Computación Gráfica

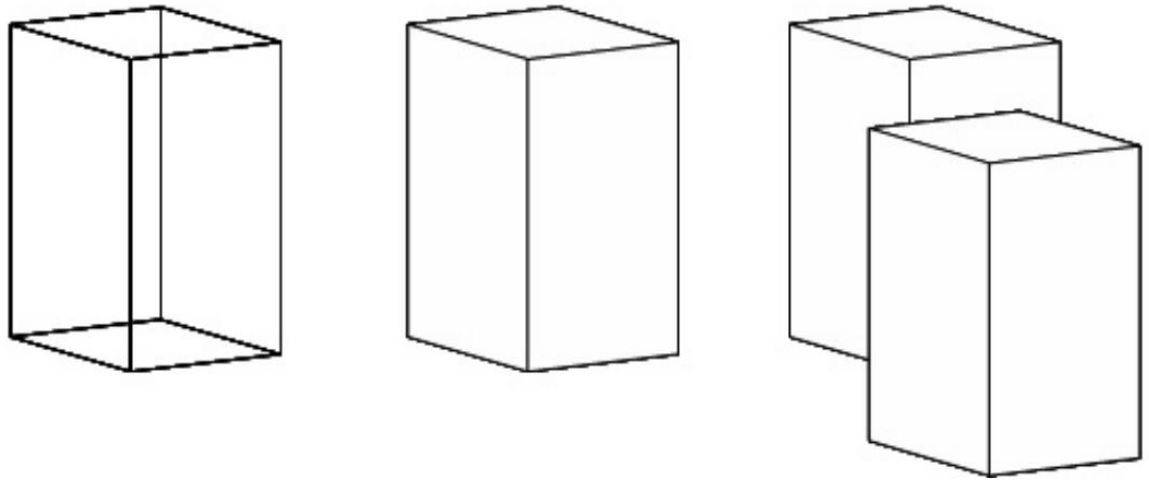
Ing. Gabriel Ávila, MSc.

Detección de visibilidad

Detección de visibilidad

En el renderizado de escenas, no todas las caras de los sólidos son visibles desde un punto de vista. ***Existen objetos que se deben omitir***.

Por ejemplo, las caras traseras de un objeto, o cuando hay objetos que están tapados por otros.



Detección de visibilidad

Se trata del conjunto de métodos y/o algoritmos, que permiten optimizar recursos para el renderizado.

Se basan en el principio de que solo es necesario renderizar aquellas superficies que son visibles a la cámara. La ubicación y método de proyección de la cámara ya se encuentran definidos.

Detección de visibilidad

El enfoque puede ser la ***determinación de superficies visibles*** o la ***eliminación de superficies ocultas***.

Los métodos se clasifican según el espacio en el que operan:

- **Métodos en el espacio de los objetos**: Se verifican las relaciones entre objetos o superficies en el espacio 3D.
- **Métodos en el espacio de la imagen**: Se verifica la visibilidad en el plan de proyección, pixel por pixel.

Detección de visibilidad

Los diferentes métodos para la detección de visibilidad dependen de:

- La complejidad de la escena.
- Tipos y estructuras de datos de los objetos.
- Hardware disponible.
- Requerimientos de la aplicación.

Además, dichos métodos pueden enfocarse en la exactitud o en la velocidad en la evaluación de las superficies.

Métodos de detección de visibilidad

- Buffer Z.
- Remoción de superficies – *Backface, frustum and occlusion culling.*
- Trazado de rayos – *Ray-Casting.*
- Líneas de barrido.
- Algoritmo del pintor.
- Octrees.
- Árboles BSP – *Binary Space Partition Tree.*
- Etc.

