Instituto Tecnológico de Morelia

práctica 02: backface

Graficación

Profesor:
Martinez Guzman Bryan Eduardo

Santiago Gonzalez Lara 22121360
Mayavi Sarael Gonzalez Ornelas 22120661

7 de marzo del 2025

BACKFACE CULLING

Un algoritmo fundamental en el proceso de renderizado es el Back-face Culling, una técnica utilizada para determinar qué caras de un objeto tridimensional no necesitan ser renderizadas porque están orientadas en dirección opuesta a la cámara.

Cuando se modelan objetos en un entorno 3D, estos están compuestos por múltiples caras (o polígonos). Sin embargo, muchas de estas caras nunca serán visibles desde una determinada perspectiva, ya que están orientadas en sentido contrario al observador. Renderizar estas caras innecesariamente supone un desperdicio de recursos computacionales, por lo que el Back-face Culling permite descartarlas antes de que lleguen a la etapa de rasterización, optimizando el rendimiento del sistema gráfico.

aplicación

```python
import numpy as np




class Face:

   def __init__(self, vertices):


       """ Recibe una lista de 3 puntos (vértices) que forman una cara. """

       self.vertices = np.array(vertices)

       self.normal = self.calculate_normal()


   def calculate_normal(self):

       """ Calcula el vector normal usando el producto cruzado entre dos
aristas de la cara. """

       v0, v1, v2 = self.vertices

       edge1 = v1 - v0

       edge2 = v2 - v0

       normal = np.cross(edge1, edge2)
```

```python
        return normal / np.linalg.norm(normal)  # Normaliza el vector



def backface_culling(faces, camera_position):
    """
    Evalúa qué caras deben renderizarse según el algoritmo Back-face Culling.
    - faces: lista de objetos Face
    - camera_position: posición del observador en 3D
    """
    visible_faces = []

    for face in faces:
        # Calcula un vector desde el primer vértice de la cara hasta la cámara
        view_vector = camera_position - face.vertices[0]
        view_vector = view_vector / np.linalg.norm(view_vector)  # Normaliza

        # Producto escalar entre la normal y el vector de vista
        dot_product = np.dot(face.normal, view_vector)

        # Si el producto es positivo, la cara es visible
        if dot_product > 0:
            visible_faces.append(face)

    return visible_faces
```

```python
# ==== PRUEBA ====

# Definimos algunas caras de un cubo

faces = [

   Face([[0, 0, 0], [1, 0, 0], [1, 1, 0]]),  # Frente

   Face([[0, 0, 0], [0, 1, 0], [1, 1, 0]]),

   Face([[0, 0, 1], [1, 0, 1], [1, 1, 1]]),  # Atrás

   Face([[0, 0, 1], [0, 1, 1], [1, 1, 1]]),

]



# Posición de la cámara

camera_position = np.array([0.5, 0.5, -1])  # Frente al objeto



# Filtramos las caras visibles

visible_faces = backface_culling(faces, camera_position)



print(f"Caras visibles: {len(visible_faces)} de {len(faces)}")
```

resultado

```
C:\Users\reliv\AppData\Local\Microsoft\WindowsApps\python3.11.exe "C:\Users\reliv\PycharmProjects\ISC-ITM\6to\Graficación\actividades\actividad 7\backface.py"
Caras visibles: 2 de 4

Process finished with exit code 0
```

APLICACIÓN

INSTITUTO TECNOLÓGICO DE MORELIA
ITM
"José María Morelos y Pavón"