PRÁCTICA 4: Trazado de rayos

4.1.- Escena

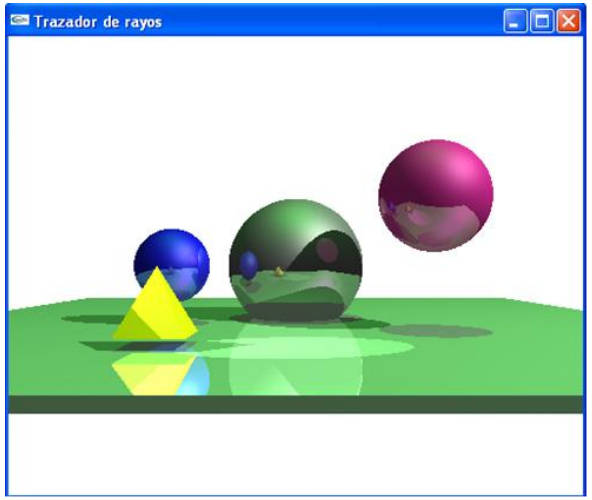
4.2.- Visibilidad

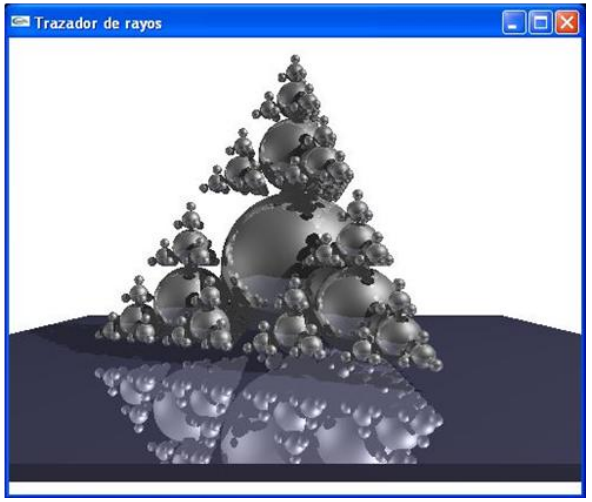
4.3.- Iluminación local

4.4.- Sombras, Reflejos y Sobremuestreo

Trazado de rayos

- El objetivo de la práctica es saber cómo implementar un trazador de rayos sencillo
- 4 partes. Como objetivos de cada parte:
  1. Definir e implementar clases de objetos 3D para organizar geométricamente una escena
  2. Generar y tratar los rayos primarios al atravesar la escena para el cálculo de la visibilidad
  3. Definir e implementar clases de fuentes luminosas e incluirlas en la escena para el cálculo de la iluminación local
  4. Generar y tratar rayos secundarios y resolver problemas de «aliasing»



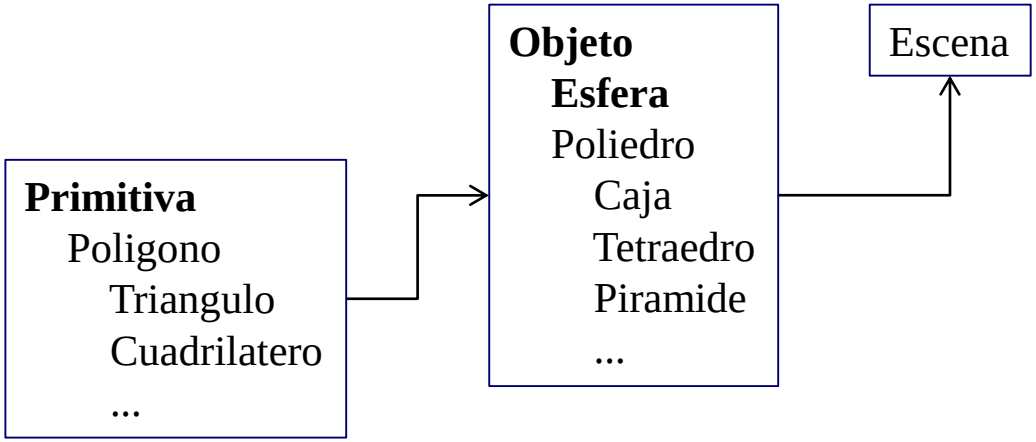


Trazado de rayos

- Pasos a seguir para cada parte:
  - Construcción de objetos 3D
  - Construcción de la escena
  - Generación de rayos primario (trazado)
  - Cálculo de la visibilidad mediante trazado de rayos
  - Construcción de fuentes de luz
  - Inclusión de fuentes de luz en la escena
  - Cálculo de la iluminación local
  - Ampliaciones a rayos secundarios

4.1 Construcción de objetos 3D

- Se debe construir una jerarquía de Objetos 3D que formarán la escena y las clases derivadas de Primitiva necesarias para construir los objetos
- Se debe construir una clase Escena. Cada escena debe manejar su estructura dinámica de objetos y responder adecuadamente al trazado de un rayo



Ejemplo de estructura de clases

4.1 Construcción de objetos 3D

```
// Clase madre. Se deben implementar los métodos virtuales en cada clase derivada
class Objeto
{
public:
    Color colDifuso, colEspecular;          // Colores difuso y especular
    float ka,kd,ks;                         // Factores ambiental, difuso y especular
    int m;                                  // Exponente de concentración de brillo
    Objeto();
    virtual ~Objeto();
    virtual Vector normal(Punto p)const;    //Devuelve la normal en p
    virtual int rayIntersection(Punto p, Vector v, float &t)const;
    //Devuelve si hay intersección (1) o no (0). Si hay, se produce en P(t)= p + v*t
    void setColor(Color cd, Color ce=Color::BLANCO,
                  float Ka=0.3f, float Kd=0.8f, float Ks=0.3f, int em=1);
};
```

4.1 Construcción de objetos 3D

```
// Clase derivada de objeto que se proporciona como ejemplo
class Esfera: public Objeto
{
protected:
    Punto centro;
    float radio;
public:
    Esfera();
    ~Esfera();
    Esfera(Punto centro, float radio, Transformacion t=Transformacion());
    // Constructor (centro, radio, escalado)
    Vector normal(Punto p)const;
    int rayIntersection(Punto p, Vector v, float &t)const;
};
```