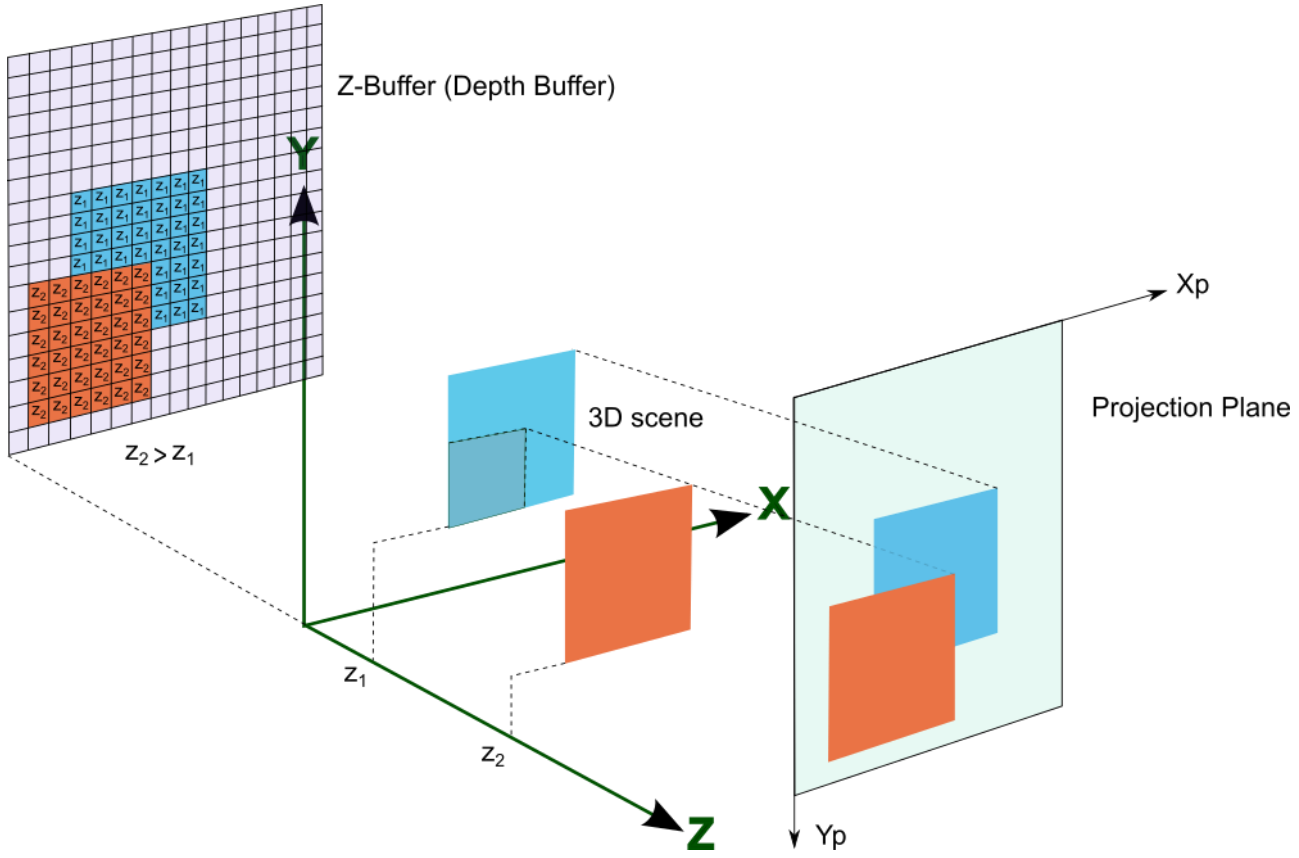
Buffer de profundidad: Buffer Z

El Z-Buffer permite gestionar las coordenadas de profundidad de diferentes elementos en una escena 3D. Normalmente es un arreglo 2D que almacena los valores de profundidad de los objetos más cercanos a la cámara.

Para obtener este arreglo, se comparan los valores de profundidad (valores en el eje z desde la cámara) para cada superficie en cada posición de pixel sobre el plano de proyección.

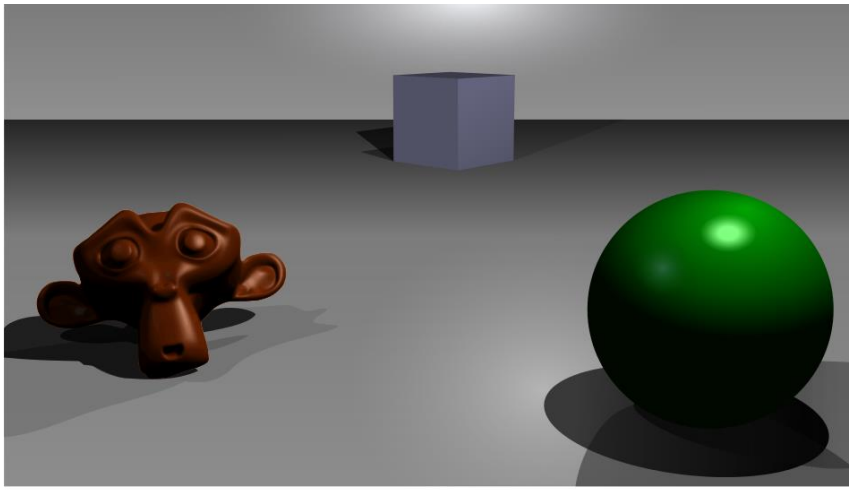
El objetivo es mostrar correctamente la percepción de profundidad, donde los objetos cercanos ocultan a los lejanos (Z-Culling).

Buffer de profundidad: Buffer Z

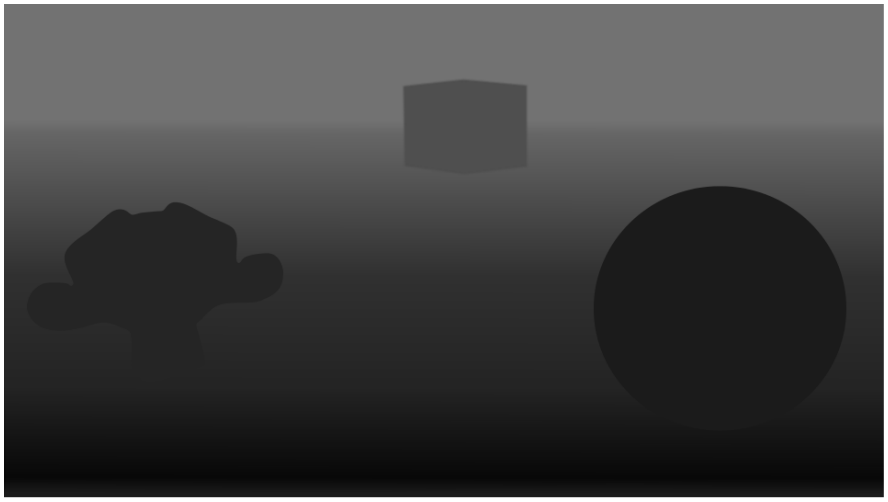


Tomado de: link

Buffer de profundidad: Buffer Z



A simple three-dimensional scene



Z-buffer representation

Tomado de: link

Buffer de profundidad: Buffer Z

Algoritmo:

1. Inicializar los arreglos correspondientes al búfer de profundidad *depthBuff(x,y)* y al búfer de imagen *frameBuff(x,y):* para todo (x,y).
2. Procesar cada polígono en la escena:
   a. Para cada (x,y) proyectado de un polígono, calcular la profundidad z.
   b. Si z < depthBuff(x,y), calcular el color de superficie para dicha posición y reemplazar el valor en el búfer de imagen.

