Computación Gráfica
Trabajo práctico Especial 1 - Ray Caster
Grupo 2

Guillermo Campelo
Juan Ignacion Goñi
Juan Tenaillon
Santiago Vazquez

Resumen

Se diseñó y desarrolló un motor de Ray Casting que determine intersecciones entre rayos y objetos permitiendo representar en pantalla un conjunto de escenasb.

Palabras Clave: *Ray Casting, escena, rayos, objetos..*

1. Introducción

En este informe, se explicará el diseño y el desarrollo del motor de Ray Casting, explicando cada una de sus componentes y que nos motivo a desarrollarlas de la manera en que lo hicimos.

En la segunda sección nos enfocaremos en el motor de Ray Casting y como se utiliza, luego en la tercera sección nos enfocaremos en la escena y los objetos que la componen, como sean las esferas, los triangulos y los cuadrilateros. En la cuarta sección nos enfocaremos en los resultados obtenidos y por último, en la quinta sección, presentamos nuestras conclusiones.

2. Ray Caster

2.1. Descripción

Ray Casting es una técnica que utiliza la intersección de rayos con una superficie para resolver un conjunto de problemas en el campo de la computación gráfica. Representa una solución fácil al problema de visibilidad en una escena.

En este caso se modela una escena almancenada en memoria, para la cual se fija una posición y dirección para la cámara. Desde esta se disparan rayos a intervalos regulares y para cada uno se muestra el color correspondiente al objeto más cercano con el que colisiona.

2.2. La Cámara y la Escena

PARA QUE JUANI LA COMPLETE.

2.3. Modo de Uso

Para ejecutar el Ray Caster hay diferentes opciones. A continuación especificamos cuales son estas opciones y mostramos su modo de uso.

2.3.1. Nombre de Archivo de Entrada

Para poder utilizar el Ray Caster, es necesario especificar un nombre de archivo de entrada. Los posibles nombres para las escenas de entradas son de la forma $*.sc$ y para utilizar el ray caster con un archivo de este tipo hay que especificarlo de la siguiente manera:

```
-i [nombre de archivo]
```

Suponiendo un nombre de escena *scene*1.*sc*, una posible llamada al Ray Caster sería:

```
cg_tpe1 -i scene1.sc
```

Este campo es **obligatorio** para poder utilizar la aplicación.

2.3.2. Nombre de Archivo de Salida

Al ejecutar el Ray Caster, se producirÃą un archivo de salida. El usuario de la aplicaciÃşn puede elegir un nombre en particular para el mismo. Esto se hace utilizando la siguiente opción:

```
-o [nombre de archivo]
```

En caso de no especificar ningún nombre en particular para el mismo, el nombre del archivo de salida será el mismo que el archivo de entrada pero con extensión $*.bmp$ o $*.png$. Suponiendo que se requiere un nombre de archivo específico, el uso de la opción es la siguiente:

```
cg_tpe1 -i scene1.sc -o scene1.png
```

El uso de esta opción **no es obligatoria** para el uso de la aplicación.

2.3.3. Pixeles en la Imagen de Salida

Para indicar la cantidad de pixeles en la imagen de salida, se utiliza la opción indicada a continuación:

```
-size <alto>x<ancho>
```

En caso de no especificar ninguna cantidad en particular, se utilizará el tamaño por defecto de 640x480 pixeles. Un ejemplo de su uso sería:

```
cg_tpe1 -i scene1.sc -size 800x600
```

Este campo es **obligatorio** para poder utilizar la aplicación.

2.3.4. Campo de Visión

Para indicar el ángulo de apertura horizontal en grados, se va a utilizar:

```
-fov x
```

donde x es un número en grados. En caso de no especificarse ningún ángulo, el valor por defecto es de 60Âř. Un ejemplo de su uso sería:

```
cg_tpe1 -i scene1.sc -fov 80
```

Este campo es **obligatorio** para poder utilizar la aplicación.

2.3.5. Asignación de Color

Para indicar el modo de asignación de color de las primitivas en la escena, hay que utilizar esta opción:

```
-cm [random | ordered]
```

Donde *random* es el valor por defecto e indica una asignación de colores de forma aleatoria y *ordered*, donde la asignación de color dependerá del orden en que fueron descubiertas las primitivas. El orden de asignación es: *violeta*, *azul*, *verde*, *amarillo*, *naranja* y *rojo*. Un ejemplo de su uso sería:

```
cg_tpe1 -i scene1.sc -cm ordered
```

El uso de esta opción **no es obligatoria** para el uso de la aplicación.

