El *face culling* es una técnica de optimización en gráficos 3D que se utiliza para mejorar el rendimiento de renderizado. Su objetivo es evitar dibujar las caras de los objetos que no son visibles para el espectador, como las que están orientadas hacia el lado opuesto de la cámara. Esto reduce la cantidad de cálculos que el sistema gráfico debe realizar, haciendo el proceso más eficiente.

1. Orientación de las caras

Cada cara en un modelo 3D tiene una orientación que se determina por el orden en que sus vértices se definen. Esta orientación es importante para saber si una cara está "mirando" hacia la cámara o en dirección opuesta.

- Normalmente, la orientación se define en sentido horario (*clockwise*) o antihorario (*counter-clockwise*). Por convención, una cara que tiene sus vértices en sentido antihorario (visto desde el frente) se considera como "visible".

2. Normales de las caras

Cada cara tiene una *normal*, un vector perpendicular a la superficie de la cara. Este vector ayuda a determinar si una cara está mirando hacia la cámara o en dirección opuesta. Si la normal apunta hacia la cámara, la cara es visible; si apunta en dirección contraria, se considera oculta.

3. Configuración del culling en OpenGL o DirectX

En librerías gráficas como OpenGL, puedes activar el *face culling* usando una instrucción como `glEnable(GL_CULL_FACE);` y luego especificar qué caras quieres descartar con `glCullFace(GL_BACK);` para ocultar las caras traseras o `GL_FRONT` para las frontales.

4. Proceso de eliminación de caras no visibles

Cuando el motor gráfico renderiza la escena:

- Evalúa la orientación de cada cara con respecto a la cámara.
- Determina, según el ajuste de *face culling*, si la cara debe ser renderizada.
- Si está activado el *back-face culling*, las caras que están orientadas lejos de la cámara se eliminan de la lista de caras a renderizar.

Esto reduce el número de polígonos que se procesan en la etapa de rasterización, optimizando el renderizado.

5. Beneficios de usar el face culling

- **Rendimiento**: Al evitar dibujar las caras traseras de los objetos, se reducen los cálculos necesarios en la GPU, mejorando el rendimiento.
- **Apariencia**: Al eliminar las caras traseras, se evitan problemas visuales donde se verían partes internas de los objetos cuando no deberían ser visibles.
- **Menos complejidad de sombreado**: Solo se procesan las caras que aportan a la imagen final, ahorrando recursos de iluminación y sombreado.

Ejemplo de cómo se ve en código en OpenGL:

```
glEnable(GL_CULL_FACE);      // Activar el face culling
glCullFace(GL_BACK);         // Eliminar las caras traseras
glFrontFace(GL_CCW);         // Definir el sentido antihorario como frontal
```

En resumen, el *face culling* optimiza el renderizado eliminando caras innecesarias, haciendo que el procesamiento gráfico sea más eficiente y logrando un mejor rendimiento en escenas complejas.

Para determinar si una cara en un modelo 3D es **frontal (visible)** o **trasera (no visible)**, se utiliza el concepto de *producto vectorial* y la orientación de la *normal de la cara* respecto a la posición de la cámara. Aquí te explico el fundamento matemático detallado que permite definir esta clasificación.

1. Normales de la Cara y Orientación de los Vértices

Cada cara de un polígono tiene una *normal*, que es un vector perpendicular a la superficie de la cara. La normal puede determinarse mediante el producto vectorial de dos de los lados de la cara (específicamente, de dos vectores formados a partir de sus vértices).

Supongamos que tenemos una cara triangular definida por tres puntos o vértices en el espacio:

- $V_1(x_1, y_1, z_1)$
- $V_2(x_2, y_2, z_2)$
- $V_3(x_3, y_3, z_3)$

Para obtener la normal de la cara:

1. Calculamos dos vectores en el plano de la cara:

$$\vec{A} = V_2 - V_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

$$\vec{B} = V_3 - V_1 = (x_3 - x_1, y_3 - y_1, z_3 - z_1)$$

2. Obtenemos la **normal** (\vec{N}) usando el producto vectorial entre (\vec{A}) y (\vec{B}):

$$\vec{N} = \vec{A} \times \vec{B} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix}$$

Expandiendo el determinante:

$$\vec{N} = ((y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_1))\hat{i} - ((x_2 - x_1)(z_3 - z_1) - (z_2 - z_1)(x_3 - x_1))\hat{j} + ((x_2$$

La normal (\vec{N}) obtenida define la dirección de la cara en el espacio.

2. Vector de la Cámara y Producto Escalar

Para determinar si esta cara es visible desde la cámara (frontal) o está orientada hacia el lado opuesto (trasera), necesitamos:

1. La **posición de la cámara** en el espacio, $C(x_c, y_c, z_c)$.
2. Un vector que va desde la cámara hacia cualquier punto en el plano de la cara, llamado el **vector de visión** o **vector de la cámara**.

Tomando el punto (V_1) en el plano de la cara, el vector de visión desde la cámara hacia este punto es:

$$\vec{V} = V_1 - C = (x_1 - x_c, y_1 - y_c, z_1 - z_c)$$

3. Clasificación Usando el Producto Escalar

Ahora, calculamos el **producto escalar** entre el vector de visión (\vec{V}) y la normal (\vec{N}):

$$\text{Producto Escalar} = \vec{N} \cdot \vec{V} = N_x \cdot (x_1 - x_c) + N_y \cdot (y_1 - y_c) + N_z \cdot (z_1 - z_c)$$

Este producto escalar tiene un significado importante:

- **Si el producto escalar es positivo**, la cara está orientada hacia la cámara, por lo tanto es **frontal** y debería renderizarse.
- **Si el producto escalar es negativo**, la cara está orientada en la dirección opuesta a la cámara, por lo tanto es **trasera** y se puede descartar.
- **Si el producto escalar es cero**, la cara es perpendicular a la dirección de la cámara, lo que significa que la cámara ve el borde de la cara (esto es raro en modelos prácticos, pero puede ocurrir).

4. Aplicación del Culling en el Pipeline de Renderizado

Esta clasificación de las caras, en **frontales** o **traseras**, se realiza en el pipeline gráfico antes de la etapa de rasterización. Cuando se activa el *face culling* (ej., `GL_CULL_FACE` en OpenGL), el sistema gráfico solo renderiza las caras que cumplen con la condición de visibilidad basada en el producto escalar.

En resumen:

1. **Calcula la normal** de cada cara.
2. **Define el vector de visión** entre la cámara y un punto en la cara.
3. **Calcula el producto escalar** entre la normal de la cara y el vector de visión.
4. **Clasifica la cara** como frontal o trasera dependiendo del signo del producto escalar.

Este proceso ayuda a optimizar el renderizado evitando el cálculo y la visualización de caras que no contribuyen a la imagen final.