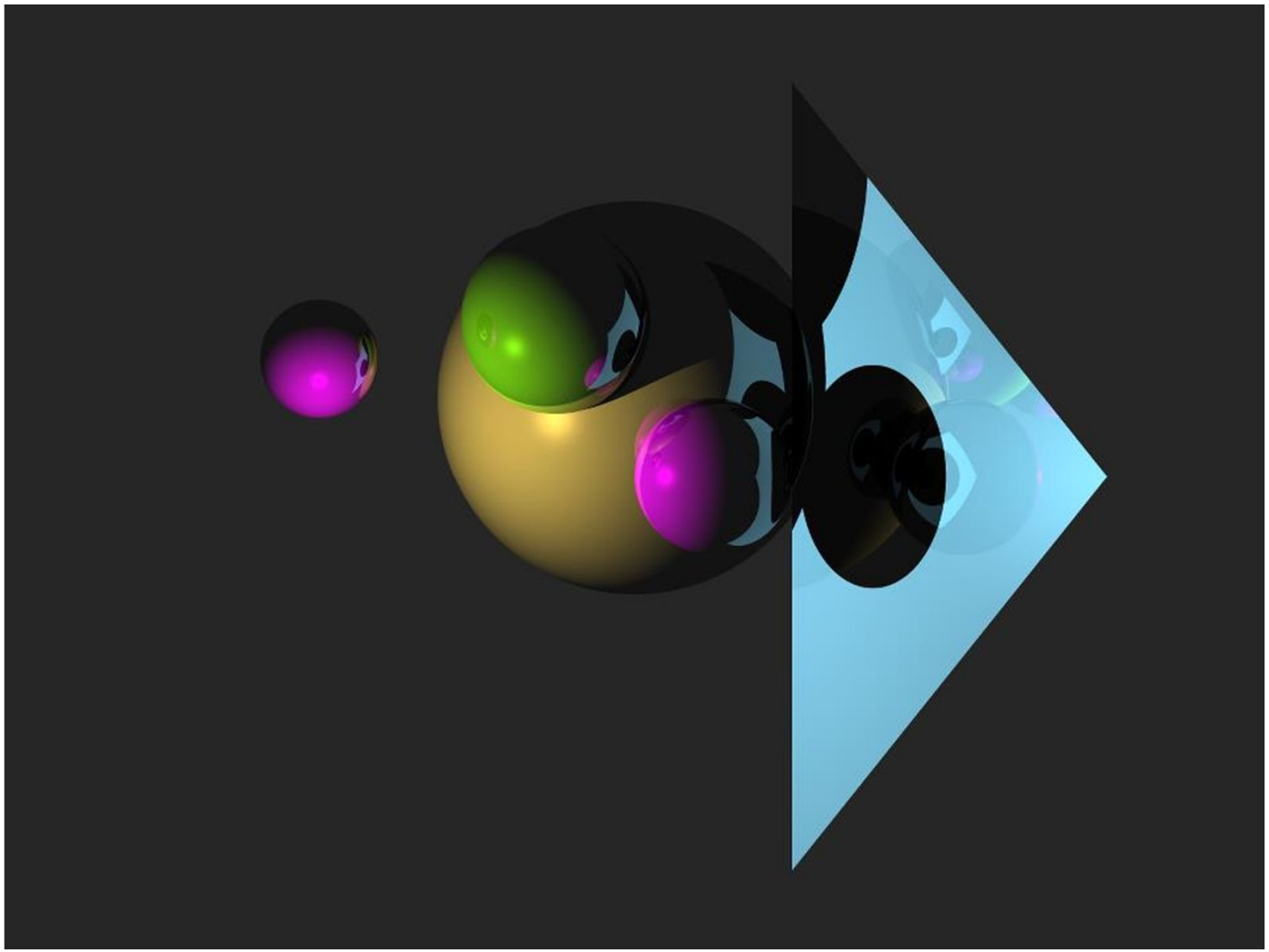
Piani(2)



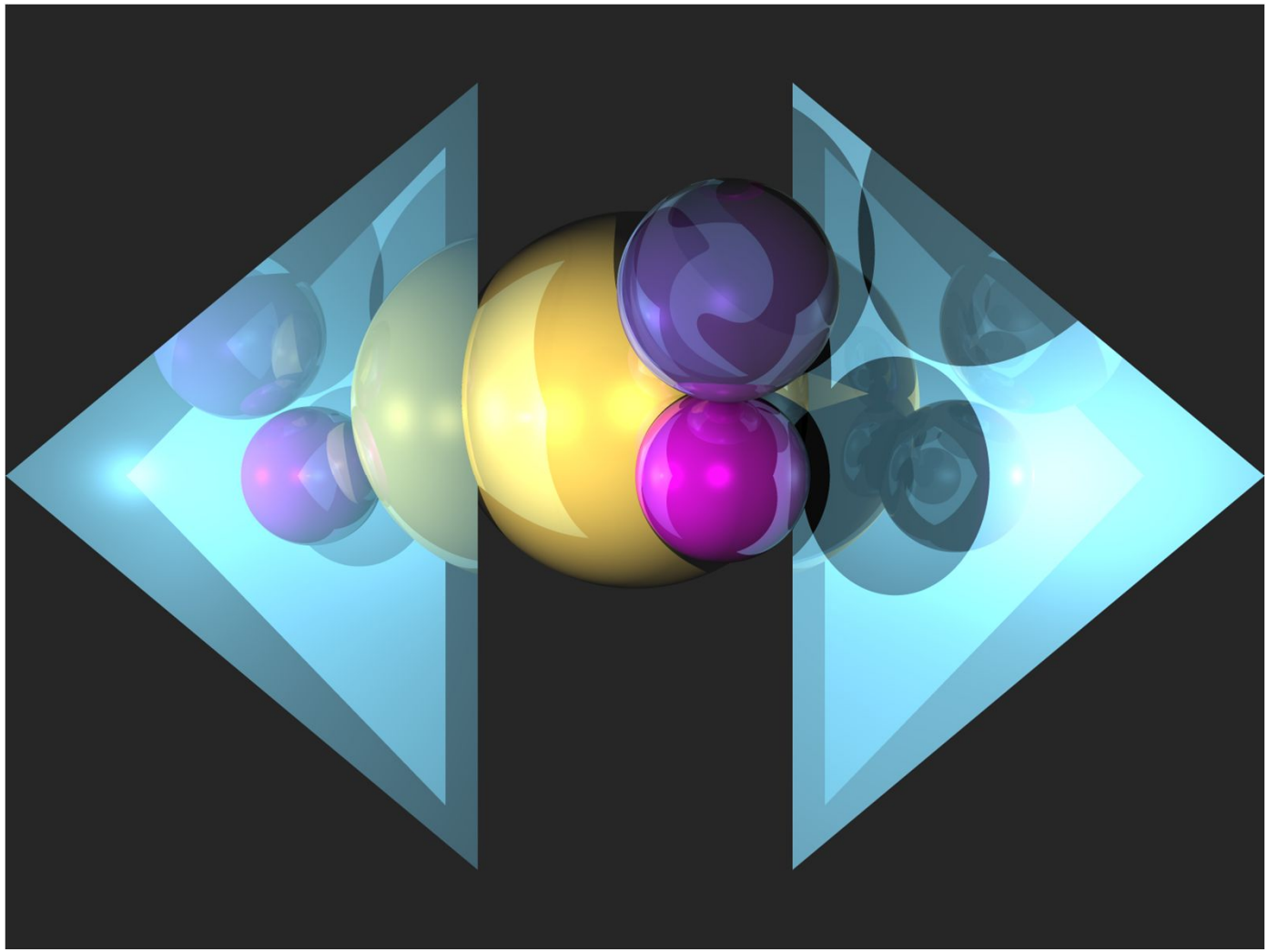