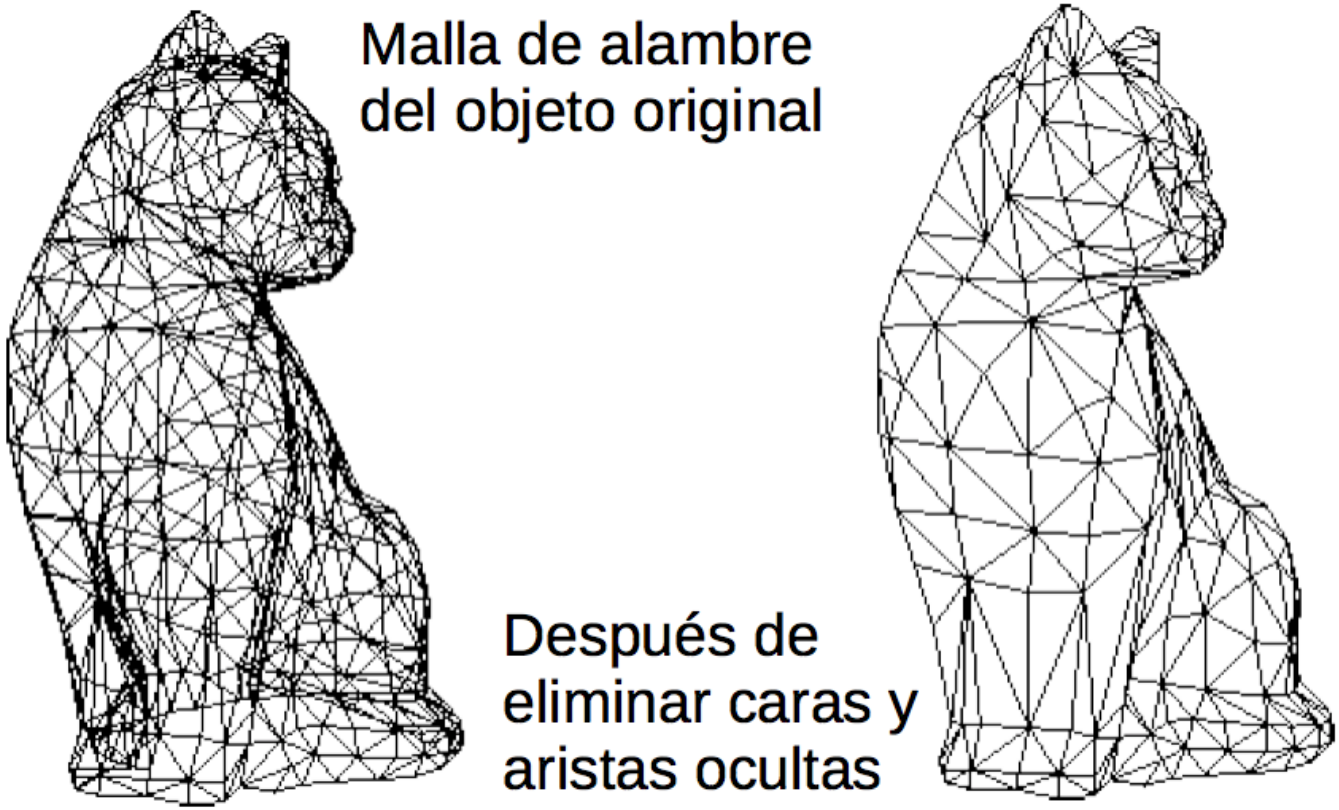
Buffer de profundidad: Buffer Z

Para cada pixel a renderizar, se almacena la profundidad del objeto renderizado más cercano (la distancia a la cámara).

Algoritmo:

```
for all (x,y) (* initializing the background *)
    depthBuff(x,y) = - ∞ //far plane
    frameBuff(x,y) = backgroundColor
for each polygon P (* loop over all polygons *)
    for each position (x,y) on polygon P
        calculate depth z
        if z > depthBuff(x,y)
            then depthBuff(x,y) = z
            frameBuff(x,y) = surfColor(x,y)
```

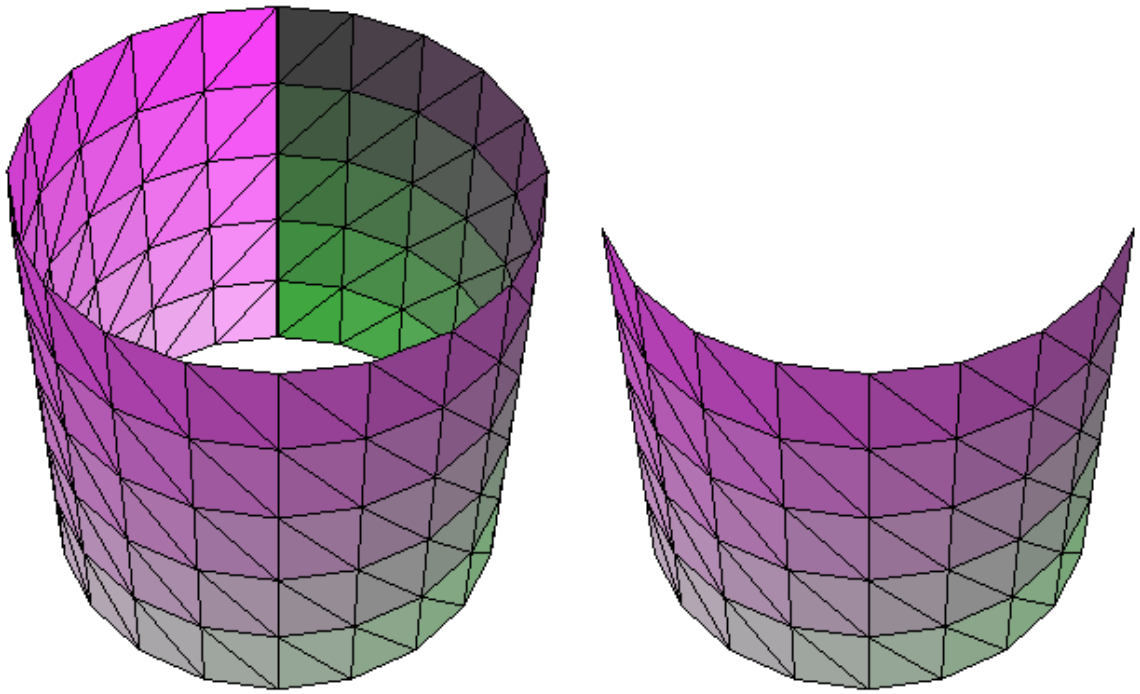
Buffer de profundidad: Buffer Z



Tomado de: Rueda A (2014).

