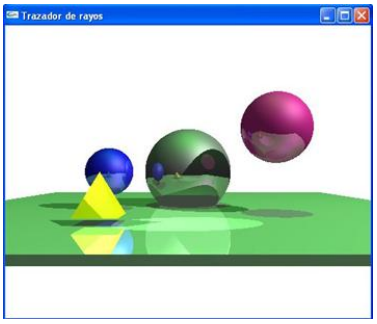
Gráficos por Computador. Prácticas de Laboratorio

Práctica 4. Trazado de rayos



El objetivo de la práctica es implementar un trazador de rayos sencillo. Dicho trazador debe trabajar como mínimo con los objetos gráficos *Esfera* y *Caja*. Se debe crear una escena según criterio propio con instancias de las clases implementadas. La construcción del trazador se hará progresivamente y apoyándose en la biblioteca de clases básicas *Algebra* y *Primitiva*.

4.1 Construcción de objetos

Las clases que se deben implementar seguirán la jerarquía:

Objeto
- **Esfera**
- **…**
- **Poliedro**
  - **Caja**
  - **…**

Es recomendable apoyarse en primitivas tipo polígono para construir poliedros.

Las clases Objeto y Esfera se proporcionan ya implementadas.

El alumno deberá definir e implementar adecuadamente estas clases según su criterio siempre que se cumplan los requisitos siguientes:

1. Todo Objeto debe responder a los mensajes:

```
int rayIntersection(Punto p, Vector v, float &t) const;
```

que devuelve en **t** el valor del parámetro de la recta para el que se produce la intersección $P(t)=p+v*t$ entre la semirrecta que parte del punto **p** y sigue la dirección del vector **v** y el objeto. La función devolverá 0, si el rayo no interseca con el objeto o es tangente a él, o 1 en caso de existir la intersección. Debe tenerse en cuenta que la intersección sólo será válida si t>0, condición necesaria para que esté por delante del inicio del rayo. Opcionalmente, el alumno puede contemplar el caso en que la intersección se produzca en una cara interior del objeto, devolviendo un valor diferente (por ejemplo -1). Este caso se produce cuando el inicio del rayo está dentro del objeto.

Para el desarrollo de este método en poliedros convexos –como la Caja- se recomienda **seguir el método de Haines** visto en teoría donde se analizan las t's "entrantes" y "salientes". Este método se puede implementar en el nivel de Poliedro siendo válido para cualquier clase derivada.

```
Vector normal(Punto p) const;
```

que devuelve el vector perpendicular unitario a la superficie del objeto en el punto **p** y dirigido hacia el exterior.
En el caso que se trate de un poliedro deberá identificarse la cara a la que pertenece **p** y devolver la normal a esa cara.

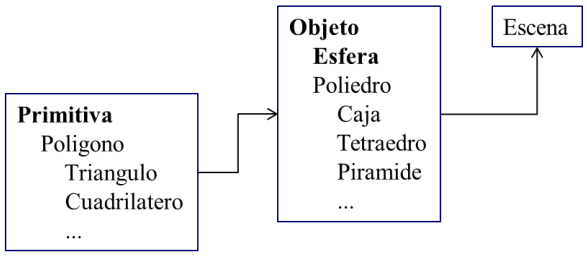
Además de las clases geométricas anteriores, se deberá implementar la clase **Escena**, que servirá como contenedor de todos los objetos de una escena. Además, deberá ofrecer la opción de insertar nuevos objetos. El almacén de objetos debe ser una estructura de datos preferiblemente dinámica.

2. La clase Escena debe responder obligatoriamente al mensaje

```
Color rayTrace (Punto p, Vector v) const;
```

donde se devuelve el color del primer objeto de la escena con el que interseca el rayo que parte de **p** y sigue la dirección **v**. El color del objeto será, en esta primera parte de la práctica, el color difuso.

Un ejemplo de la jerarquía a desarrollar es el siguiente:



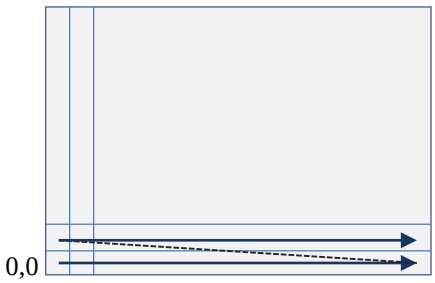
Ejemplo de estructura de clases

4.2 Visibilidad por trazado de visuales

Se quiere representar una escena resolviendo el problema de la visibilidad mediante el método de trazado de visuales (*ray casting*) por los píxeles del raster detectando intersecciones. Este es el algoritmo básico de resolución del problema de la visibilidad en el espacio de la imagen. Para ello se hará uso de las clases del apartado 4.1 debiendo generarse un código ejecutable *Trazador.exe* que visualice una escena.