4.1 Construcción de objetos 3D

- El resto de objetos se construirán según criterio del alumno y deben cumplir los siguientes requisitos:
  - Debe haber un constructor que permita situar el objeto en la escena, posiblemente pasando al constructor una transformación
  - Deben implementarse los métodos

    Vector normal(Punto p) const;
    - Devolverá la normal en el punto **p** de la superficie del objeto

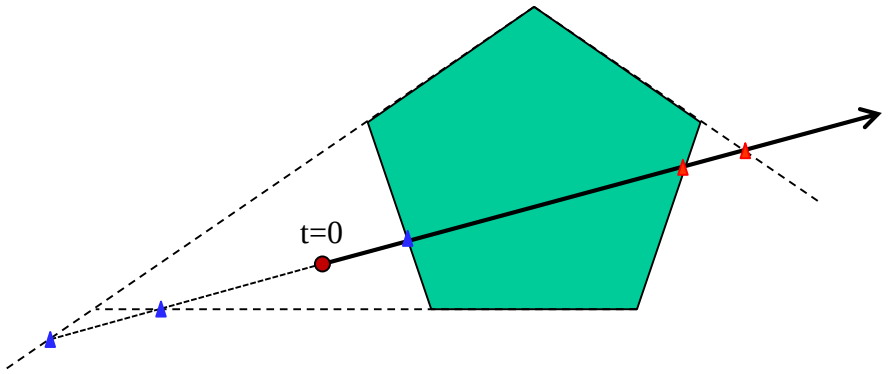
    int rayIntersection(Punto p, Vector v, float &t) const;
    - Devolverá en ***t*** la intersección primera entre una semirrecta que parte del punto **p** y sigue la dirección del vector **v**. ***t*** es el valor del parámetro de la recta para el que se produce la intersección. $P(t)= p + v \cdot t$
    - La función devolverá:

      0, si el rayo no interseca con el objeto o es tangente a él

      1, si la intersección existe
    - opcionalmente se podría devolver un valor diferente para distinguir, por ejemplo, que la intersección es interior al objeto (**p** dentro)

4.1 Construcción de objetos 3D

- Como mínimo debe implementarse la clase Caja
- Se sugiere utilizar el método de Haines (max entrante , min saliente) para calcular la intersección con el rayo. Este método evita tener que determinar la interioridad de puntos a polígonos



4.1 Construcción de la escena

- La clase Escena debe gestionar una lista indexada de objetos
- La lista de objetos será preferiblemente dinámica
- Escena debe implementar los métodos públicos

  int **add** (Objeto *o); /* Añade el objeto *o* a la lista */

  Color **rayTrace** (Punto inicio, Vector direccion) const;

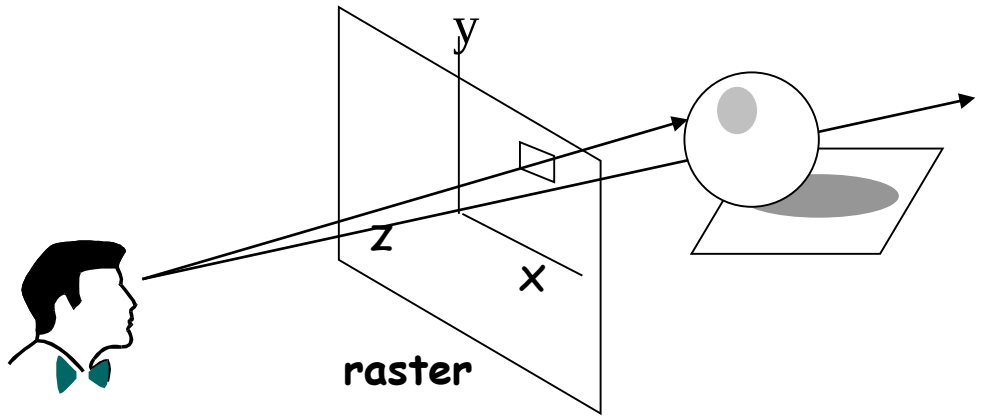
  /* Devuelve el color del primer objeto alcanzado por el rayo que parte de *inicio* y lleva *direccion* */
- Se proporcionan los ficheros *Escena.h* y *Escena.cpp* parcialmente desarrollados

