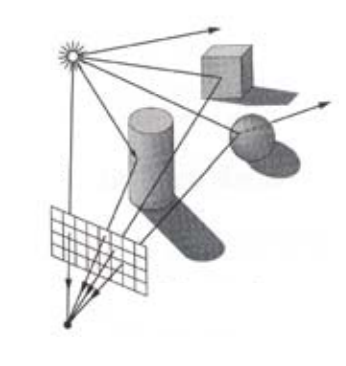
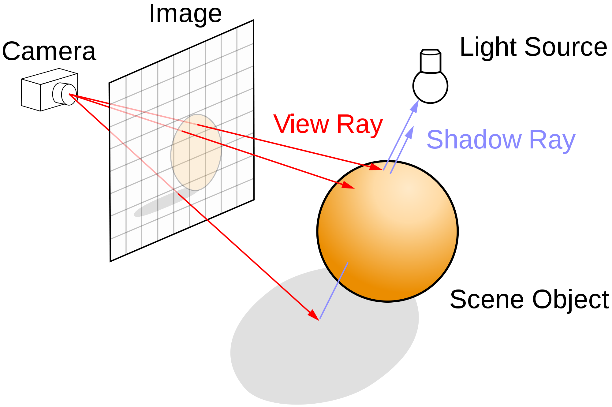
**Introducción**

En este proyecto se ha planteado el uso de nuestros conocimientos aprendidos en la asignatura de informática gráfica sobre la técnica de trazado de rayos para implementar un trazador de rayos propio que sea capaz de generar imágenes mínimamente complejas y físicamente correctas respecto a sombras, iluminación de la escena, comportamiento de los rayos de luz, etc.

**Descripción:**

**El método del Trazado de Rayos inverso**

El Trazado de Rayos Inverso es una de las técnicas más populares en el mundo de los gráficos por computador. Propuesto inicialmente por Turner Whitted en 1980, se basa en la idea de lanzar rayos desde el sistema de visualización (cámara/ojo) hacia la escena y computar las propiedades lumínicas, en lugar de lanzarlos desde las fuentes de luz a la escena y después hasta el sistema de visualización como propone el Trazado de Rayos Directo. En las dos imágenes siguientes se aprecia la diferencia entre las dos aproximaciones:

Trazado Directo Trazado Inverso

**El algoritmo desarrollado**

El algoritmo que se ha desarrollado, condensa la idea del Trazado Inverso. Para ello, se lanza un rayo desde el ojo hacia la escena pasando por cada píxel de la pantalla de visualización (con matices si hay antialiasing), y si el rayo colisiona con algún objeto de la escena pregunta por las propiedades del material y el color del mismo. Realizando algunos cálculos, al final se obtiene el color de cada píxel, que es lo que compone la escena final que se genera.

El algoritmo tiene en cuenta la componente ambiental, difusa y especular de cada objeto, así como sus propiedades reflectantes y refractarias. Para estas dos últimas es necesaria la llamada a una función recursiva que calcula el color final. También se tienen en cuenta las sombras generadas por los objetos.

Las escenas que el algoritmo es capaz de generar está compuestas por planos, triángulos y esferas.

También se realiza antialiasing por supermuestreo para mejorar los contornos de los objetos al evitar “dientes de sierra”.

**Arquitectura**

En esta sección se detallará la arquitectura del sistema desarrollado, empleando para ello algunos diagramas UML y descripciones textuales.