Piani(3) + antialiasing



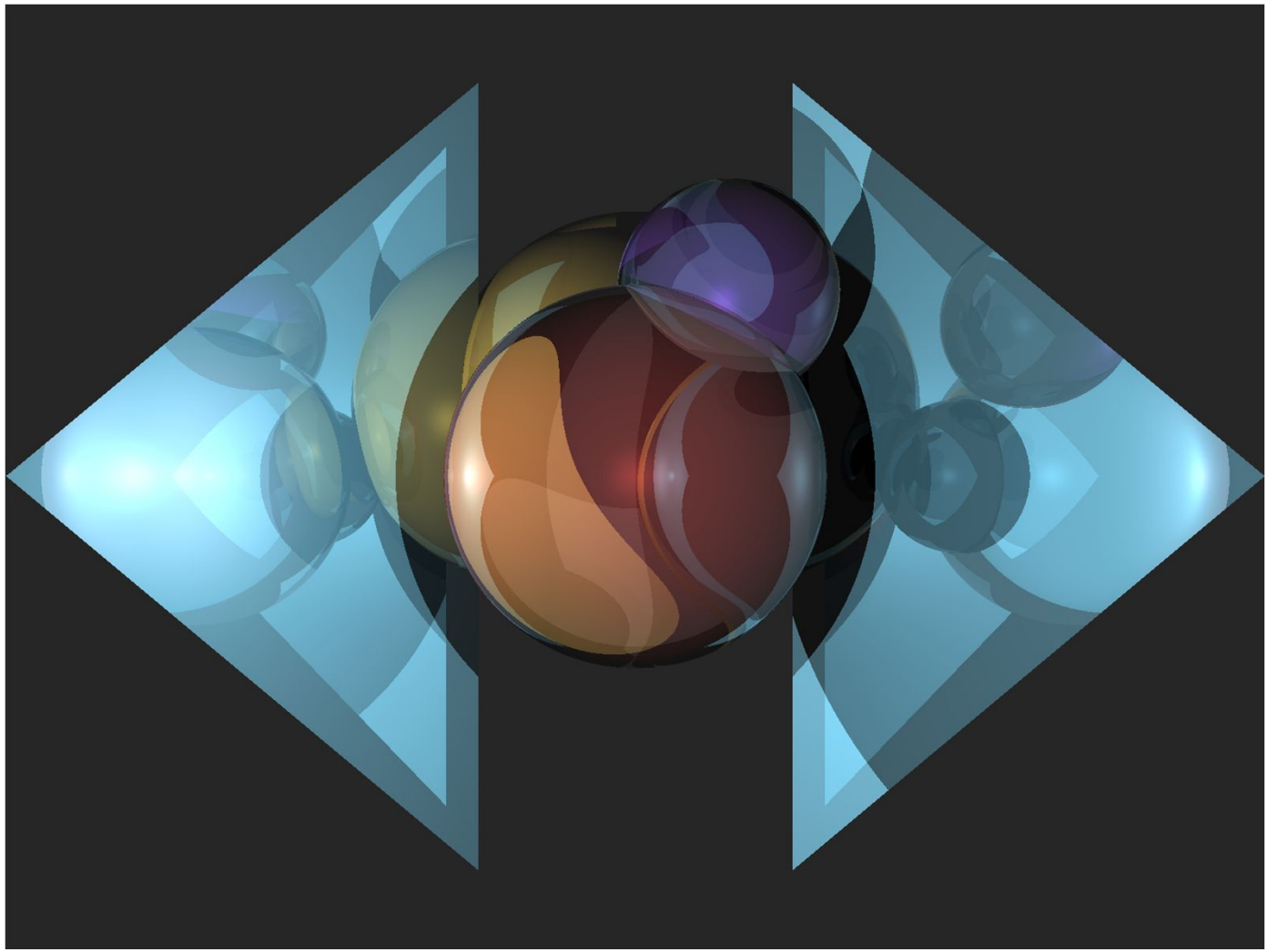