4.2 Visibilidad por trazado de visuales

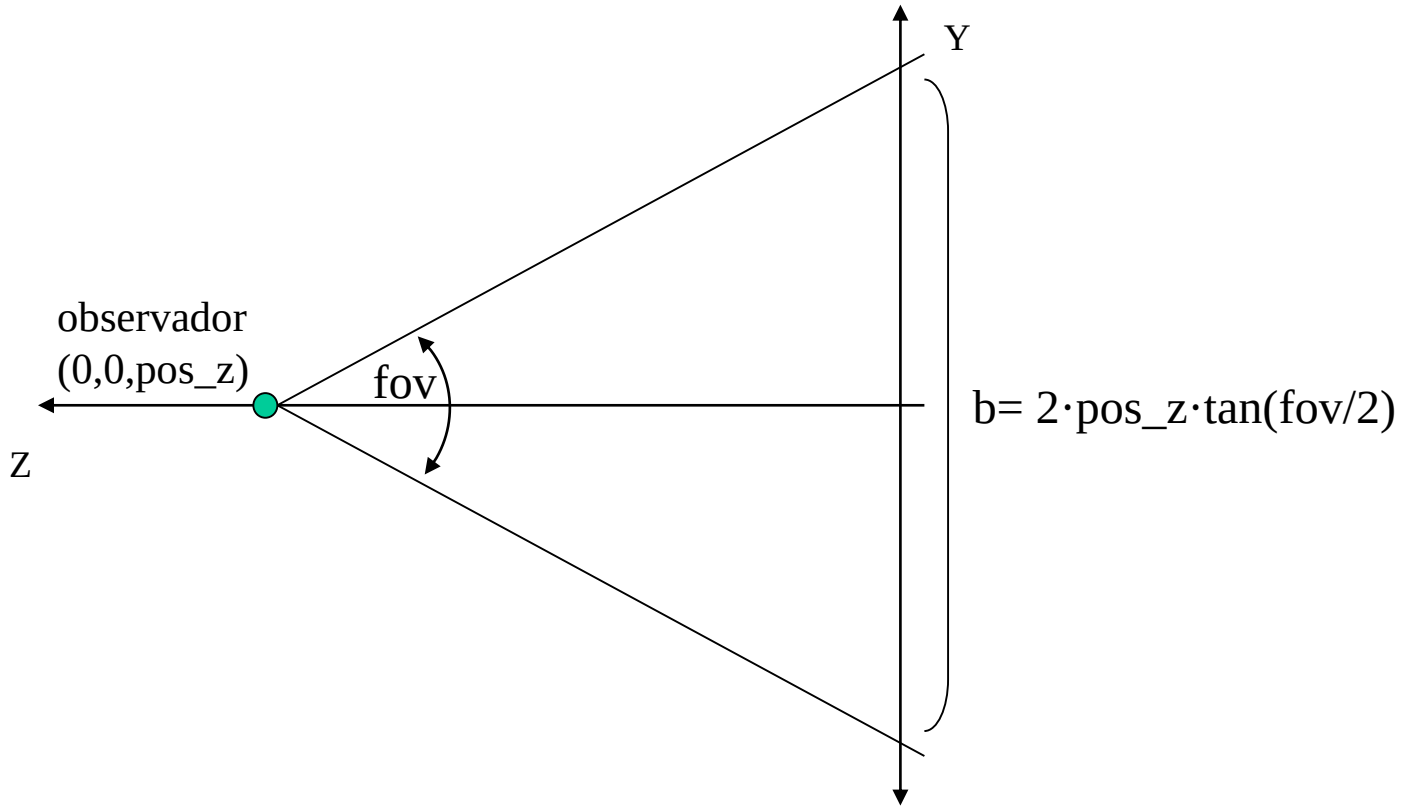
- Objetivos
  - Llegar a dominar la resolución del problema de la visibilidad por el trazado de visuales como paso previo a la iluminación por trazado de rayos
  - Representar una escena resolviendo el problema de la visibilidad mediante el método de trazado de visuales (*ray casting*) por los píxeles del raster detectando la intersección con el objeto más cercano

4.2 Visibilidad por trazado de visuales

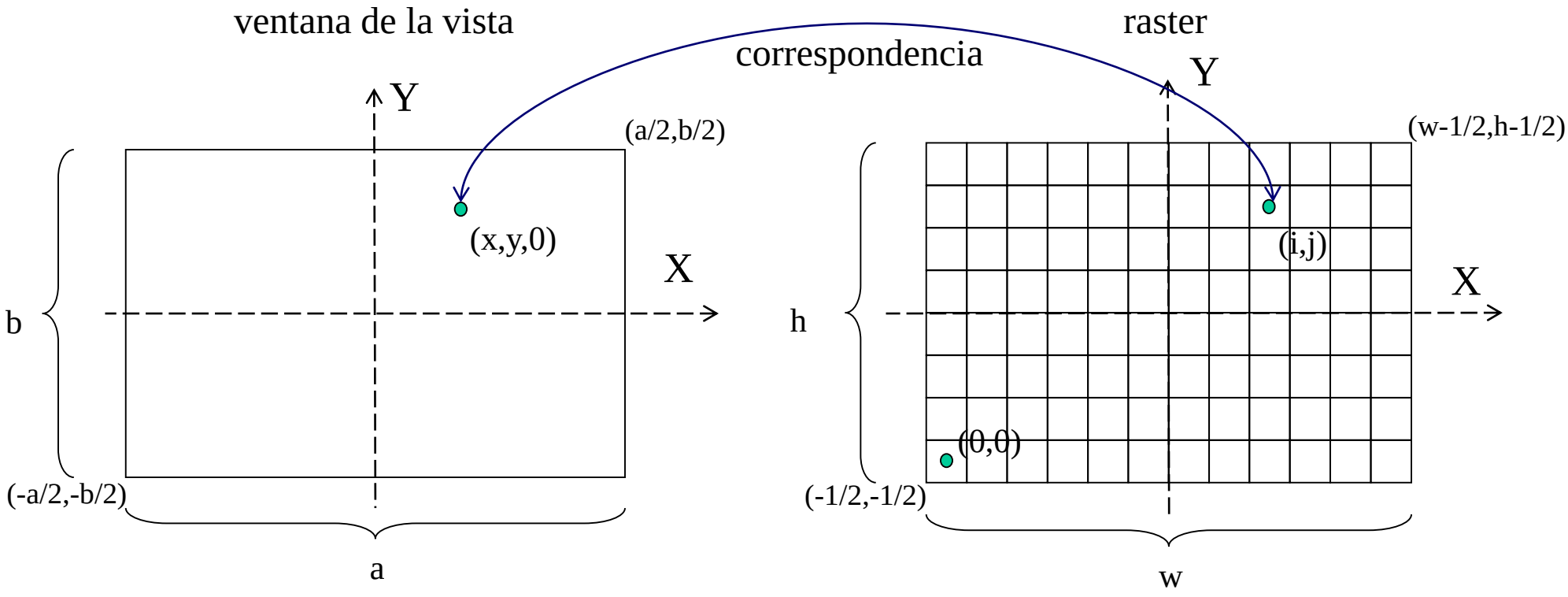
- Pasos a seguir
  1. Construir la escena
  2. Construir la estructura de datos «raster»
  3. Definida la vista, construir visuales (rayos primarios) que pasan por los centros de los píxeles
  4. Evaluar en la escena el primer objeto visto desde cada visual
  5. Poner cada píxel al color difuso del objeto obtenido antes
  6. Copiar la estructura «raster» a la pantalla