2.3.6. Variación del Color

Para indicar el modo de variación del color de los elementos de la escena, hay que utilizar esta opción:

```
-cv [linear | log]
```

Donde *linear* es el valor por defecto e indica que la variación del color es proporcional a la distancia a la cual la primitiva se encuentra de la posición de la cámara y *log*, donde la variación del color es logarítmica. Un ejemplo de su uso sería:

```
cg_tpe1 -i scene1.sc -cv log
```

El uso de esta opción **no es obligatoria** para el uso de la aplicación.

2.3.7. Tiempo de Ejecución

Para determinar el tiempo que demoró la ejecución de la aplicación para renderizar la escena indicada, se utiliza:

```
-time
```

Un ejemplo de su uso sería:

```
cg_tpe1 -i scene1.sc -time
```

El uso de esta opción **no es obligatoria** para el uso de la aplicación.

3. Primitivas

Para modelar las escenas, se debieron modelar los distintos objetos presentes en la misma. En este caso, las primitivas seleccionadas fueron: *tringulo*, *cuadriltero* y *esfera*.

A continuación explicaremos, para cada una de las primitivas, cómo se decidió el diseÃśo de las mismas.

3.1. Rayo

3.1.1. Representación

3.2. Triángulo

3.2.1. Representación

3.2.2. Intersección

3.3. Cuadrilátero

3.3.1. Representación

Para representar un cuadrilatero, se pensó en la utilización de los cuatro puntos en el espacio. Se eligió esta representación debido a que, además de ser la más simple, también resulta útil al momento de calcular los planos en los que se encuentra para poder determinar la intersección de los rayos con el mismo.

En la siguiente sección, se explicará a grandes rasgos, cómo fue realizada la intersección del rayo con el cuadrilaátero.

3.3.2. Intersección

Para determinar la intersección entre un cuadrilátero y un rayo, se dividió el cálculo en dos partes.

La primera parte consistie en determinar si el rayo en cuestión, intersecta al plano que contiene a la primitiva. Para realizar esto, se calcula el *plano contenedor* y una vez obtenida la ecuación del tipo

$$\mathrm{Ax} + \mathrm{By} + \mathrm{Cz} + \mathrm{D} = 0$$

En esta ecuación, se obtiene el t para el cual el rayo intersecta al plano y en caso de no existir tal t, el rayo no intersecta a la primitiva, pero en caso de si existir, es necesario determinar si el punto de intersección entre el plano y el rayo pertenece a la primitiva.

Para la segunda parte del cálculo, se supuso que cualquier cuadrilátero son dos triángulos. Donde si el cuadrilátero tiene por puntos $p1$, $p2$, $p3$ y $p4$, el primer triángulo estaría compuesto por $p1$, $p2$ y $p3$ y el segundo estaría compuesto por $p1$, $p4$ y $p3$. Asumiendo esto, despues se utiliza una función que determina si un punto está dentro de un triángulo que fue explicada en la sección 3.2.

3.4. Esfera

3.4.1. Representación

3.4.2. Intersección

4. Resultados Obtenidos

El motor de Ray Casting tiene la capacidad de renderizar cualquier escena que estÃĺ cargada en memoria. En esta implementación, se decidió mostrar la capacidad del motor mediante la renderización de seis escenas. Tres de ellas, propuestas por la cátedra y las otras tres, propuestas por el grupo desarrollador. A continuación, enseñaremos cada una de ellas con una breve descripción.

4.1. Primera Escena: `scene1.sc`

Esta escena contiene una pirámide rectangular, cuyo lado de la base y su altura tiene una longitud de 5. La base de la misma está centrada en el origen, en su vértice superior tiene posicionada en perfecto equilibrio una esfera de radio 1 y a una distacia de 3, en cada uno de los vértices, tiene una esfera de radio 0.5. Ver **Figura 1**.