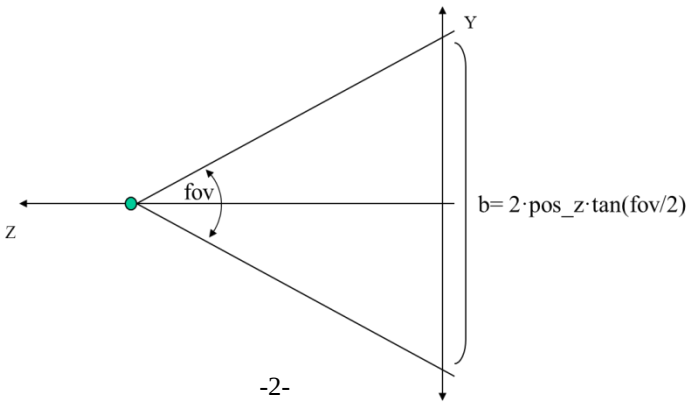
Para construir Trazador se seguirán los pasos siguientes:

1. Construir la escena
   - Se proporciona ya construida una escena ejemplo compuesta de cinco esferas
   - Además se debe construir una escena propia que incluya al menos una Caja
2. Construir la estructura de datos «raster»
   - El *raster* es un puntero a un bloque de memoria de *ancho* x *alto* valores RGB. *ancho* x *alto* son las dimensiones en píxeles del marco de dibujo y se fijan en la función *reshape()*. Cada valor R,G y B se codifica como *unsigned char*, es decir, un valor entre 0 y 255. Por tanto, la estructura *raster* se construye así:

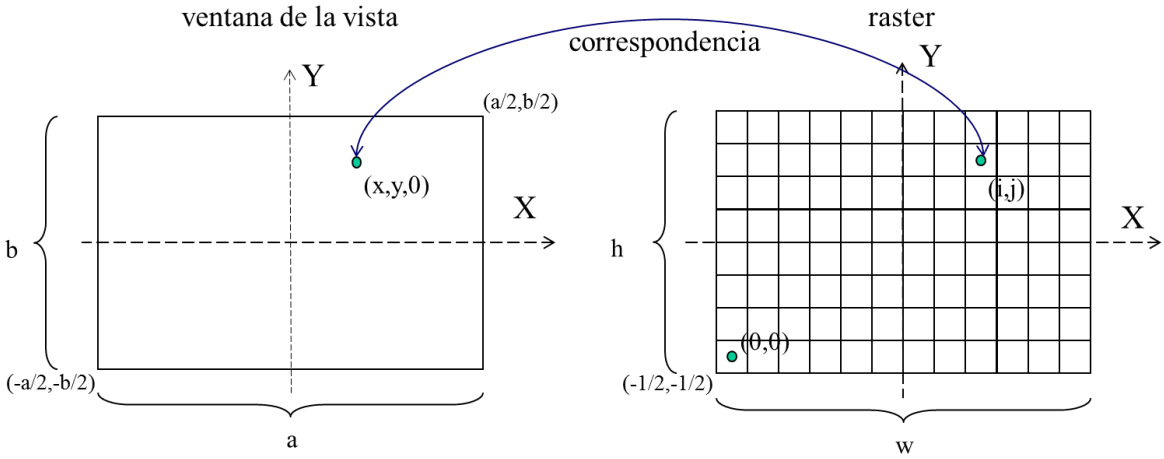



```
raster=(unsigned char *)malloc(sizeof(unsigned char)*3*ancho*alto);
```

   - Los valores almacenados son, consecutivamente, rgbrgbrgb... correspondiendo los tres primeros al píxel (0,0), los siguientes tres al (1,0) y así sucesivamente -por filas-. El píxel (0,0) es el inferior izquierdo del marco de dibujo.
3. Definida la vista, construir visuales (rayos primarios) que pasan por los centros de los píxeles
   - La vista se define situando al observador en (0,0,pos_z), el plano del raster el plano XY y un campo de visión vertical (fov) . La relación de aspecto de la ventana de la vista será la misma que la del marco de dibujo *ancho/alto.*