Triangoli



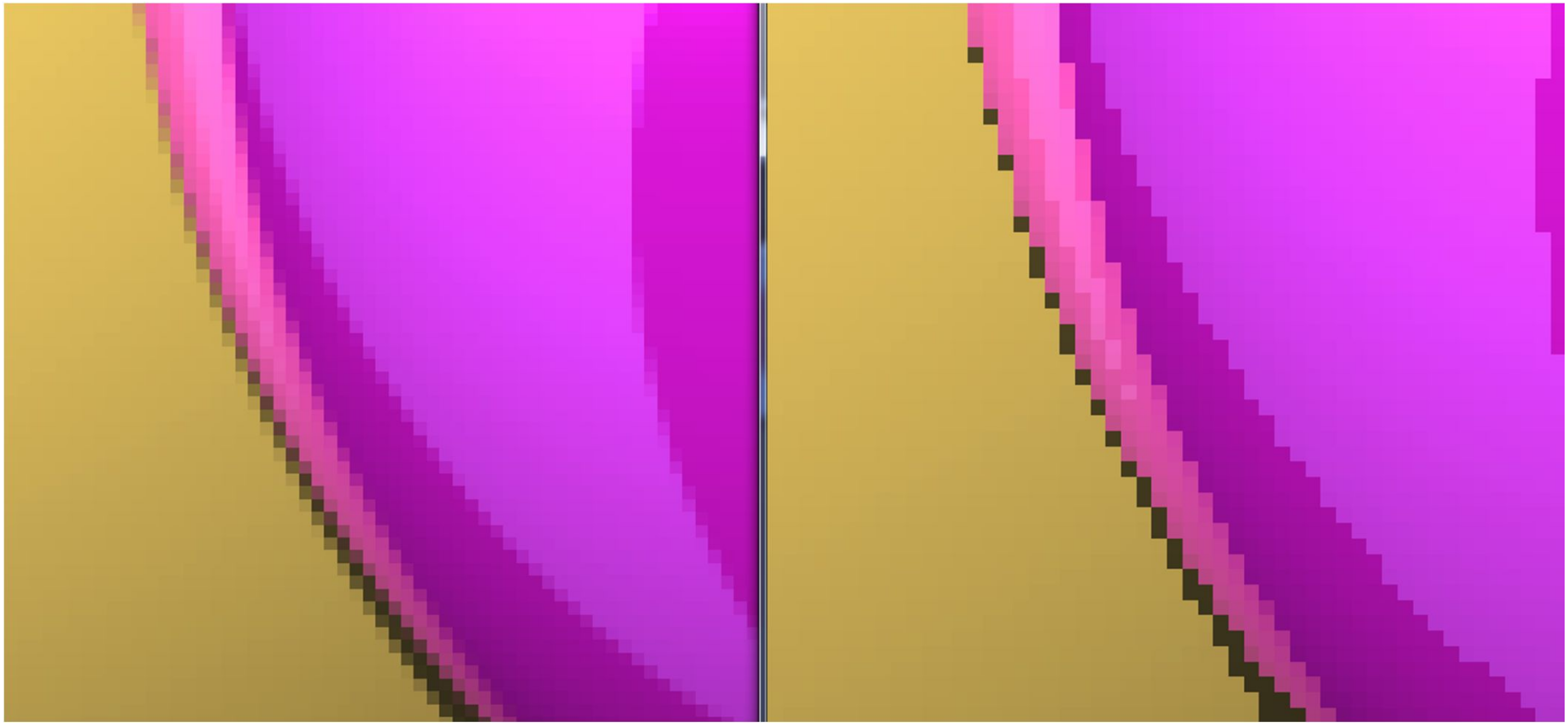
Rifrazione



