Informática Gráfica

Juan Carlos Torres

Curso 2024/25

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Disclaimer
You can edit this page to suit your needs. For instance, here we have a no copyright statement, a colophon and some other information. This page is based on the corresponding page of Ken Arroyo Ohori's thesis, with minimal changes.

Colophon
This document was typeset with the help of KOMA-Script and LaTeX using the kaobook class.

Conectividad en mallas 10

La conectividad (o topología) de una malla describe cómo están conectados los elementos geométricos entre sí (p.e. que poligonos son vecinos de cada polígono). La información de conectividad es crucial porque define la estructura del objeto, permitiendo realizar operaciones como deformaciones, subdivisiones y otras transformaciones geométricas que son esenciales en la animación y modelado 3D de forma eficiente. La representación de sopa de triángulos vista en el capítulo 2 es muy simple pero tiene muchas limitaciones por no almacenar información de conectividad (Figura 10.1).

La mayor parte de las operaciones requieren acceder a los elementos vecinos, permitiendo que los algoritmos avancen (o naveguen) por la superficie. El que estas relaciones estén almacenadas de forma explícita en la representación influye en la eficiencia de los algoritmos.

A modo de ejemplo, pensemos que queremos saber si una malla es abierta o cerrada [1]. Si utilizamos la representación indexada cara-vértice (Sección 2.4) podemos usar un algoritmo iterativo que para cada par de vértices consecutivos de cada cara busque si hay otra cara que los contenga en orden inverso. Este algoritmo tiene complejidad cuadrática con el número de caras.

Algunas de las operaciones usadas frecuentemente en algoritmos que trabajan con mallas de polígonos y que requieren información de conectividad son [Bot+10]:

a) Acceso ordenado a vértices de una cara.
b) Acceso a caras que inciden en una arista.
c) Acceder a los vértices de una arista.
d) Acceder a las caras de un vértice.

Para facilitar la navegación por la malla se puede incluir en la representación información de adyacencia entre elementos, normalmente como enlaces entre elementos:

- cara-cara: Cada cara indexa a sus caras vecinas
- cara-vértice: Esta es la única relación almacenada en la estructura indexada vista en la sección 2.4.
- cara-arista: Se almacenan explícitamente las aristas y cada cara tiene el identificador de sus aristas.
- arista-cara: Cada arista indexa sus dos caras. Si la malla es abierta puede tener solo una cara.
- arista-vértice: Cada arista indexa sus dos vértices.
- arista-arista: Cada arista indexa las aristas que continúan a partir de sus dos vértices.
- vértice-vértice: Cada vértice almacena la dirección de sus vértices vecinos (a través de las aristas).
- vértice-arista: Cada vértice indexa sus aristas.
- vértice-cara: Cada vértice indexa las caras que lo contienen.

Figura 10.1: Es fácil hacer sopas, pero no le gustan a todo el mundo.

1: Una malla es cerrada si todos los polígonos tienen un polígono vecino a través de cada arista

Cada relación almacenada facilita la navegación en la malla a costa de aumentar el espacio ocupado y la complejidad de algunas operaciones, especialmente las que editan la malla. No es por tanto necesario, ni conveniente almacenarlas todas, siempre que las operaciones necesarias se puedan realizar con un número fijo de accesos.

Grafos de navegación

El grafo de navegación de un representación es un diagrama que muestra las relaciones de adyacencia que están presentes en ella. En el aparecen los tres elementos geométricos de la malla: Vértices (Vertices), Aristas (Edges) y Caras (Faces). Un arco en el grafo indica que la relación está almacenada. La figura 10.2 muestra el grafo de navegación completo incluyendo todas las posibles relaciones.

La representación de vértices indexados descrita en el listado 2.3 no contienen aristas, su grafo de navegación es el que se muestra en la Figura 10.3.

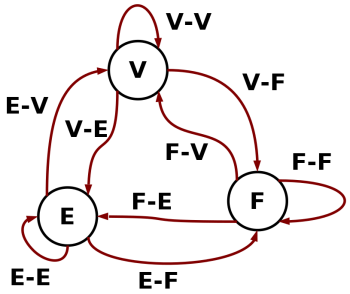

Figura 10.2: Grafo de navegación mostrando todas las posible relaciones.