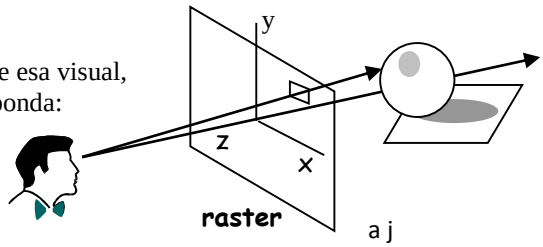
- Cada visual se forma haciendo pasar un rayo por el centro de cada píxel desde la posición del observador. Es necesario calcular la coordenada *(x,y)* de cada píxel *(i,j)* en el sistema de referencia de la escena para poder construir el rayo como *(Punto(0,0,pos_z), Vector(x,y,-pos_z))*. Para ello hay que utilizar la correspondencia de rectángulos en 2D:



4. Evaluar en la escena el primer objeto visto desde cada visual
   - Cada rayo primario es enviado a la escena para que descubra el primer objeto atravesado por él. El método Escena::rayTrace() comprueba la intersección del rayo con cada uno de los objetos de la escena y selecciona el más cercano al inicio del rayo. Con esta información es posible calcular el color en el punto de la superficie del objeto donde se produce la intersección con el rayo. Este cálculo puede ser más o menos laborioso dependiendo del modelo de iluminación elegido.
   - En este apartado, el cálculo del color se hace simplemente consultando el atributo *colDifuso* del objeto:

```
para cada objeto
        calcular la intersección con el rayo
        si está más cerca seleccionar este objeto
si hay algún objeto seleccionado devolver su color difuso
si no, devolver el color del fondo
```

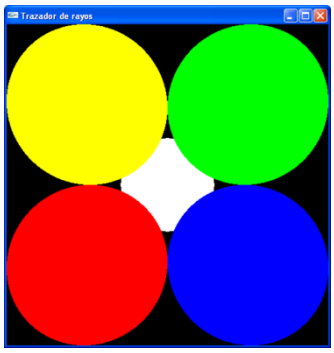
5. Poner cada píxel al color difuso del objeto obtenido antes
   - Una vez trazado el rayo y conocido el color visto desde esa visual, se rellena el RGB en la posición del *raster* que corresponda:




```
posraster=raster
para cada pixel (i,j) recorriendo por filas
        calcular la y del pixel que corresponde
        calcular la x del pixel que corresponde a i
        trazar el rayo (observador, visual) en la escena obteniendo el color
        *posraster= color.r()*255
        incrementar posraster
        *posraster= color.g()*255
        incrementar posraster
        *posraster= color.b()*255
        incrementar posraster
```

6. Copiar la estructura «raster» a la pantalla
   - En la función *display()* se copia la estructura *raster* a la pantalla utilizando las dos funciones de OpenGl siempre que la imagen haya cambiado debido a un nuevo trazado de visuales

```
glRasterPos2i(0,0);
glDrawPixels(ancho, alto, GL_RGB, GL_UNSIGNED_BYTE, raster);
```



Si el problema se resuelve correctamente, cada primitiva aparecerá en pantalla con aspecto plano y de color uniforme. Las primitivas se taparán unas a otras según su proximidad al observador. En el directorio de prácticas se suministra el ejecutable a obtener con una escena test y el código de construcción de tal escena en *Trazador.cpp* que se suministra incompleto.

Una vez se ha obtenido la escena de la figura, se puede proceder a cambiar la escena por una propia que contenga poliedros (cajas al menos).

4.3 Iluminación local

En este apartado se aplicará el **modelo de iluminación de Phong** usando el vector intermedio **H**. Para ello se seguirán estas indicaciones:

- Asignar valores a los atributos reflexivos de los objetos con el método Objeto::setColor(). Estos atributos se deben usar en el modelo de Phong para calcular el color en un punto de la superficie del objeto.
- Crear una jerarquía de clases para fuentes luminosas con los atributos necesarios para caracterizar cada una de ellas.
  FuenteLuminosa
  Ambiental
  Puntual
  Direccional
  Focalizada
- Las fuentes luminosas responderán, como mínimo a los mensajes siguientes:
  - **Color intensity(Punto p)** que devuelve la intensidad -color RGB- de la fuente emitida en dirección al punto **p**. Nótese que sólo en las fuentes focalizadas la intensidad no es uniforme y depende de la dirección de emisión.
  - **Vector L (Punto p)** que devuelve el vector unitario de iluminación sobre el punto **p** de la superficie. En el caso de luz ambiente el vector devuelto será el (0,0,0). En el caso de luces direccionales, **L** no depende de **p**.
- Deberá añadirse a la clase **Escena** un nuevo dato miembro que mantenga las fuentes luminosas presentes en la escena.
- Se modificará el cálculo del color en el punto de la superficie objetivo dentro del método Escena::rayTrace(). Ahora hay que aplicar el modelo de iluminación local para cada luz encendida:
  seleccionar objeto más cercano y el punto intersección
  acumular la luz ambiental al color
  para cada luz encendida en el hemisferio visible del punto intersección
  acumular reflexión difusa al color
  acumular reflexión especular al color

Como mínimo debe modificarse *Trazador.cpp* para que pulsando 'L' se enciendan todas las luces y pulsando 'l' se apaguen, excepto la ambiental que siempre debe estar presente. Hay que elegir las posiciones e intensidades de las fuentes de tal manera que sea evidente en la imagen el tipo de luz utilizada.