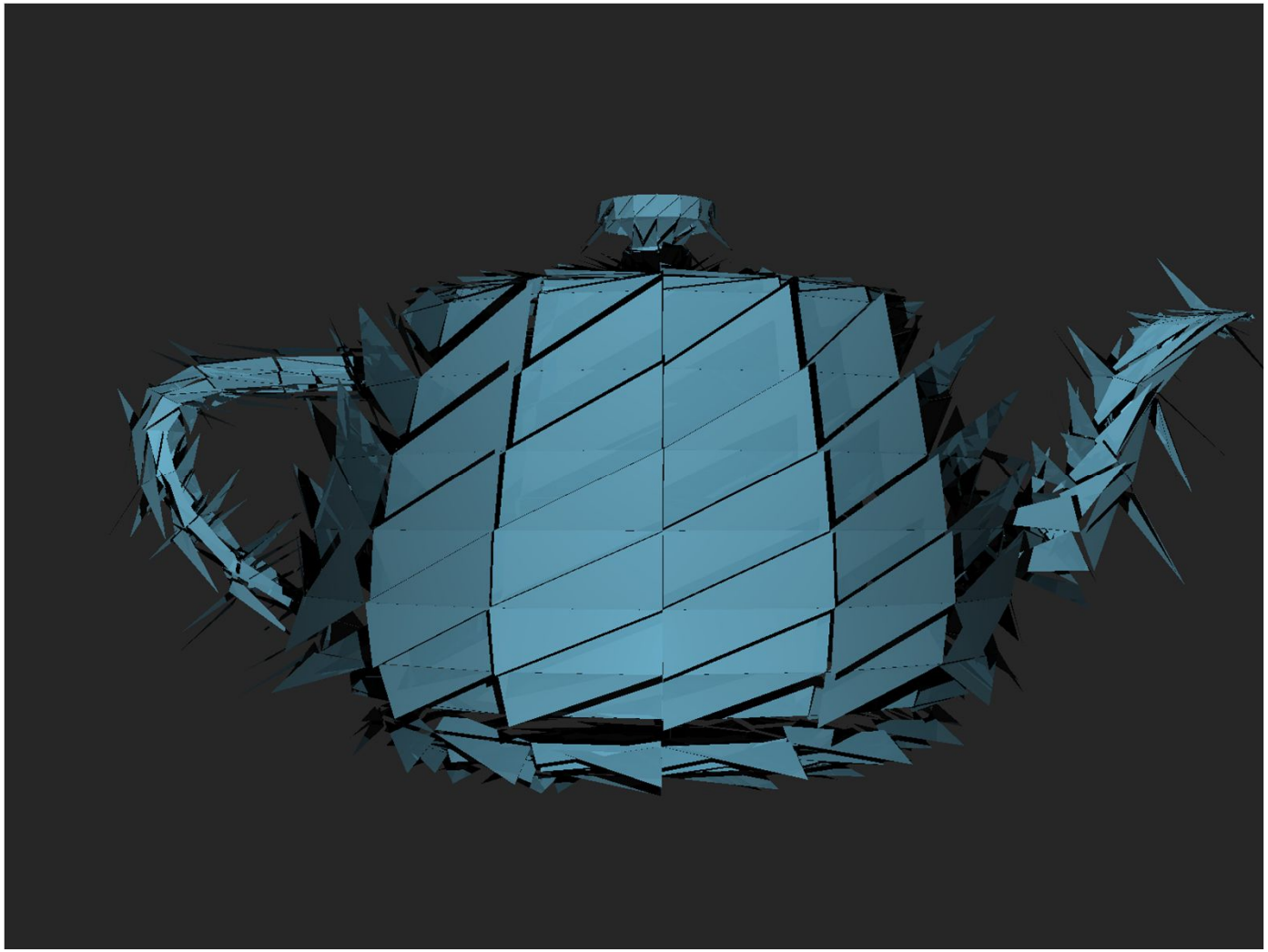
Rifrazione (2)



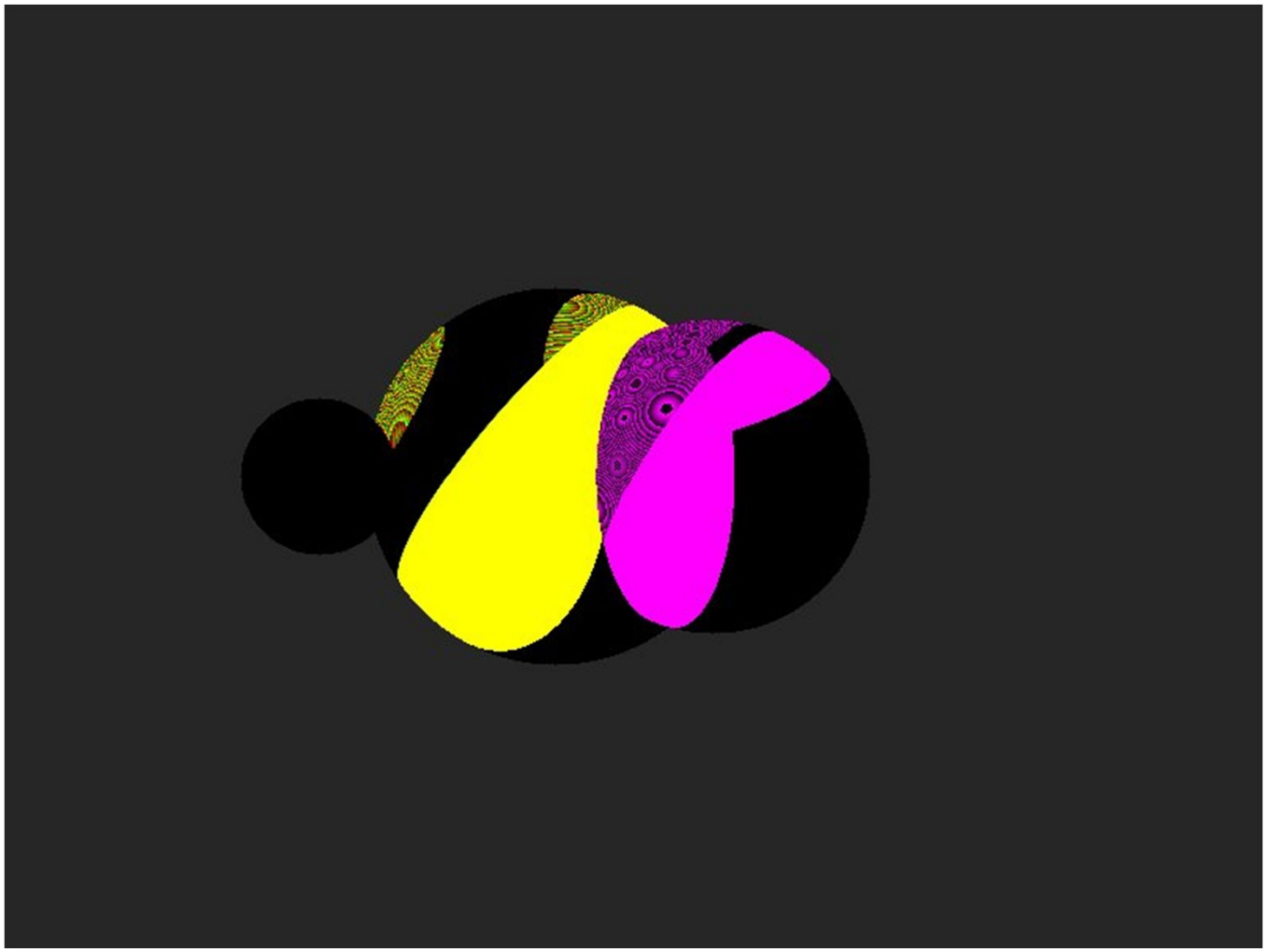