4.2 Definición de la vista



4.2 Correspondencia entre sistemas



b= f(pos_z,fov)	alto de la ventana
a/b=w/h	isotropía

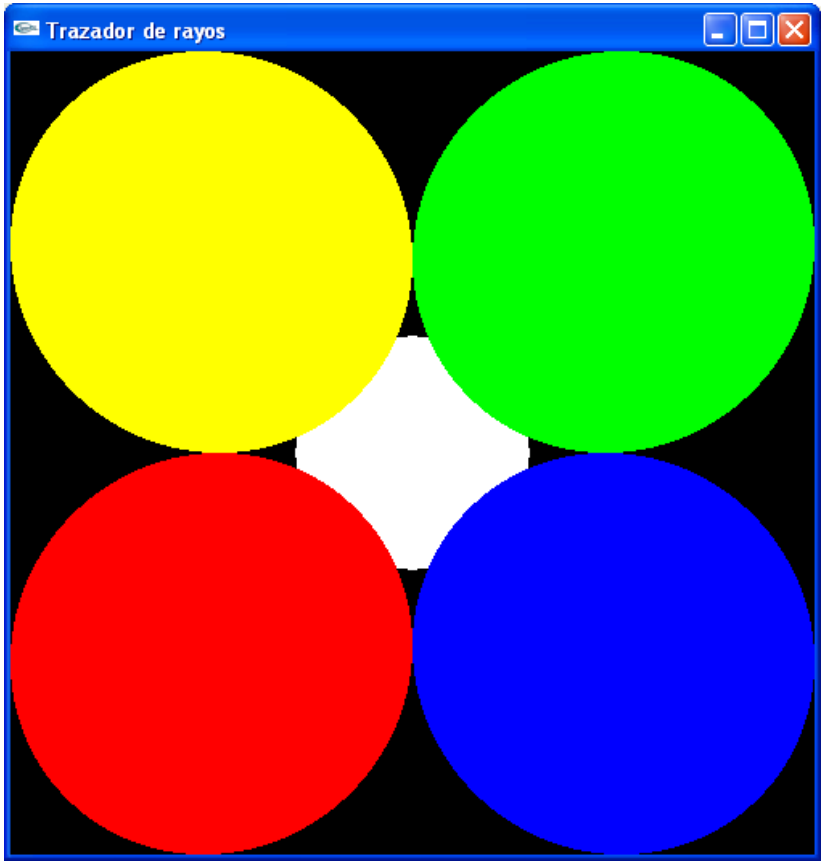
x= f(i,w,a)	correspondencia entre rectángulos
y= f(j,h,b)	

4.2 Trazando visuales

```
si raster = null, crear raster
b= f(pos_z,fov)
a= b* w/h
unsigned char * t = raster
para cada fila j del raster
    y= f(j,b,h)
    para cada columna i del raster
        x=f(i,a,w)
        color= escena->rayTrace(Punto(0,0,pos_z),Vector(x,y,-pos_z))
        *t++=(unsigned char)(color.r()*255);
        *t++=(unsigned char)(color.g()*255);
        *t++=(unsigned char)(color.b()*255);
el raster está listo
```

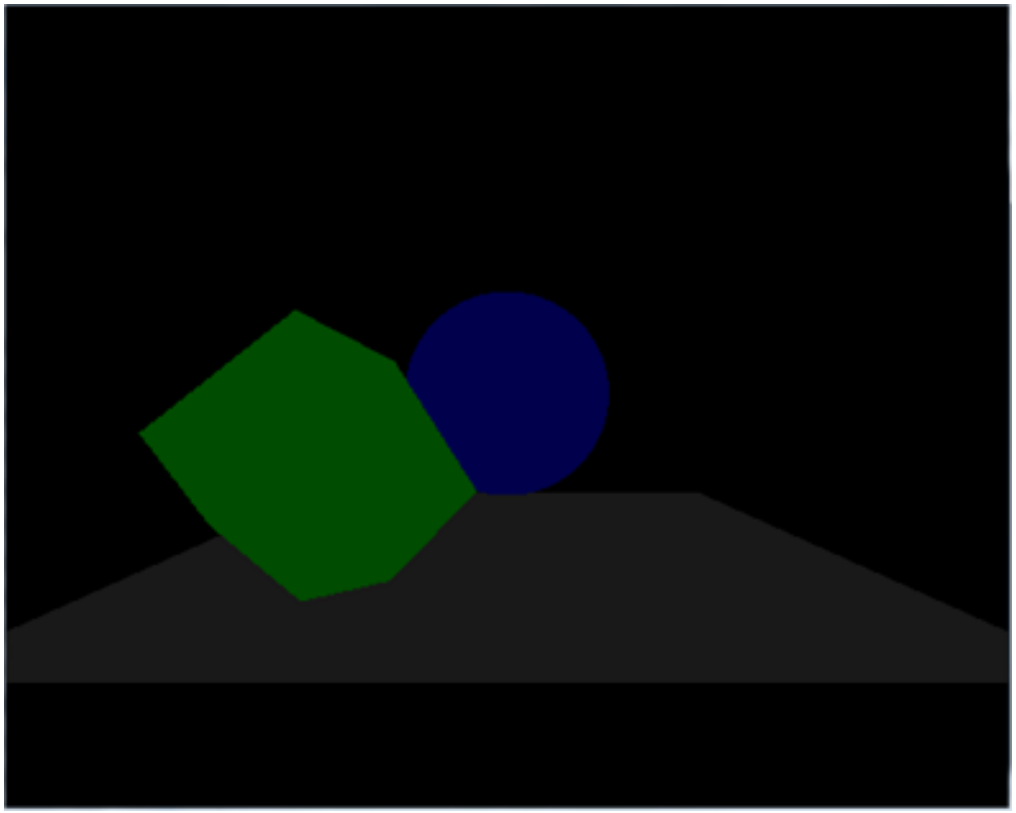
4.2 Visibilidad por trazado de visuales

- Se suministra incompleto *Trazador.cpp*
  - completar código
  - comprobar con esta imagen