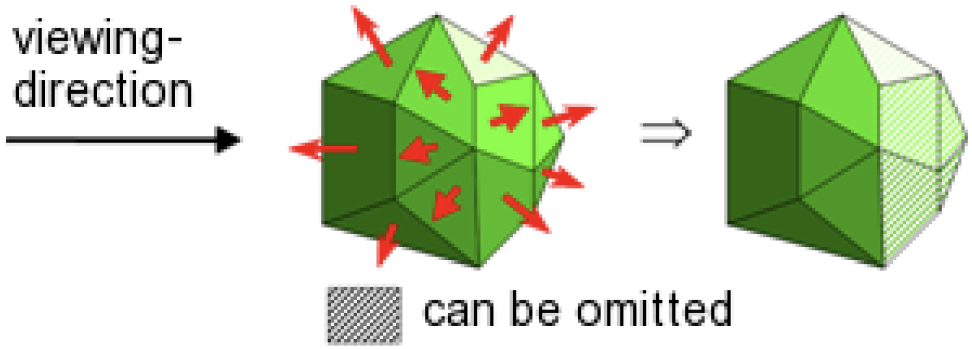
Remoción de superficies (Culling)

Consiste en eliminar aquellos objetos y/o superficies que no sean visibles a la cámara en la escena



Backface culling

Los polígonos cuya normal apunta en una dirección que se aleja de la cámara, no deberían ser visibles (se eliminan del render):