Con AA e senza



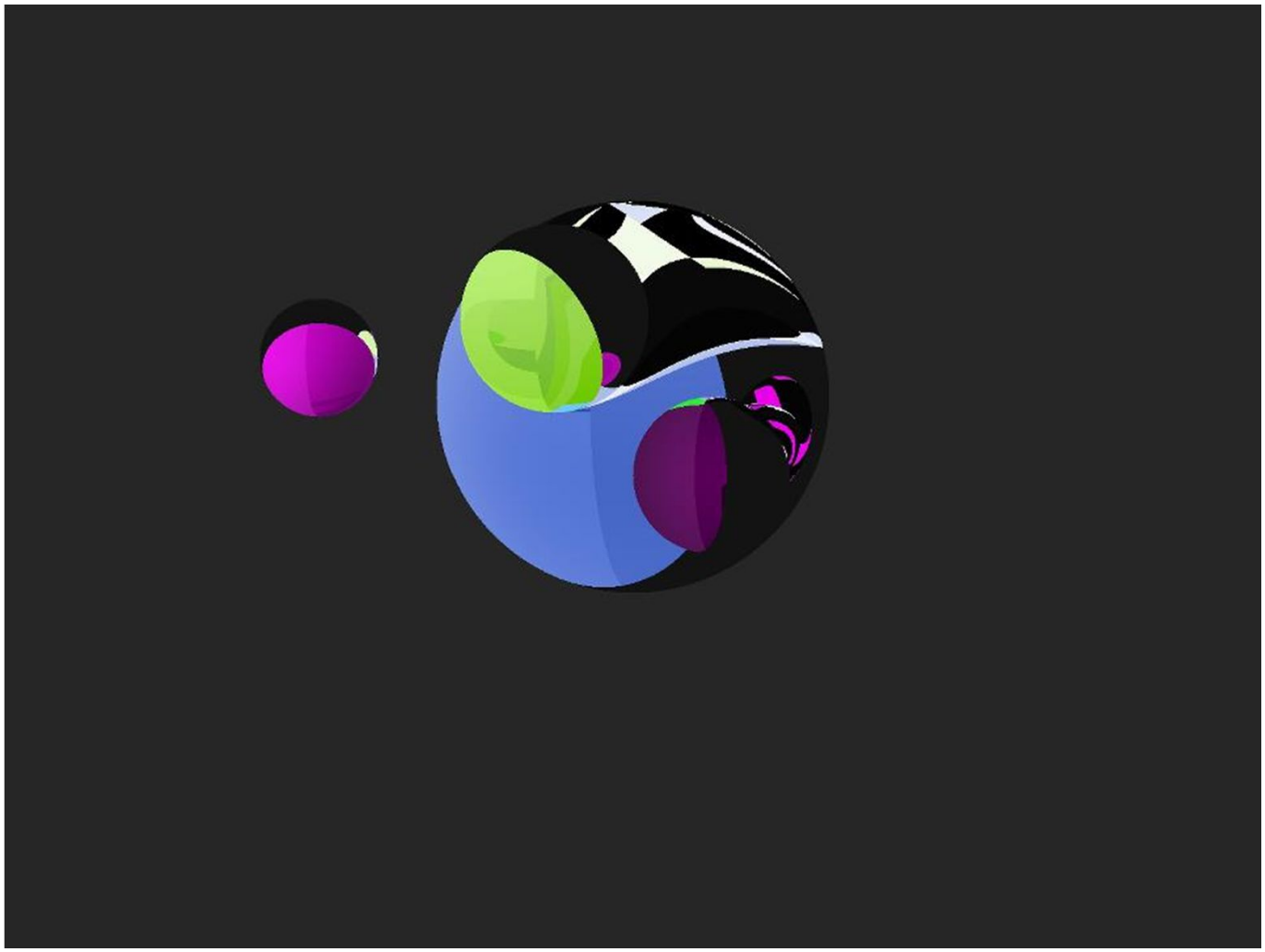
Import da 3ds Max



Tracer + LSD



Tracer + LSD (2)



Team

- Tumaykin Danil
- Carlin Alessandro
- Begnozzi Marco
- Cauzzi Marco

- https://github.com/DrOverclock/Project0