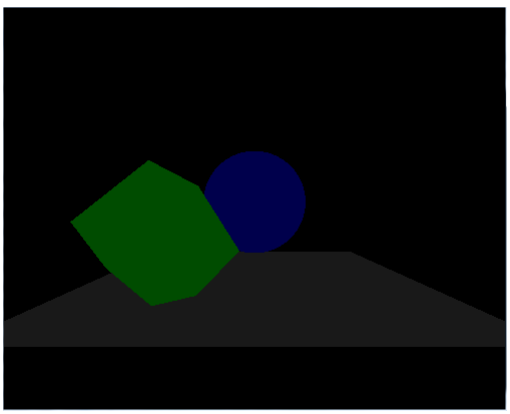


Imagen 1. Sólo iluminación ambiente

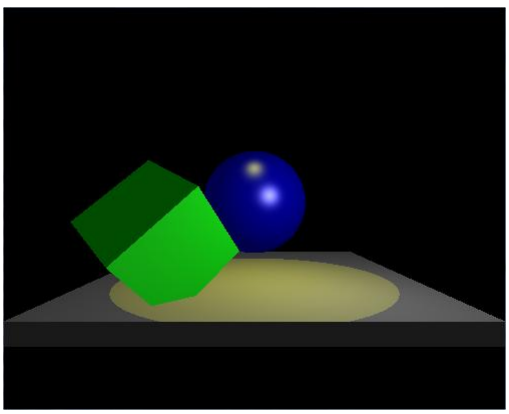


Imagen 2. Luces puntual y focal

4.4 Sombras arrojadas

El siguiente paso es la generación de sombras arrojadas de unos objetos sobre otros. Para ello debe modificarse el cálculo del modelo de iluminación local teniendo en cuenta la visibilidad de las diferentes luces desde el punto que se observa.

Los rayos de sombra (secundarios) se trazan desde el punto hacia la fuente de luz recogiendo las intersecciones con todos los objetos de la escena. Si la intersección con algún objeto sucede antes de alcanzar la fuente de luz, la luz no es visible y no se considera en cálculo de la iluminación del punto.

Es importante atender al problema de la **autosombra** que puede llevar a bloquear una luz por el cálculo prematuro de la intersección del rayo con el propio objeto cuando, por errores numéricos, el comienzo del rayo se sitúa ligeramente por debajo de la propia superficie.

Se debe modificar *Trazador.cpp* para que responda a la pulsación de 'S' activando las sombras y 's' desactivándolas.

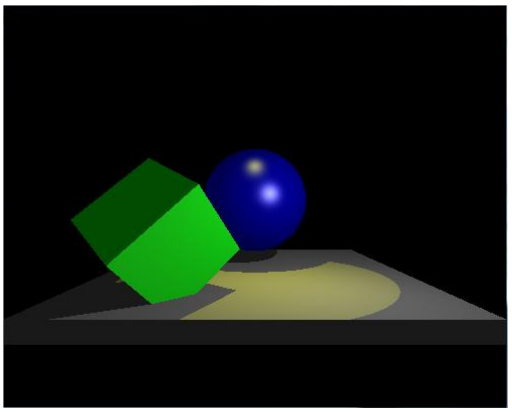


Imagen 3. Sombras arrojadas

4.5 Reflexiones

Las reflexiones se producen ampliando el modelo de iluminación local al aporte de la intensidad que viene del rayo reflejado. Este modelo de iluminación se ha estudiado en clase (Whitted).

Como mínimo se debe considerar 1 rebote (árbol de rayos de profundidad 2). En este caso el rayo reflejado se calcula a partir de la visual o rayo primario y la normal en el punto observado. La intensidad que viene desde el rayo reflejado se calcula de la misma manera que en antes. Este proceso puede es recursivo según se ha estudiado en teoría.

Se debe modificar *Trazador.cpp* para que atienda a las teclas 'R', para que se activen los rayos secundarios de reflexión, y 'r' que los desactiva.