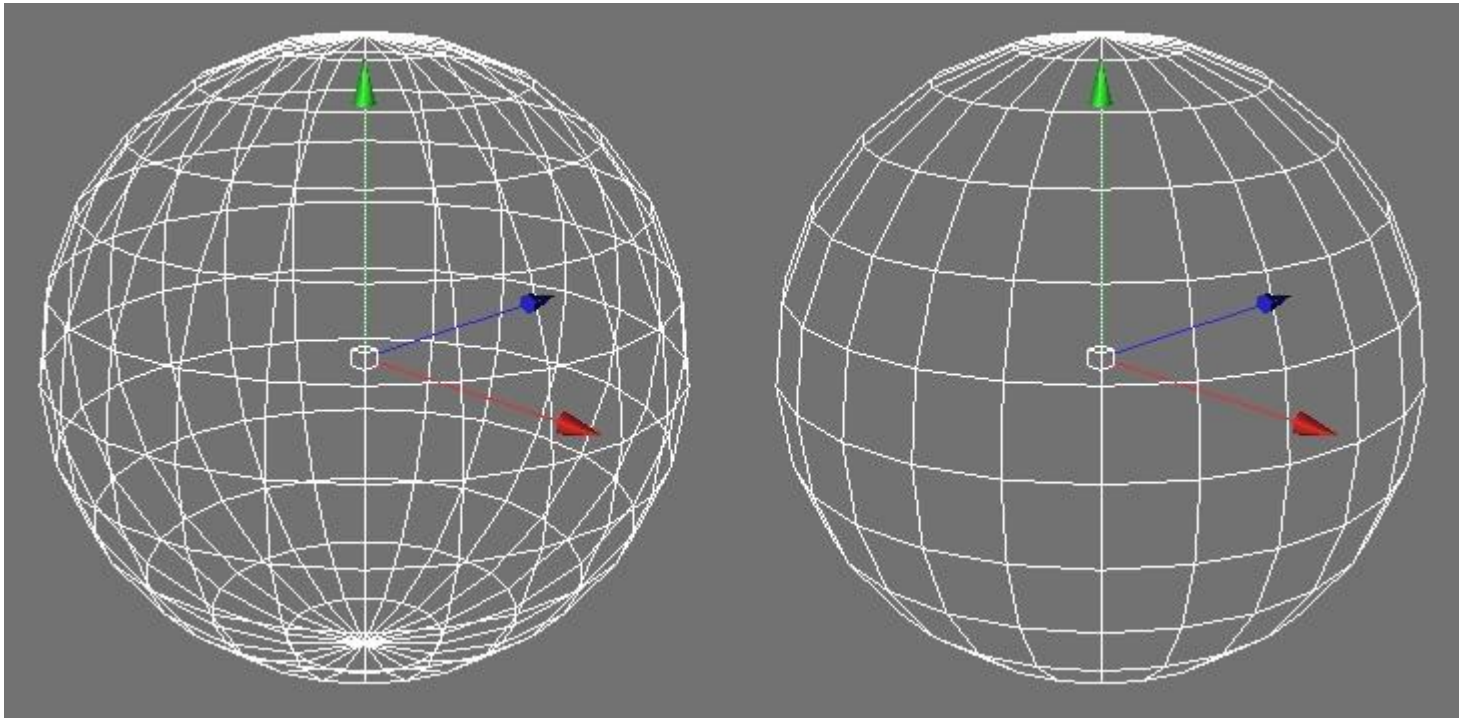


Vista ortográfica: $V_{vista} \cdot N > 0$ →Invisible

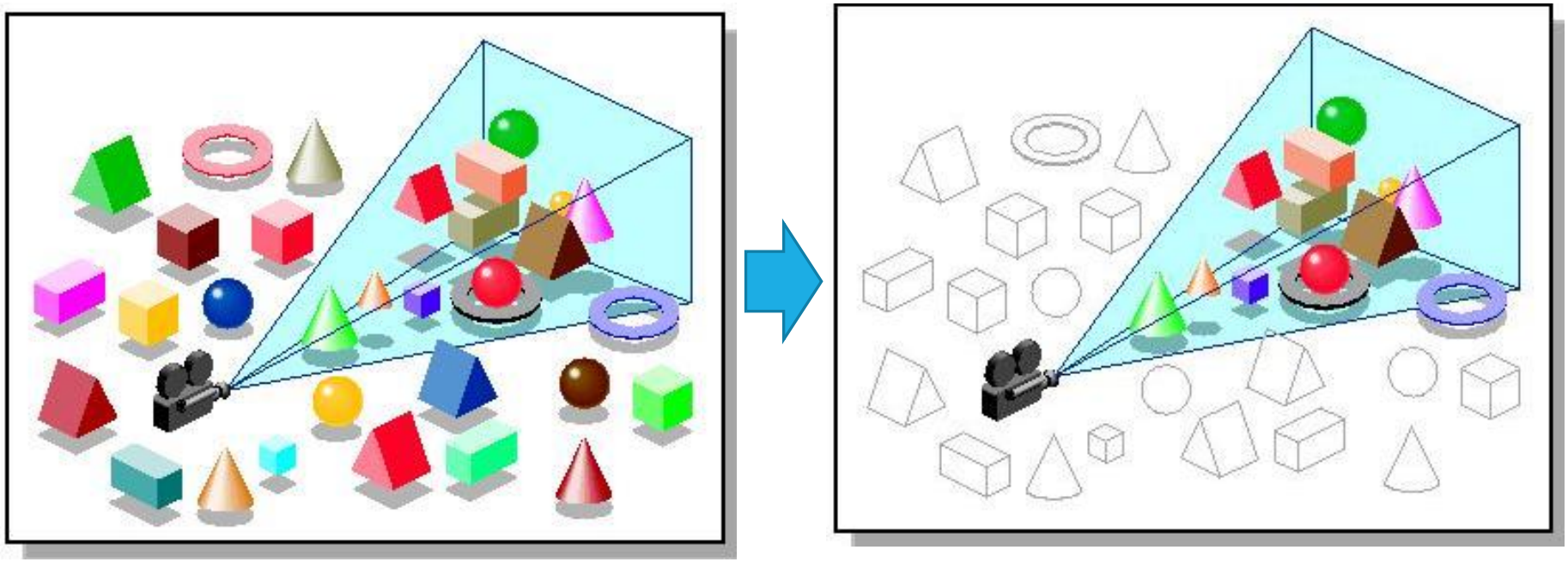
Vista perspectiva: Se transforma la escena mediante la transformada de la cámara y de visualización, y se aplica entonces la misma ecuación de la vista ortográfica.

En promedio, el 50% de los polígonos se elimina con este método.