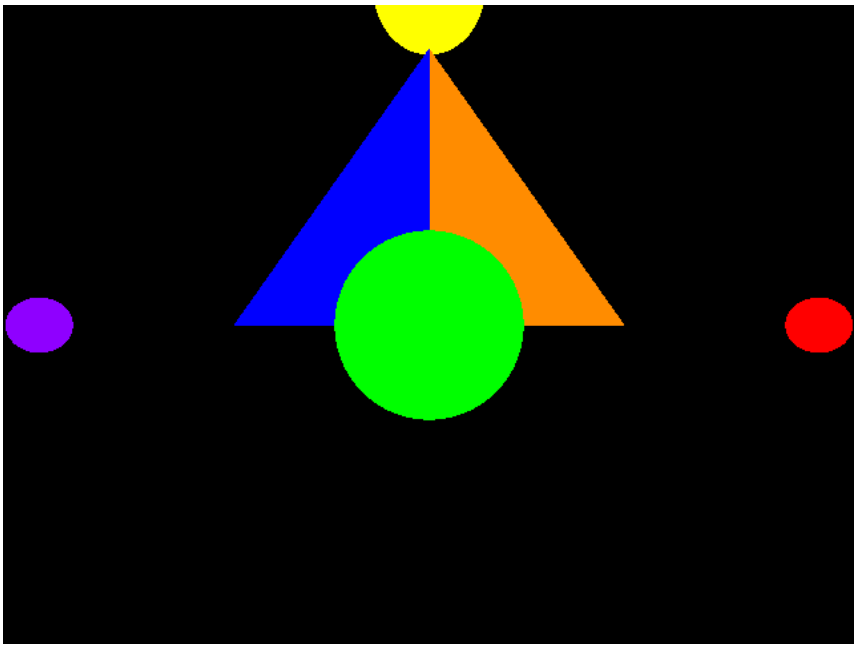

Figura 1: Escena número 1, triángulo y esferas.

4.2. Segunda Escena: `scene2.sc`

Esta escena contiene 64 esferas de radio 0.5 dispuestas en forma de cubo. Esto es, 4 esferas sobre el eje x, 4 sobre el eje y y 4 sobre el eje z formando un cubo de 64 esferas. Cada una de ellas debe estar a una distacia de 1.5 de la otra

y el centro de la figura debe estar en el centro del sistema de coordenadas. Ver **Figura 2**.

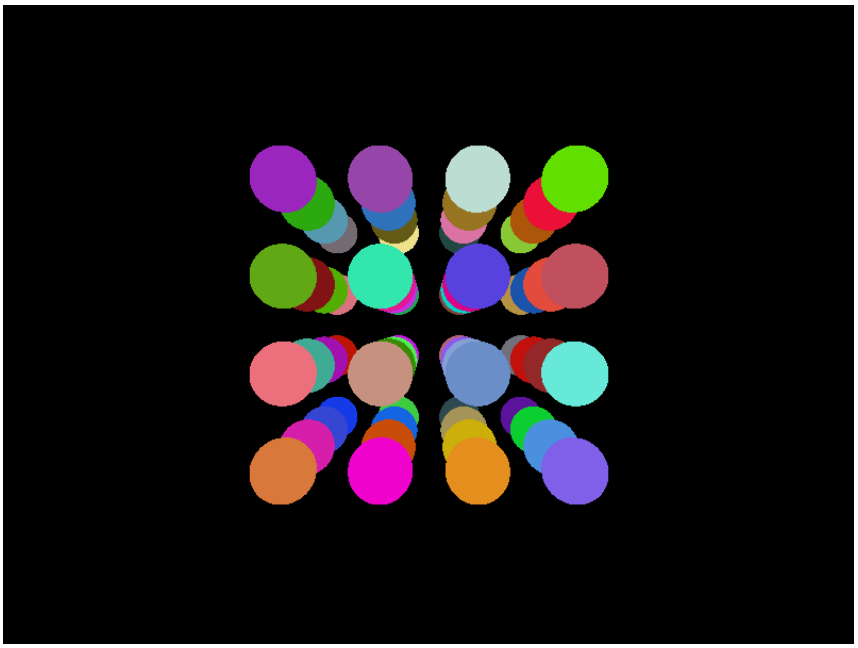

Figura 2: Escena número 2, esferas formando un cubo.

4.3. Tercera Escena: `scene3.sc`

Esta escena contiene, de forma intercalada, 3 cubos de lados 2 y dos esferas de radio 1 alineados de manera centrada sobre el eje x. Las bases de los cubos y las esferas están apoyadas sobre el eje $y = 0$ y la separación entre las figuras es de 0.5. Ver **Figura 3**.

4.4. Cuarta Escena: `scene4.sc`

Esta escena es similar a la tercera escena, pero la distancia entre las primitivas es 0. Ver **Figura 4**.