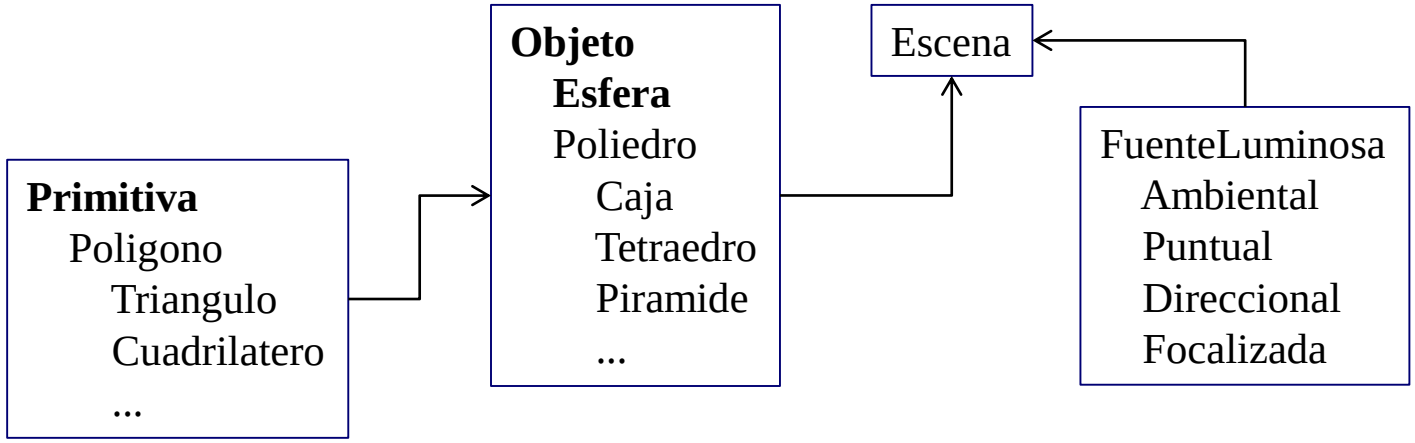
4.2 Escena propia

- Una vez comprobado el funcionamiento se debe usar el trazador sobre una escena propia que contenga poliedros (cajas p.e)



4.3.- Iluminación local

- Objetivos:
  - Aplicar los conceptos de iluminación local del modelo de Phong
  - Visualizar una escena con iluminación
- Pasos a seguir:
  - Crear la jerarquía de clases de fuentes de luz con los atributos necesarios para caracterizar cada una de ellas.
  - Añadir a la escena fuentes luminosas y calcular la iluminación local



Ejemplo de estructura de clases

4.3.- Iluminación local

```
#define ON 1
#define OFF 0
class FuenteLuminosa
{
protected:
        Color I;
        int encendida;
        Punto posicion;
public:
        FuenteLuminosa();
        FuenteLuminosa(Color intensidad, Punto pos=Punto());      //Constructor
        virtual Color intensity(Punto p)const;                    //Devuelve la intensidad vista desde p
        virtual Vector L(Punto p)const;                           //Vector unitario desde p hacia la luz
        void setColor(Color c);                                   //Fija el color (intensidad)
        Punto position() const;                                   //Devuelve la posición de la luz
        void setPosition(Punto pos);                              //Fija la posicion de la luz
        void switchLight(int);                                    //Enciende (1) o apaga (0)
        int switchOn()const;                          //Devuelve el estado de la luz (1) encendida, (0) apagada
};
```

4.3.- Iluminación local

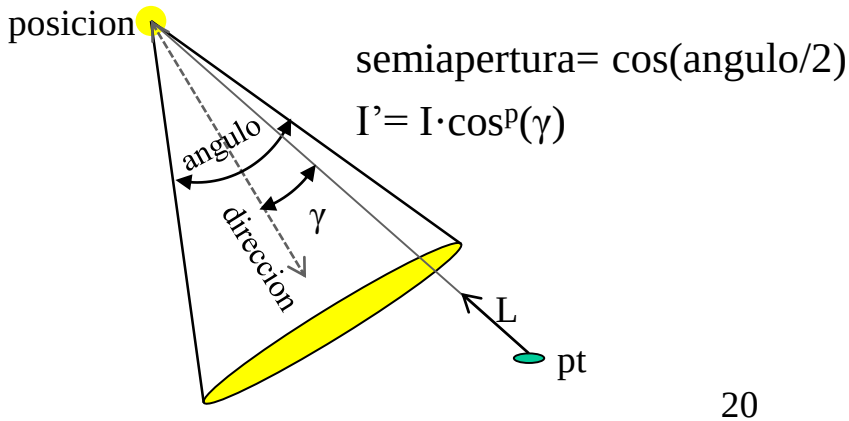
```
class Ambiental: public FuenteLuminosa
{
public:
        Ambiental();                                    //Sólo debe haber una fuente ambiental
        Vector L(Punto p) const;                        //Devuelve el vector nulo. No es aplicable
};

class Puntual: public FuenteLuminosa
{
public:
        Puntual();
        Puntual(Color c, Punto pos=Punto());            //Sitúa la fuente puntual y fija su intensidad
};

class Direccional: public FuenteLuminosa
{
protected:
        Vector direccion;
public:
        Direccional();
        Direccional(Color intens, Vector direc=Vector(0,-1,0));   //Define la intensidad y la dirección de iluminación
        void setDirection(Vector d);                    //Cambia la dirección de iluminación
        Vector L(Punto p) const;                        //Devuelve la dirección unitaria de iluminación negada
};
```

4.3.- Iluminación local

```
class Focalizada: public FuenteLuminosa
{
protected:
        Vector direccion;     // Dirección de central del foco
        float p;              // Exponente que regula la distribución (concentración) de la intensidad en el foco
        float semiapertura;   // Coseno entre el eje y la generatriz del cono
public:
        Focalizada();
        Focalizada(Color inten, Punto posic, Vector dir, float concentracion, float angulo);
        Color intensity(Punto pt) const;          //Intensidad = 0 si cos(γ) < semiapertura
        void setShape(Vector dir, float concentracion, float angulo);
};
```