10.1. Representación de mallas

Introducir las aristas en la estructura es simple: las caras almacenan una lista de aristas, y las aristas sus dos vértices. Accedemos a los vértices a través de las aristas (Figura 10.4). Pero esto no nos permite resolver las operaciones b y d en tiempo constante

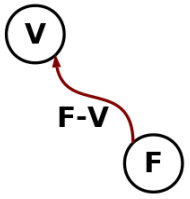

Figura 10.3: Grafo de navegación de una representación de vértices indexados.

10.1.1. Aristas aladas

Una de las formas mas usadas de representar mallas de polígonos es mediante la estructura de aristas aladas. El elemento central en ella son las aristas. De cada cara se almacena solamente el identificador de una de sus aristas. Para cada arista se almacena, además de sus vértices (V_i y V_f), las dos caras que separa la arista (C_d y C_i), y las aristas adyacentes en cada una de estas caras (Figura 10.6). Esta estructura es particularmente útil para algoritmos que necesitan navegar rápidamente a través de la malla, como aquellos usados en refinamiento de mallas, detección de colisiones, o aplicación de texturas.

Asumiendo una ordenación CCW de las caras, las caras directa e inversa se identifican por el sentido de recorrido de la arista. En la cara directa (C_d) la arista se recorre tal como está definida en la estructura (de V_i a V_f). En la cara inversa se recorre en sentido contrario.

El grafo de navegación de la estructura se muestra en la Figura 10.8. Podemos observar como es posible navegar desde cualquier entre cualquier par de elementos bien de forma directa o indirectamente, pasando por las aristas. Con esta estructura es posible realizar las operaciones b) y c) con un acceso a la estructura y la d) con dos accesos.

Cara	Aristas				
1	0	1	2	3	NULL
2	4	5	6	1	NULL
3					

Aristas	V_i	V_f
0	0	1
1	1	2
2	2	3
3	3	0
4	1	5
5	5	4
6	4	2

Vértice	X	Y	Z
0	0	0	0
1	0	2	-1
2	0	3	2
3	0	1	2
4	2	4	1
5	2	3	-1

Figura 10.4: Estructura caras-aristas-vertices.

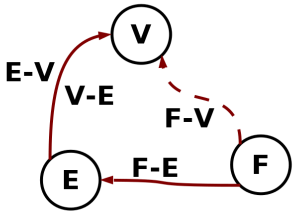

Figura 10.5: Grafo de navegación de la estructura caras-aristas-vértices.

Caras	
Id	Arista
0	2
1	5
2	1
3	3
4	4

Aristas								
Id	Vi	Vf	Cd	Ci	Ada	Ads	Aia	Ais
0	0	1	2	4	3	1	4	8
1	1	2	2	1	0	2	6	4
2	2	3	2	0	1	3	7	6
3	3	0	2	3	2	0	8	7
4	1	5	1	4	1	5	8	0
5	5	4	1	3	4	6	7	8
6	4	2	1	0	5	1	2	7
7	3	4	3	0	3	5	6	2
8	5	0	3	4	5	3	0	4

Vértices				
Id	X	Y	Z	Arista
0	0	0	1	0
1	1	0	1	1
2	1	1	1	2
3	0	1	1	3
4	1	1	0	6
5	1	0	0	8

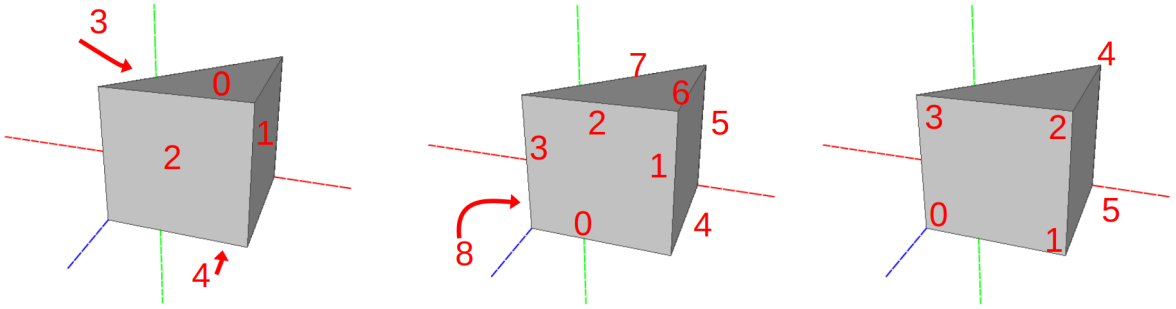

Figura 10.7: Estructura aristas aladas, rellenada con los datos de un prisma triangular. En la imagen se muestran de izquierda a derecha los identificadores de caras, aristas y vértices.