Backface culling



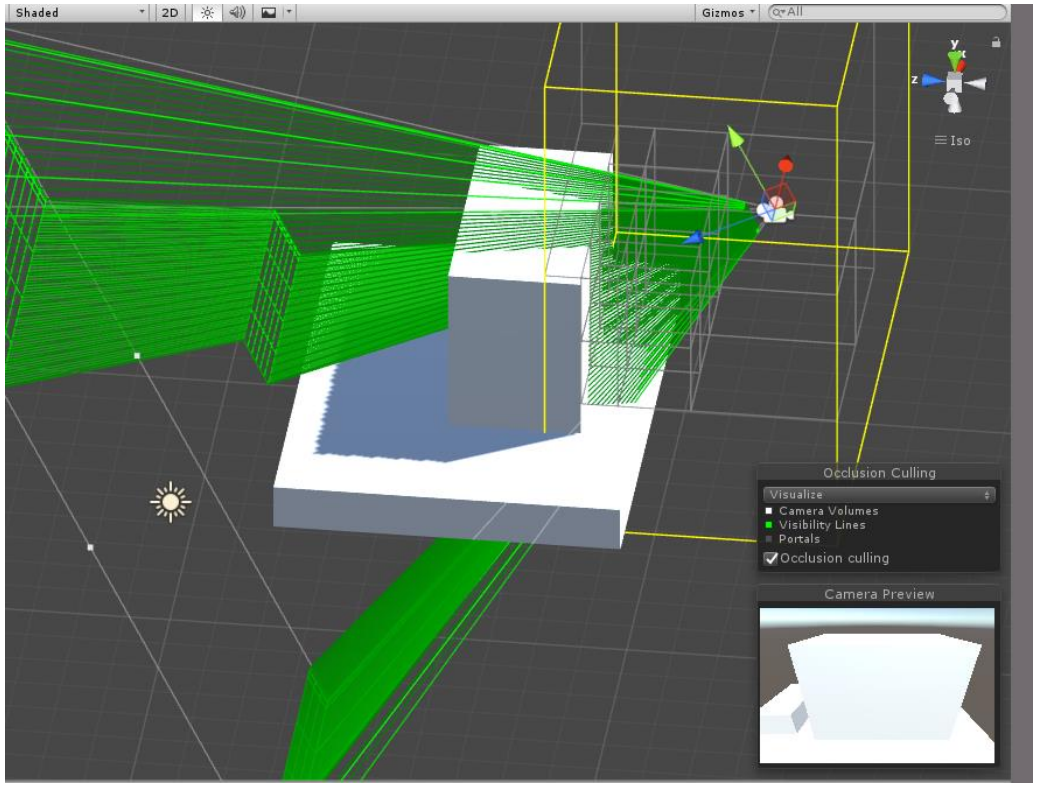
Frustum Culling



Tomado de: link

Occlusion culling

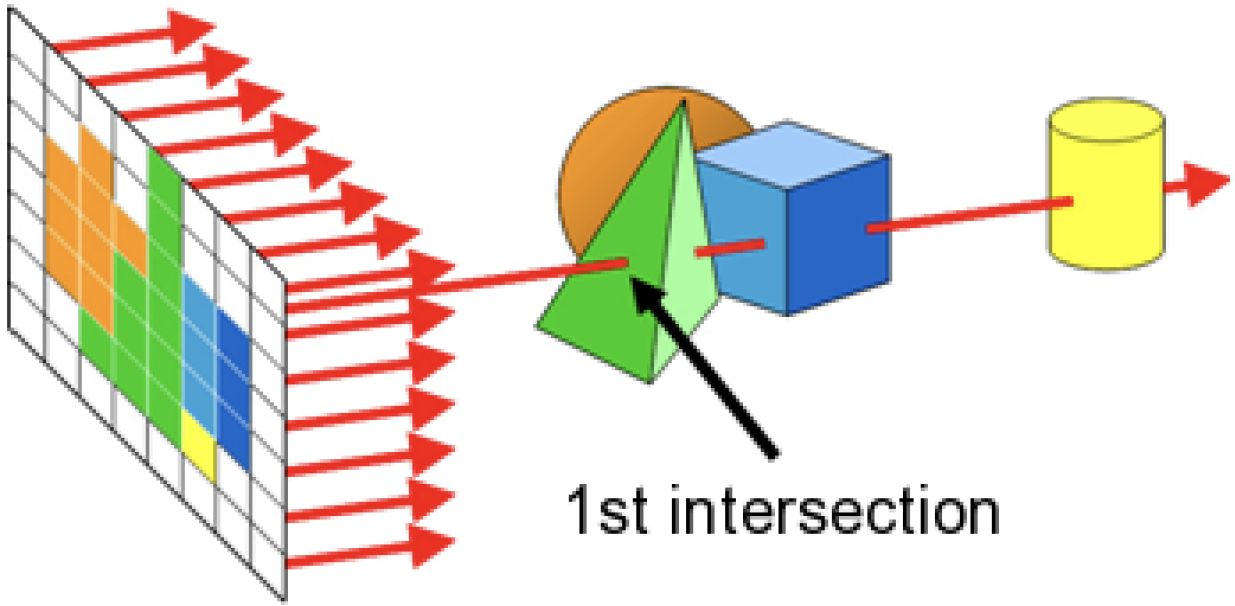
Se eliminan aquellos objetos que se encuentran tapados por otros en la escena, lo que los hace invisibles a la cámara.



Ray-Casting

Se calcula para cada pixel qué es lo que se debería ver.

Se envía un rayo desde cada pixel (en dirección de la proyección) para determinar el color a renderizar. Dicho color corresponde al primer objeto que interesecta el rayo.



Ray-Casting

Permite renderizar cualquier tipo de polígonos. Es necesario tener en cuenta la normal de la superficie, para calcular el sombreado correcto.

Se trata de un método costoso, pues requiere evaluar las intersecciones para cada rayo.

Ray-Casting

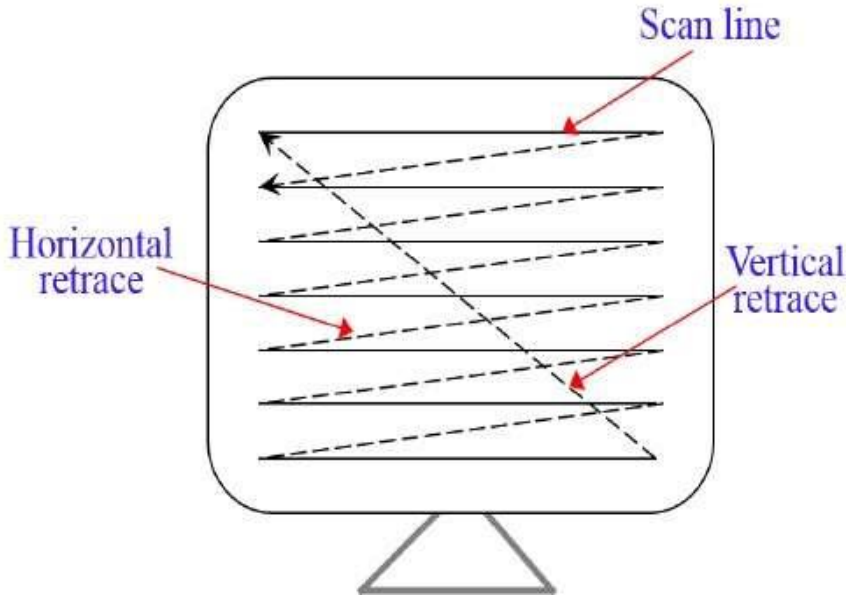
Algoritmo:

```
Ray-Casting =
for each pixel of the image-plane:
    1. generate a line through the pixel in viewing-direction (“viewing-ray”)
    2. intersect the ray with all objects
    3. choose the nearest intersection point to the viewer
    4. color the pixel with the color of the surface at this intersect. point
```

Líneas de barrido

Se hace un escaneo de toda la escena, para cada intersección de la línea con una superficie, se realizan cálculos de profundidad.

La información de color de la superficie más cercana a la cámara se envía al búfer de imagen.



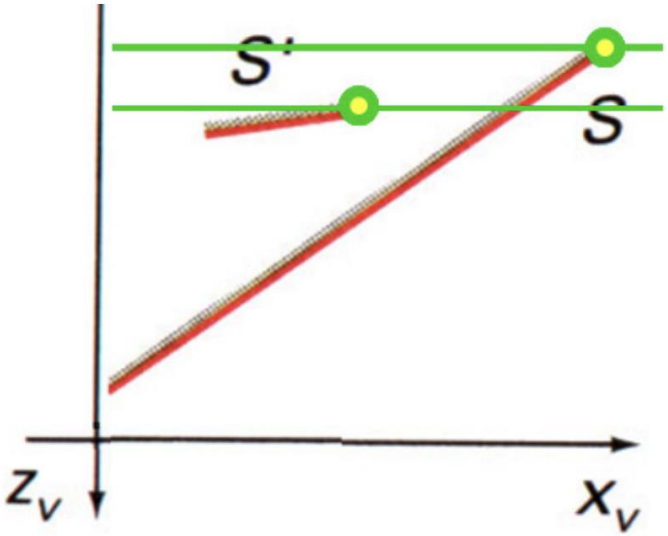
Tomado de: Rueda A.