**Diagramas UML**

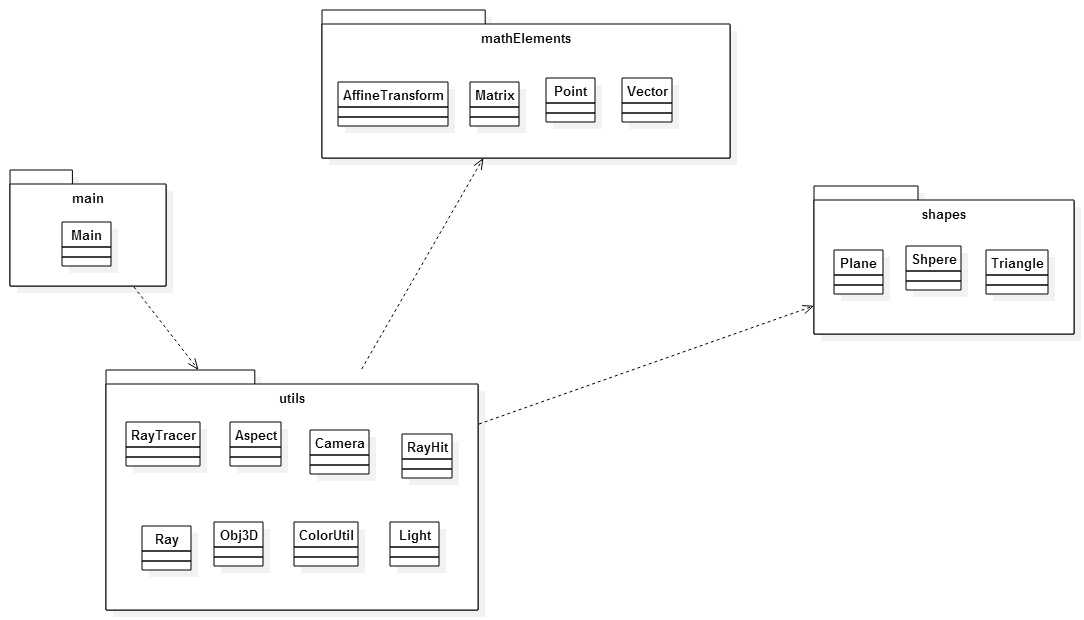


Diagrama de paquetes

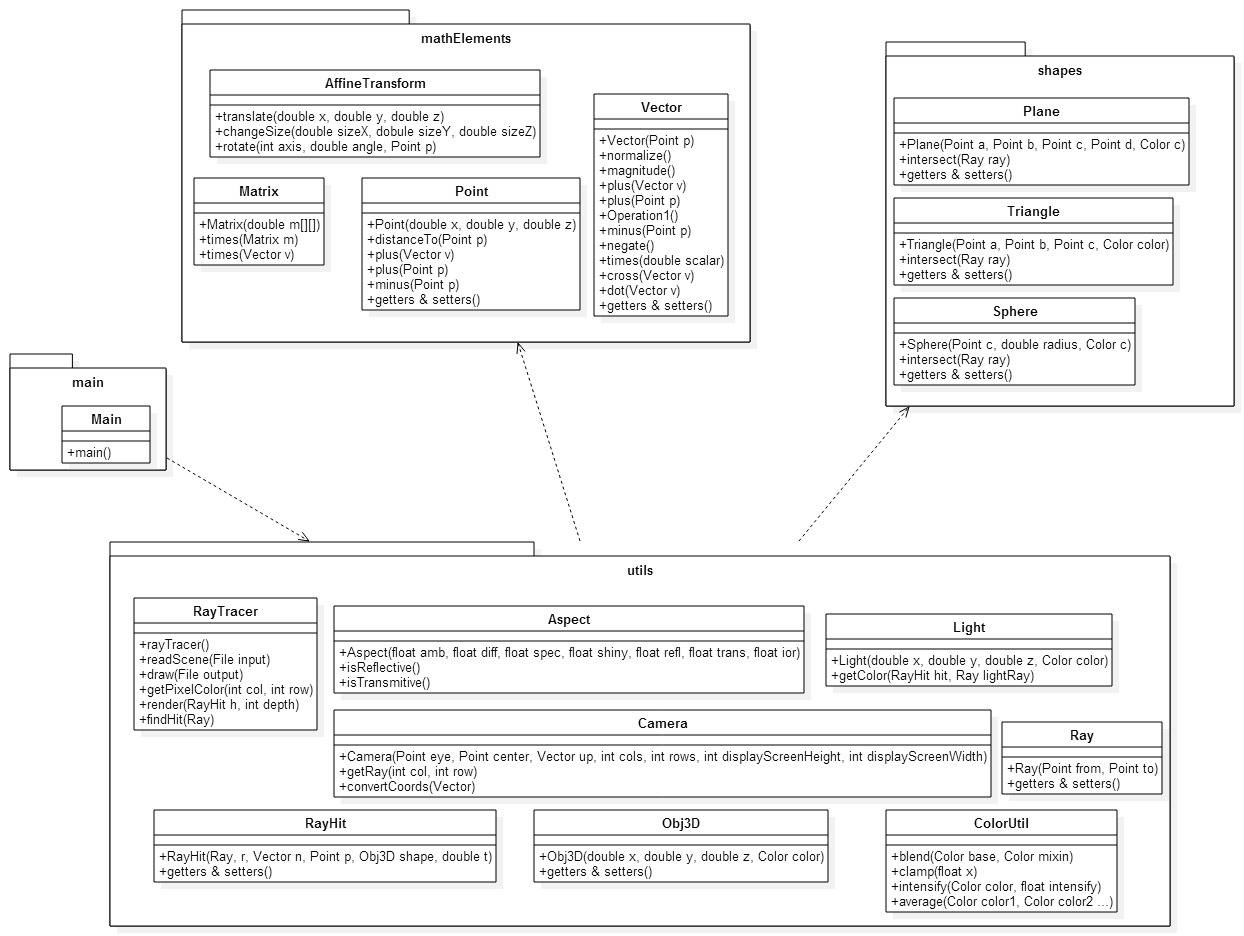


Diagrama de paquetes extendido

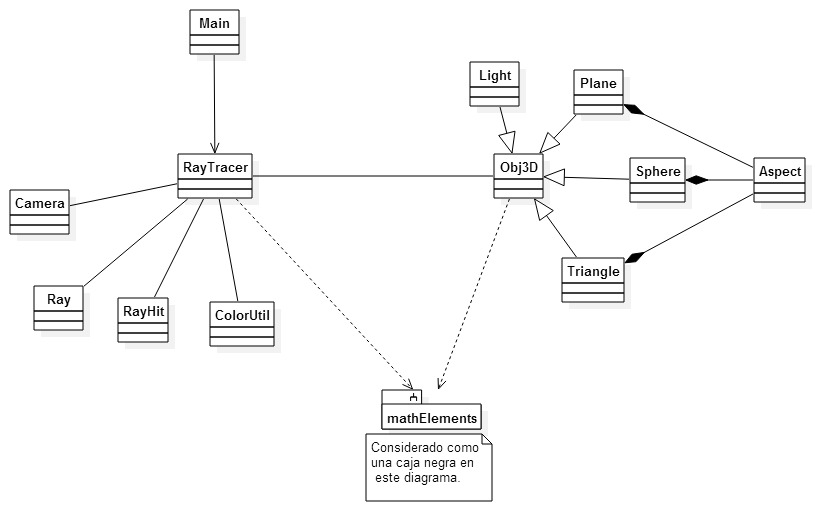


Diagrama de clases

**Breve descripción de cada clase**

Main.- Contiene el método principal del programa. Se encarga de crear un objeto RayTracer que será el más importante durante el resto de la ejecución.

RayTracer.- Se encarga de leer el fichero de entrada para generar el mundo 3D, además de colorear del color adecuado cada pixel de la imagen que se genera al final. Gestiona también el antialiasing y la recursividad para rayos reflejados y transmitidos.

Camera.- Se crea en un momento muy temprano de la ejecución y es la encargada de fijar la cámara en el mundo, los sistemas coordenados y la pantalla de visualización.

Ray.- Clase que encapsula un rayo, que puede ser de diferentes tipos (de la cámara al mundo, rayo de sombra…).

RayHit.- Clase que encapsula el lugar del mundo donde colisiona un rayo.

ColorUtil.- Clase que agrupa algunas utilidades para operar con colores, como por ejemplo mezclarlos, calcular la media de varios colores…

Obj3D.- Clase padre que representa los objetos que se alojan en el mundo 3D. Integra algunas propiedades como posición en el mundo y color.

Light.- Clase que modela el comportamiento de las luces que iluminan la escena. Tiene propiedades propias como por ejemplo intensidad RGB. Hereda de Obj3D.

Sphere.- Clase que modela una esfera 3D. Tiene propiedades propias como por ejemplo su radio. Hereda de Obj3D.