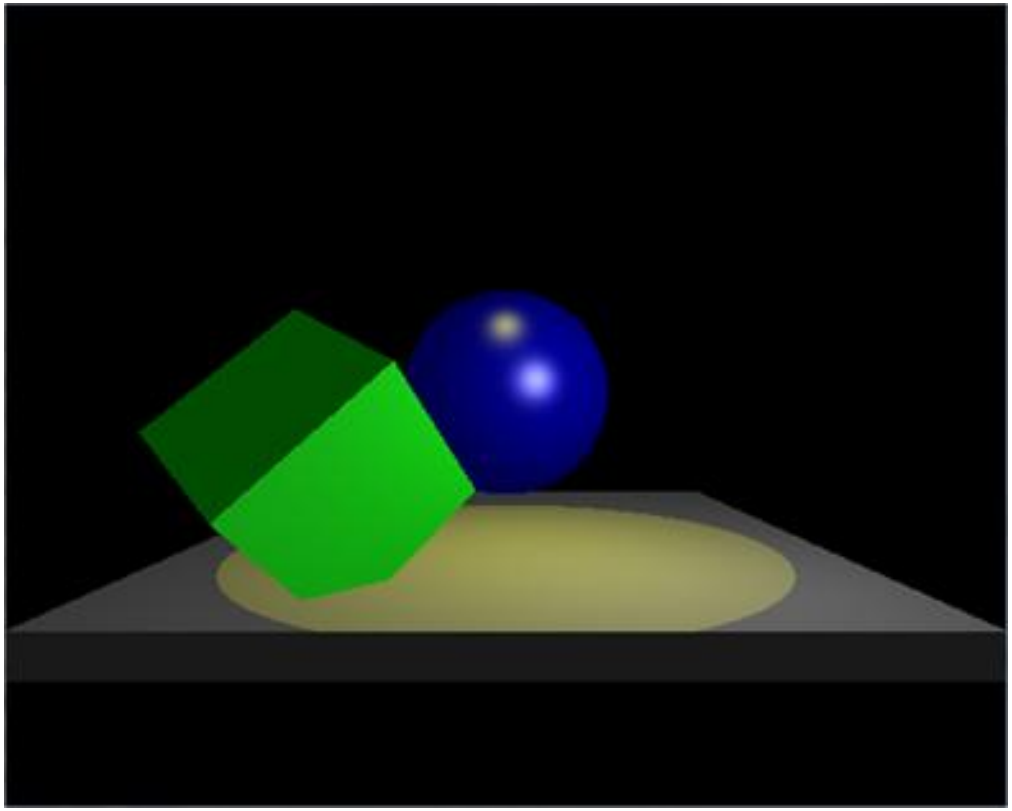
4.3.- Iluminación local

- Recomendaciones para fuentes luminosas
  - Las fuentes luminosas no uniformes deben reimplementar el método Color intensity(Punto p)
  - La fuente ambiental reimplementa el método Vector L(Punto p) devolviendo el vector nulo
  - La fuente direccional reimplementa el método Vector L(Punto p). *L* no depende de *p* en este caso

- Modificaciones en Escena
  - Deberá añadirse un nuevo dato miembro que mantenga las fuentes luminosas presentes en la escena con métodos de gestión y consulta. La estructura de datos será preferiblemente dinámica
  - Se recomienda usar una única fuente ambiental y situarla la primera en la lista de fuentes (luz 0) siempre encendida
  - Para devolver el color en Escena::rayTrace() hay que calcular la iluminación local en el punto de contacto del rayo con el objeto más próximo

4.3 Escena propia

- En Trazador
  - Al pulsar ‘L’ activar la iluminación
  - Al pulsar ‘l’ desactivar la iluminación



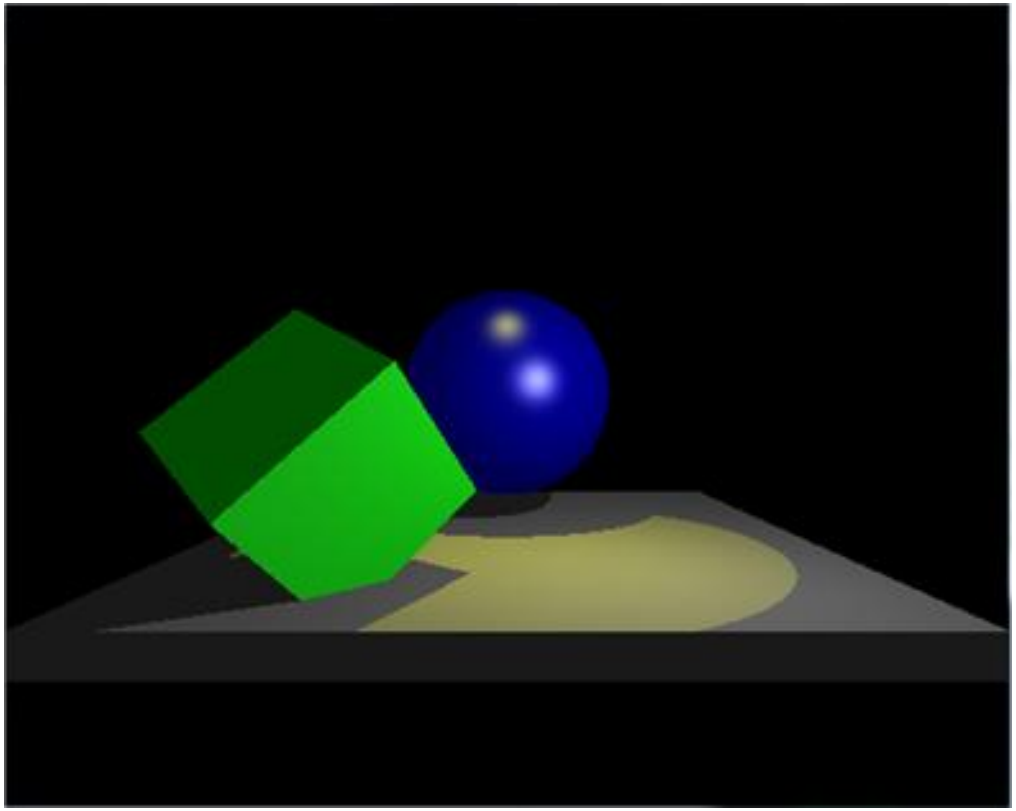
4.4 Sombras

- Modificar el modelo de iluminación para incluir visibilidad de las luces
- La luz contribuye a la iluminación del punto si
  - Está encendida
  - Está por encima de la superficie (NL>0)
  - El punto está dentro del cono de luz –caso focal-
  - No hay ningún objeto entre el punto y la luz –rayo de sombra-