Podemos crear una estructura de aristas aladas a partir una sopa de polígonos, creando la tabla de aristas cuando se están leyendo las caras:

```
Para cada vertice
  Agregar a la tabla de vertices
Para cada cara
  Para cada par de vertices (Va,Vb)
    Si existe la arista (Vb,Va)
      Agregar cara como cara inversa
    si no
      Agregar arista a la lista de aristas
      Agregar cara como cara directa
  Agregar cara a lista de caras
  Recorrer aristas y rellenar enlaces a arista siguiente
```

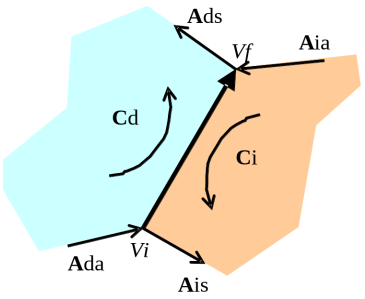

Figura 10.6: Información asociada a las aristas en una representación de aristas aladas.

Para obtener los vértices de una cara se deben realizar n+1 acceso, siendo n el número de vértices. Una vez que accedemos a la primera arista de la cara tendremos que comprobar si la cara que buscamos es la cara directa o inversa de la arista, para obtener los vértices V_i y V_f en el orden correcto y encontrar la arista siguiente de la cara (siguiendo los enlaces A_{ds} o A_{is}).

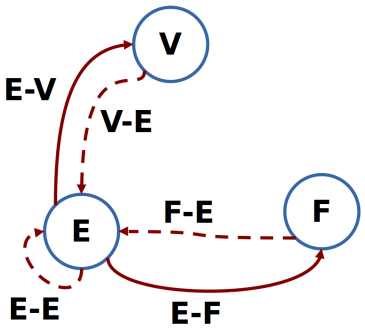

Figura 10.8: Grafo de navegación de una estructura de aristas aladas.

10.1.2. Normales de vértice en superficies discontinuas

En la sección 2.3.1 vimos que se puede asociar una normal a los vértices calculándola como un promedio de las normales en las caras adyacentes. Esto nos permite calcular una normal que genera una transición suave en la superficie, y por tanto es adecuada para superficies continuas. La Figura 10.9 muestra dos visualizaciones del mismo cilindro. El de la izquierda se ha dibujado con normales de vértice calculadas como promedio de las normales de cara. Puede observarse como desaparecen las discontinuidades en las aristas.

Cuando se quiere que en la superficie de un objeto aparezcan discontinuidades en la orientación de la superficie es necesario utilizar mas de un valor en las normales de vértice.

Esto puede conseguirse, duplicando vértices, o haciendo que la representación del objeto esté formado por mas de una superficie o modificando la estructura de datos para que cada vértice pueda tener mas de una normal.

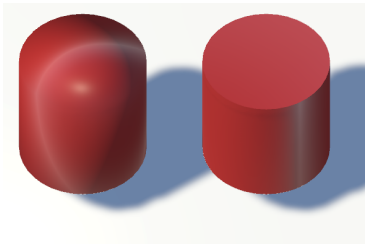

Figura 10.9: Efecto del calculo de la normal de vértice como promedio de las normales de cara.

10.1.3. Coordenadas de textura

Un problema parecido aparece cuando se desenvuelve una malla para calcular coordenadas de textura. Los vértices que están en las costuras tiene mas de una coordenada de textura. MeshLab puede usar coordenadas de textura asociadas a vértice o *wedge*, en este último caso almacena para cada vértice una normal en cada una de sus caras.

10.1.4. Semiaristas aladas

Esta es el principal inconveniente de esta estructura. Para evitarlo se puede usar **semiaristas aladas**. En esta estructura la información de las aristas se almacena como dos semiaristas, cada una contiene solamente la información de una cada y el identificador de su semiarista compañera (figura 10.11). De esta forma se mantiene la misma información topológica facilitando la obtención de la secuencia de aristas(o vértices) de las caras.

En una estructura de semiaristas aladas La información de normales de vértice y de coordenadas de textura se puede almacenar en las aristas, permitiendo que cada vértice tenga asociados valores diferentes en cada cara. Esta información se puede asociar, por ejemplo, al vértice inicial de cada arista.

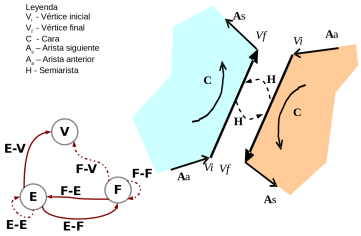

Figura 10.10: Estructura de semiaristas aladas.