Plane.- Clase que modela un Plano finito delimitado por cuatro puntos. Tiene propiedades propias como por ejemplo sus límites en el espacio. Hereda de Obj3D.

Triangle.- Clase que modela un Triangulo delimitado por tres puntos. Tiene propiedades propias como por ejemplo sus límites en el espacio. Hereda de Obj3D.

Aspect.- Clase que aglutina las propiedades de los materiales de los que están hechas las figuras que aparecen en la escena, como por ejemplo coeficiente de refracción o brillo especular.

AffineTransform.- Clase que define las transformaciones 3D que se realizan para ubicar cada elemento en su lugar dentro de la escena.

Point.- Clase que modela un punto en el espacio 3D.

Vector.- Clase que modela un vector en el espacio 3D.

Matrix.- Clase que modela una matriz. Se emplea por ejemplo para operaciones de cambio de base.

**Funcionamiento**

**Como se lanzan los rayos**

En este trazador los rayos se lanzan desde el punto del mundo donde se encuentra la cámara hasta el centro de cada píxel (a no ser que la opción antialiasing esté activa).

**Como se intersecta**

**Calculo del rayo reflejado**

**Calculo del rayo transmitido**

**Modelo de iluminación**

**Antialiasing**

Se ha implementado con el método de lanzamiento de más de un rayo por cada píxel (número de rayos indicado en el fichero de configuración de la escena). El lanzamiento de dichos rayos se hace de forma aleatoria, con esto se consigue no crear una estructura fija de renderización, por lo que el ojo humano nota menos los saltos entre píxeles.

**Pseudocódigo de la clase principal del trazador**

**Rendimiento en nuestros equipos y ejemplos**

**Documentación de las clases del código y sus comentarios**

**Conclusión**