4.5. Quinta Escena: `scene5.sc`

Esta escena contiene 11 esferas alineadas a lo largo del eje x. Todas ellas a una distancia de 0.5 y con un radio de 0.5. Ver **Figura 5**.

4.6. Sexta Escena: `scene6.sc`

Esta escena contiene 20 triángulos que forman una estrella con centro en el centro del sistema de coordenadas. Ver **Figura 6**.

Figura 3: Escena número 3, cubos y esferas intercaladas.

5. Comentarios Finales

6. Bilbiografía

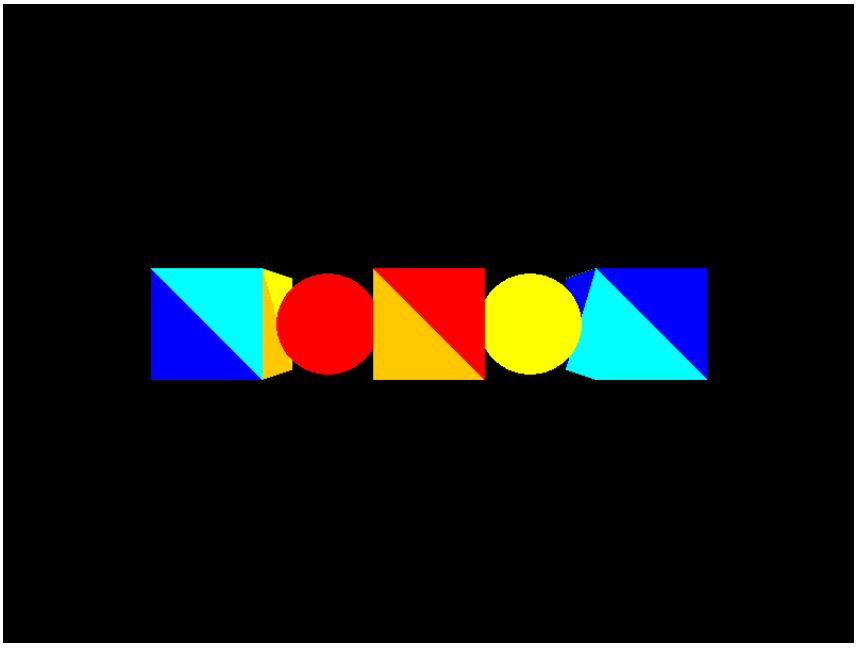

Figura 4: Escena número 4, cubos y esferas intercaladas.

Figura 5: Escena número 5, 11 esferas alineadas.

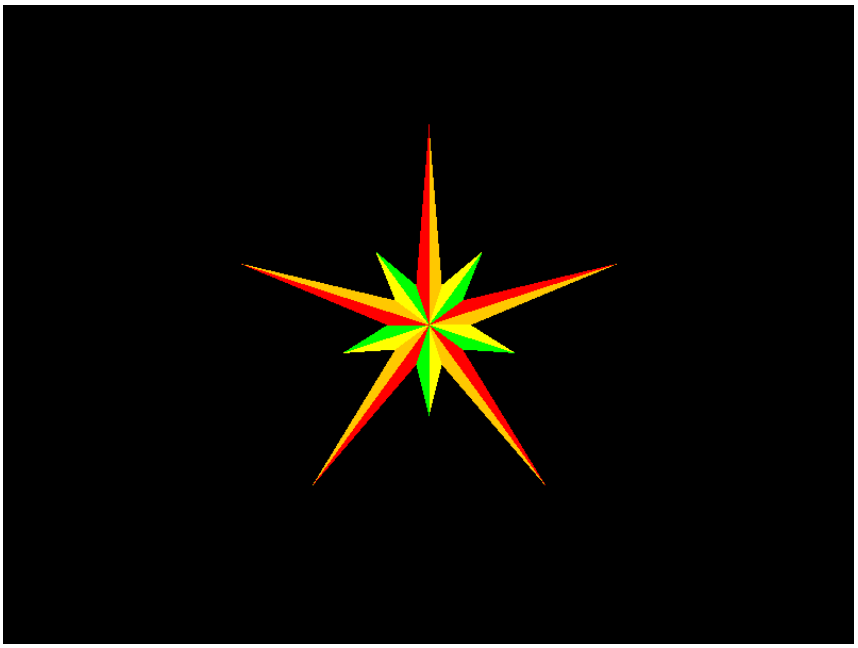

Figura 6: Escena número 6, 20 triángulos formando una estrella.