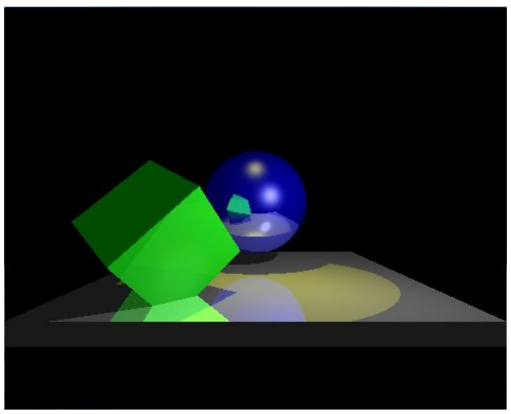


Imagen 4. Reflexiones (1 nivel)

4.6 Antialiasing

Las técnicas de sobremuestreo permiten reducir el aliasing en la imagen. También pueden utilizarse para conseguir efectos como penumbras o difuminado de los reflejos.

Para este apartado se debe implementar como mínimo sobremuestreo uniforme para rayos primarios del tipo de 5 rayos por pixel. Para ello se modificara el trazador de visuales para que genere rayos por los vértices de los píxeles. El color será el promedio del obtenido por los cuatro vértices y el centro del pixel. Por razones de eficiencia no se debe repetir el trazado de la misma visual –las que pasan por los vértices-.

El programa activará el sobremuestreo cuando se pulse 'A' y lo desactivará con 'a'.

Valoración de la práctica

La práctica se puntúa con un máximo de 2.5 puntos sobre el trazado de rayos de una escena propia del alumno que contenga, al menos, esferas y cajas. El programa *Trazador* debe atender a la interacción descrita en cada apartado.
Puntuación:

- Visibilidad (apartado 4.2) –al menos 1 esfera y dos cajas con transformación-: 0.5 puntos
- Iluminación (apartado 4.3) –al menos ambiental y dos luces, una de ellas focal-: 0.5 puntos
- Sombras (apartado 4.4) –al menos dos luces-: 0.5 puntos
- Reflexión (apartado 4.5) –al menos 1 reflexión-: 0.5 puntos
- Antialiasing (apartado 4.6) –al menos sobremuestreo 5RxPixel-: 0.5 puntos

Se entrega Trazador.exe en modo Release y todo el código fuente.

Anexo. Escena_test

Se ha diseñado una escena compuesta por dos cajas y una esfera con el fin de comprobar de manera incremental el desarrollo del trazador de rayos. La escena está iluminada por dos luces, una puntual y otra focalizada. Se ha situado la cámara en el eje z mirando hacia el origen de coordenadas.

El modelo de iluminación usado es el de Whitted estudiado en teoría (sin trasmisión)

$$I = k_a C_d I_a + \sum_i I_i (k_d C_d (N \cdot L) + k_s (N \cdot H)^m) + k_s I_r$$

donde k_a,k_d,k_s son los factores de reflexión ambiental, difuso y especular respectivamente. I_a es la iluminación ambiente, I_i es la intensidad de la luz i que llega al punto e I_r es la intensidad del rayo reflejado. C_d es el color del objeto o color difuso. N,L,H son los vectores unitarios estudiados en iluminación y m es el exponente de concentración de brillo. Las intensidades deben entenderse como colores. Obsérvese que no se ha utilizado el color especular del objeto (o se ha considerado blanco para sólo la fuente influya en el color del brillo).

Puede usarse la esta escena para comprobar el correcto de cada apartado del trazador construido. Se suministra el ejecutable que responde a la interacción planteada (teclas Ll,Ss,Rr,Aa).

Descripción de la escena

```
Camara:
  pov  0,0,3
  look 0,0,-1
  up   0,1,0
  fovy 65°
Raster:
  anchoinicial 500
  altoinicial  400
Luces:
  Ambiental
    color 1,1,1
  Puntual
    posicion 1.0,0.5,0.0
    color 1,1,1
  Focal (Warn)
    posicion 0,2,-1
    color 1,1,0
    direccion 0,-1,0
    exp.concentracion 3
    angulomax 30°
```

```
Objetos:
  Esfera
      radio  0.8
      centro 0.0,0.1,-2.0
      color  1,0,0
      ka       0.3
      kd     0.3
      ks     0.7
      exp.brillo 40
  Caja suelo (parte del cubo
unitario)
      escala   2.0,0.1,2.0
      posición 0.0,-1.0,-1.5
      color    1,1,1
      ka       0.1
      kd       0.3
      ks       0.9
      exp.brillo 100
  Caja cubo (parte del cubo
unitario)
      escala 0.5,0.5,0.5
      giro   60° sobre 1.0,1.0,1.0
      posición -1.0,-0.3,-0.3
      color  0,1,0
      ka     0.3
      kd     0.8
      ks     0.1
      exp.brillo 1
```