Algoritmo del pintor. *Depth-Sorting Method*

Organizar todos los polígonos desde atrás hacia delante, y dibujarlos en este orden.

Debido a que las partes que están más lejanas quedan escondidas por las partes más cercanas, la visibilidad de los objetos es correcta.

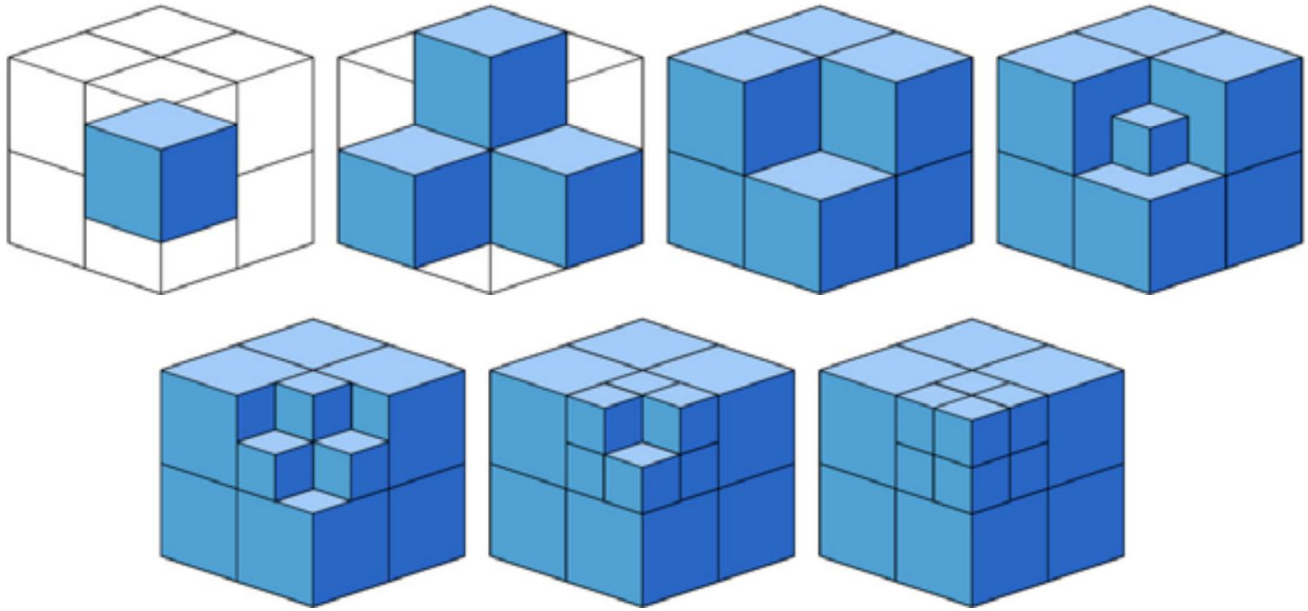
La organización de los polígonos debe hacerse de tal forma que



Octree

Si una escena es representada mediante Octrees, es posible saber qué partes de la escena serán visibles a la cámara.

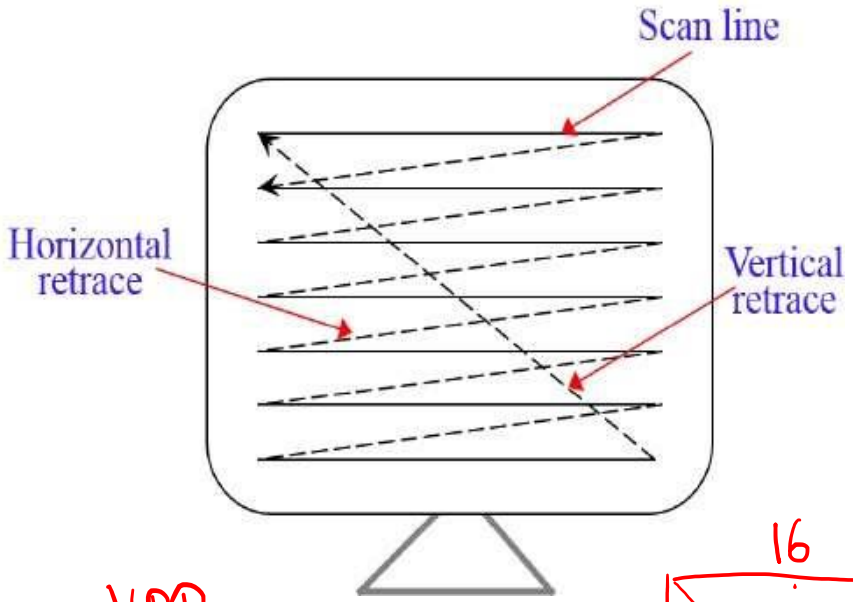
Se dibujan entonces primero el objeto en el cubo más lejano, luego los tres más cercanos