- Rayos de sombra
  - Se originan en el punto con dirección hacia la luz
  - Se trazan por la escena buscando la intersección más cercana
  - Se pueden producir problemas de autosombra

4.4 Escena propia

- En Trazador
  - Al pulsar ‘S’ activar las sombras
  - Al pulsar ‘s’ desactivar las sombras



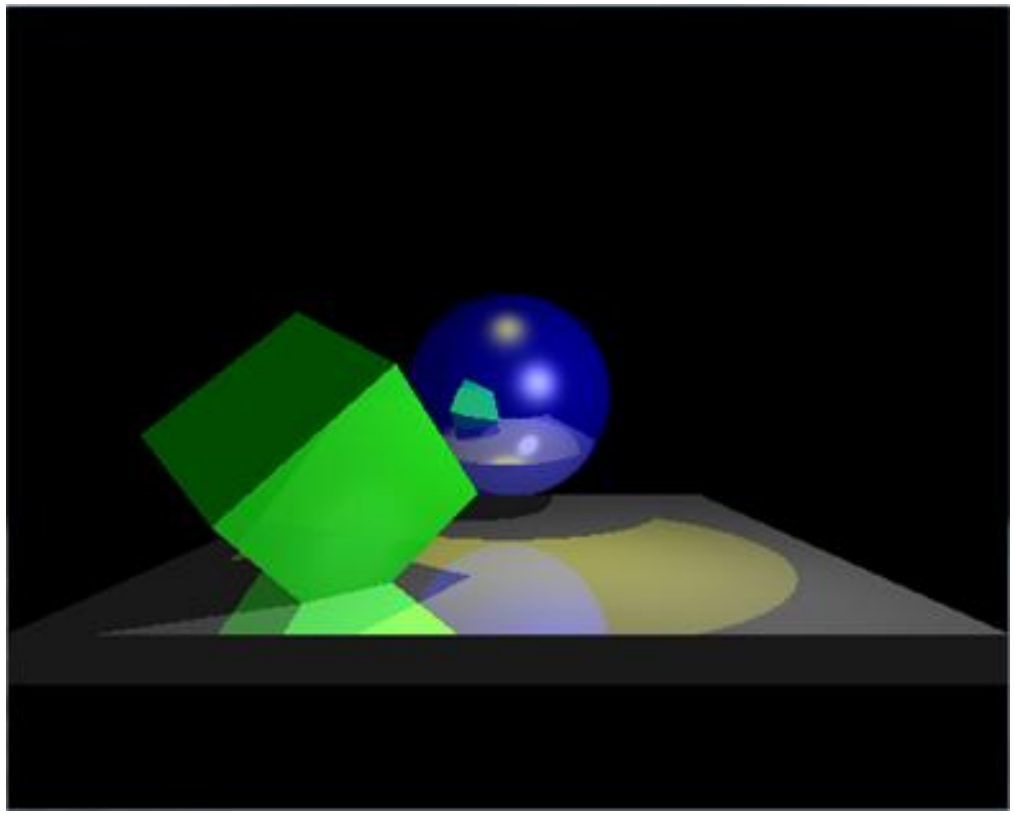
4.5 Reflexiones

- Las reflexiones de unos objetos sobre otros se consiguen:
  - Calculando el rayo reflejado de V (R)
  - Calculando la intensidad que llega por el rayo reflejado (I_r)
  - Aplicando el modelo de iluminación de Whitted ($I+=k_sI_r$)

- El proceso puede repetirse según la profundidad del árbol del rayos

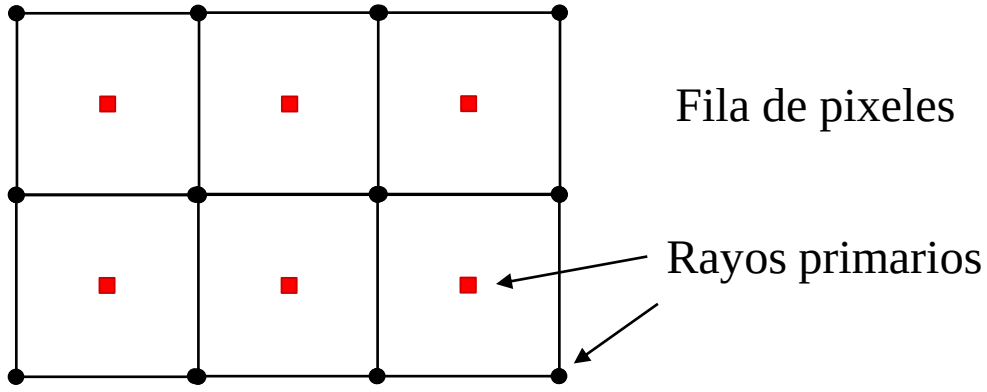
4.5 Escena propia

- En Trazador
  - Al pulsar ‘R’ activar reflexiones
  - Al pulsar ‘r’ desactivar reflexiones