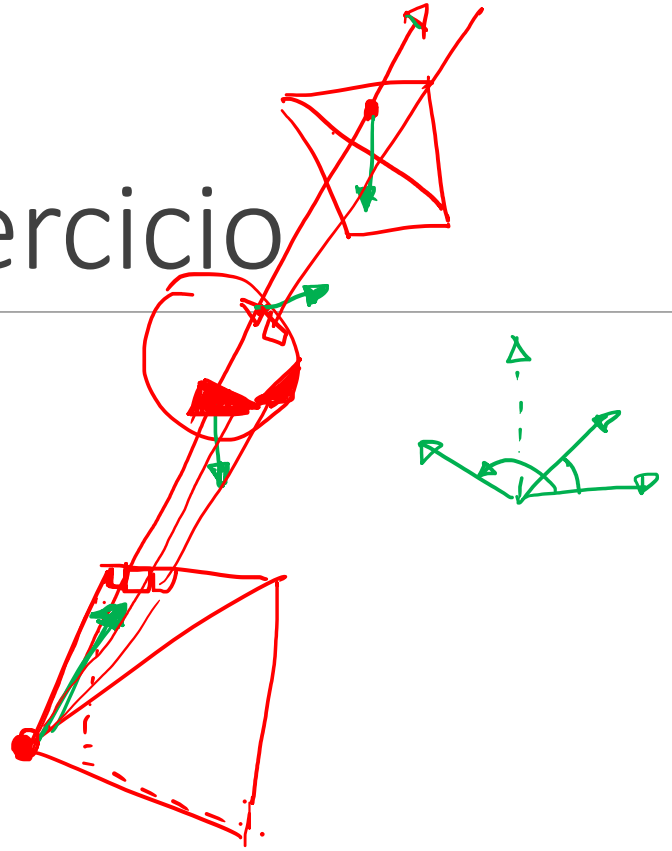


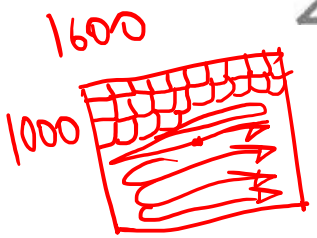
Ejercicio 1

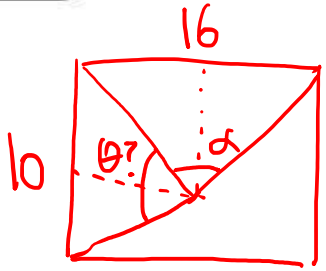
1. Crear una escena con diferentes primitivas en Three.js.
2. Utilizando RayCasting y el método de barrido, verificar las superficies cuya normal está en la misma dirección que el vector de visualización.
3. Eliminar de la escena aquellas superficies y confirmar poniendo una vista lateral que permita visualizar los volúmenes con superficies eliminadas.
4. Entregar el código desarrollado al finalizar la sesión de clase.

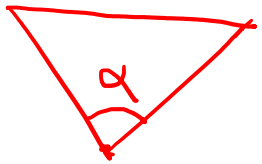
Desarrollo del ejercicio

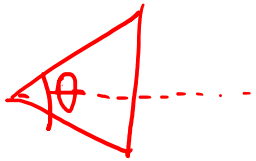






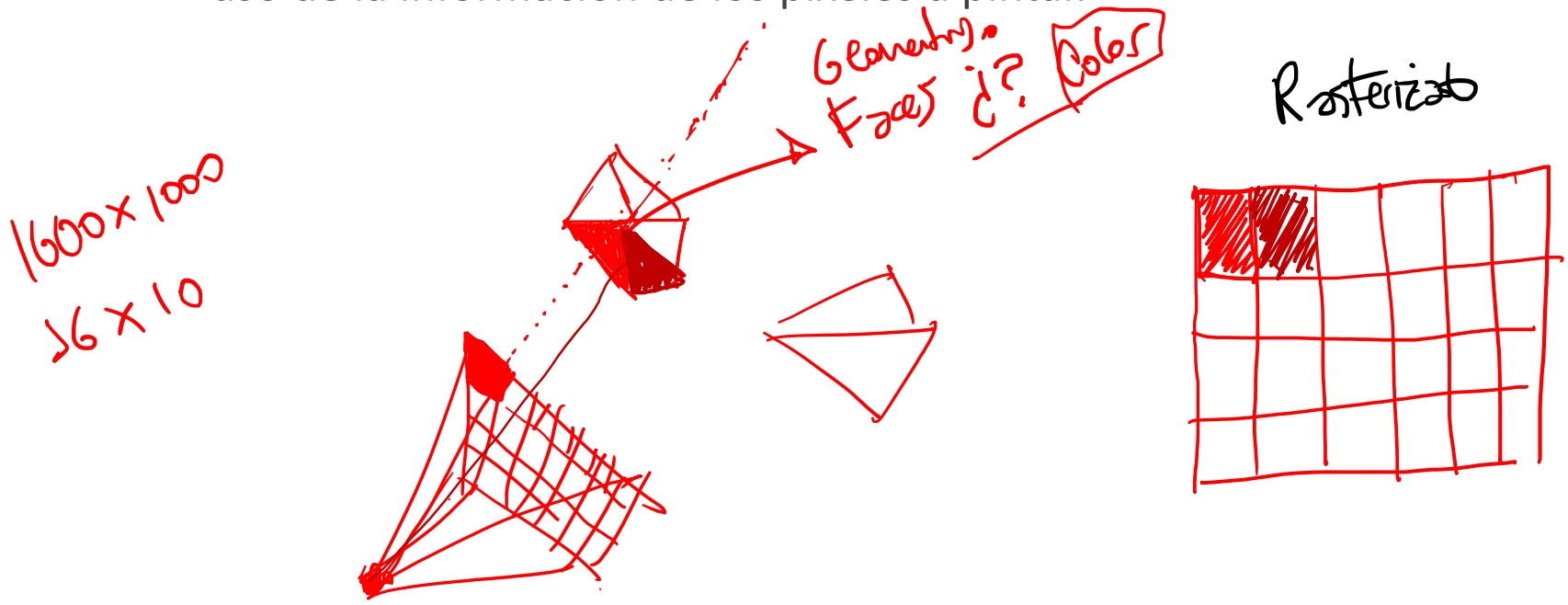






Ejercicio 2

1. Dado el resultado del ejercicio1, generar un renderizador haciendo uso de la información de los pixeles a pintar.



Bibliografía

Dunn, F. y Parberry, I. (2002). Chapter 14 - Triangle Meshes en: ***3D Math Primer for Graphics and Game Development***. Wordware Publishing, Inc.

Hughes, J et al. (2014). Chapter 8 – A simple way to describe shape in 2D and 3D en: ***Computer Graphics: Principles and Practice***. 3rd Ed. Addison-Wesley.

Rueda, A. (2014). ***Superficies visibles Modelos de iluminación y sombreado***. Introducción a la computación gráfica. Pontificia Universidad Javeriana.