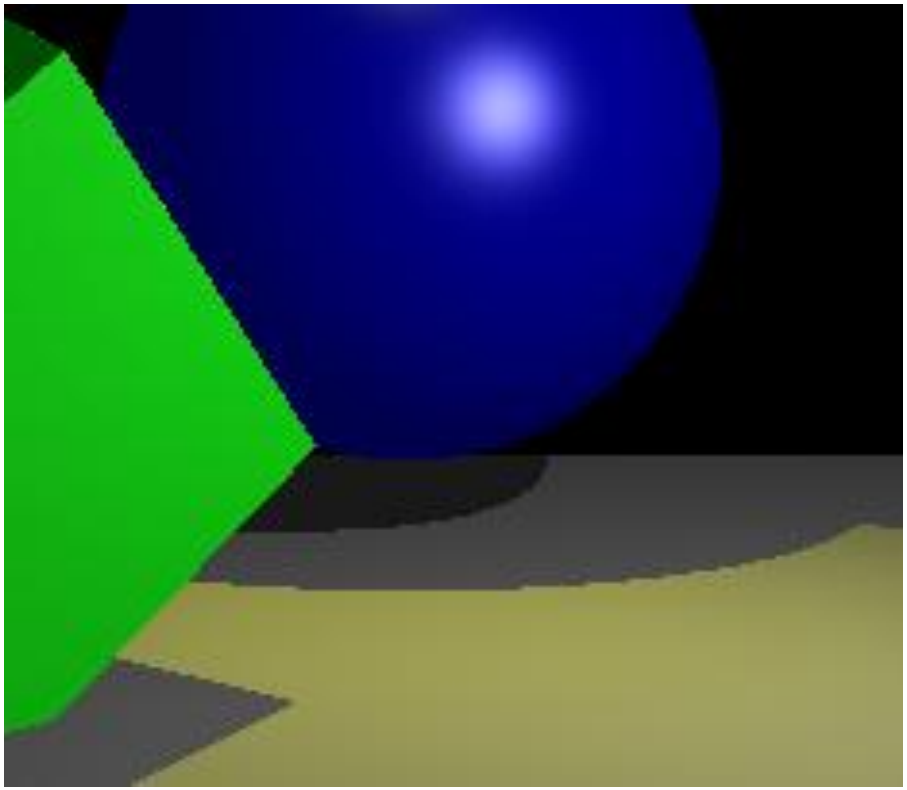
4.6 Antialiasing

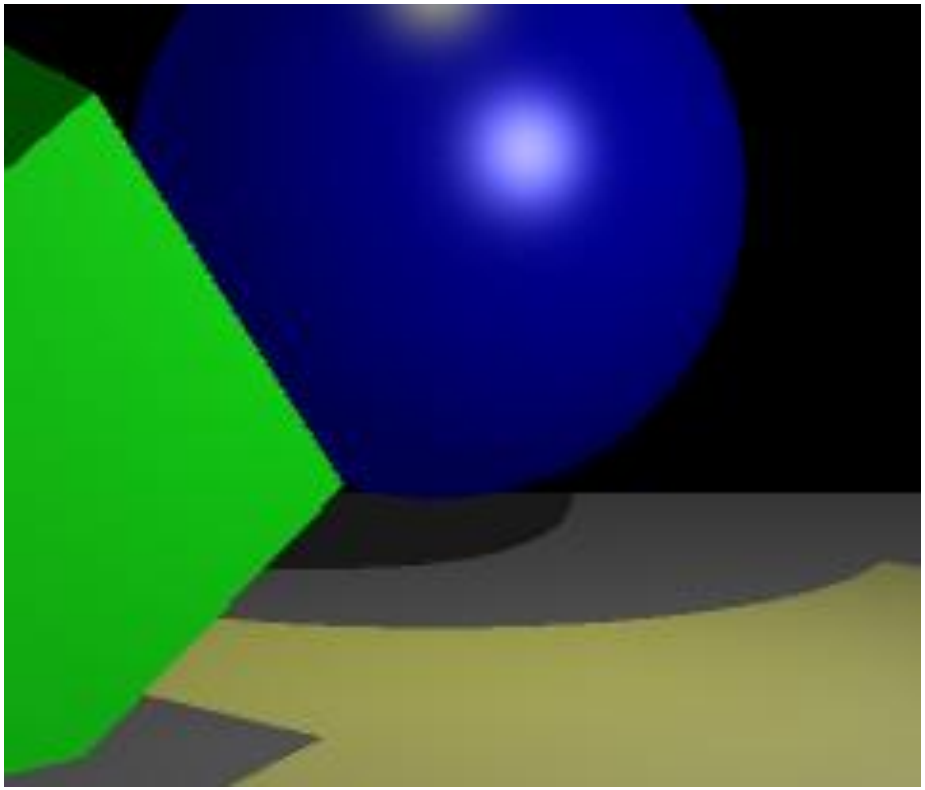
- El trazado de rayos sufre de “aliasing”
- Una forma de reducción del aliasing es el sobremuestreo
  - Sobremuestreo uniforme
    - Trazar varios rayos por pixel
    - Promediar el color
  - Adaptativo y estocástico
- No repetir rayos ya calculados



4.6 Escena test

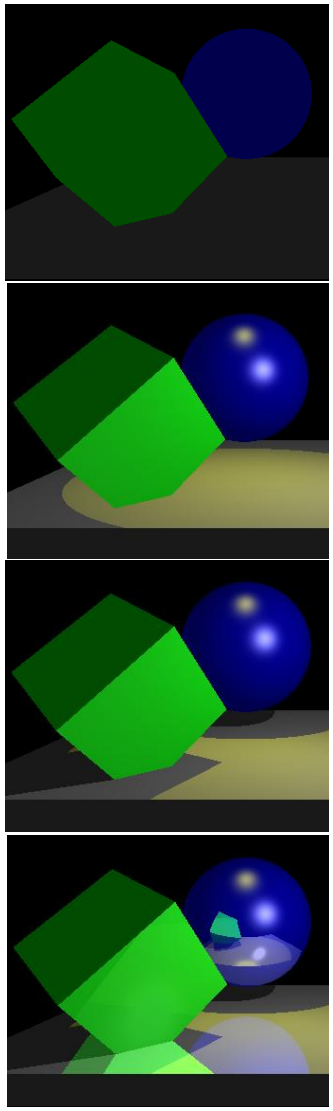
- En Trazador
  - Al pulsar ‘A’ activar sobremuestreo
  - Al pulsar ‘a’ desactivar sobremuestreo





Valoración

- Visibilidad (0.5 puntos)
  - al menos 1 esfera y dos cajas con transformación
- Iluminación (0.5 puntos)
  - al menos ambiental y dos luces, una de ellas focal
- Sombras (0.5 puntos)
  - al menos dos luces
- Reflexión (0.5 puntos)
  - al menos 1 reflexión
- Antialiasing (0.5 puntos)
  - al menos sobremuestreo 5RxPixel



Entregar Trazador.exe y todos los fuentes