Cara	Arista
0	2
1	5
2	1
3	3
4	4

Saristas	Vi	Vf	h	C	Aa	As
0	0	1	9	2	3	1
1	1	2	10	2	0	2
2	2	3	11	2	1	3
3	3	0	12	2	2	0
4	1	5	13	1	1	5
5	5	4	14	1	4	6
6	4	2	15	1	5	1
7	3	4	16	3	3	5
8	5	0	17	3	5	3
0	1	0	0	4	4	8
1	2	1	1	1	6	4
2	3	2	2	0	7	6
3	0	3	3	3	8	7
4	5	1	4	4	8	0
5	4	5	5	3	7	8
6	2	4	6	0	2	7
7	4	3	7	0	6	2
8	0	5	8	4	0	4

Vértice	X	Y	Z
0	0	0	0
1	0	2	-1
2	0	3	2
3	0	1	2
4	2	4	1
5	2	3	-1

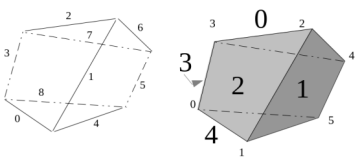

Figura 10.11: Estructura de semiaristas aladas para el modelos de la figura 10.7.

10.2. Problemas topológicos

El objetivo de guardar explícitamente información de conectividad es poder realizar de forma eficiente algunas operaciones en la malla, concretamente operaciones que requieren consultar el entorno de un elemento.

Independientemente de como se represente la malla puede tener ciertos problemas topológicos que dificultan su procesamiento. Uno de los problemas más comunes es la condición de **non-manifold** que se produce cuando una arista es compartida por más de dos caras o un vértice es compartido por dos láminas de superficie que se tocan en él (Figura 10.12). Esto hace que la malla no pueda ser desplegada en un plano sin superposiciones.

También genera problemas la no **orientabilidad** de la superficie, que impide definir un interior y un exterior consistentes; dos ejemplos clásico son la banda de Möbius y la botella de Klein.

Además, se pueden producir errores si las caras de una malla se cruzan entre sí (generando **autointersecciones**). Esto también genera errores de renderizado.

Otra situación que genera problemas topológicos es la existencia de vértices en T. Un **vértice en T** es un vértice que se encuentra en mitad de una arista, haciendo que esa arista limite con dos aristas.

Estos problemas pueden aparecer como resultado del procesamiento de una malla correcta, y pueden producir desde errores en el procesamiento de la malla hasta hacer que el algoritmo aborte.

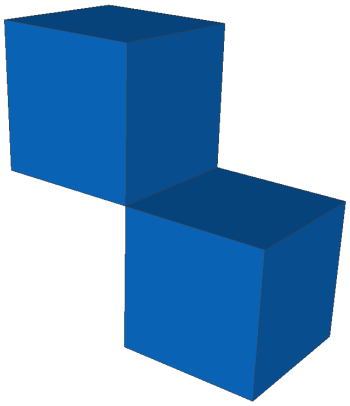

Figura 10.12: Ejemplo de objeto no manifold.

Corrección de topología con meshLab

Para corregir los problemas topológicos se puede usar MeshLab. La detección de elementos no manifold se puede realizar configurando los parámetros de dibujo de la malla (Figura 10.13).

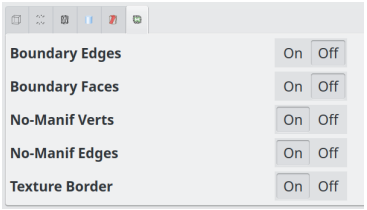

Figura 10.13: Detección de elementos no manifold en MeshLab.

Para corregir estos elementos se puede usar los filtros de limpieza y reparación:

```
Filters  > Cleaning and Repairing
```

En este grupo hay operaciones específicas para cada uno de los problemas topológicos. Algunas de las operaciones simplemente borran (o seleccionan para que después se borren) los elementos afectados. En estos casos puede ser necesario rellenar las fisuras creadas usando el filtro **Close Holes**:

```
Filters  > Remeshing, Simplification and Reconstruction > Close Holes
```

10.3. Creación de mallas por barrido

Para crear la malla que representa un objeto es necesario generar las coordenadas de sus vértices e indicar como se agrupan formando polígonos. Cuando el objeto tiene simetría podemos generar esta información a partir de un perfil de la superficie utilizando la ecuación de la simetría para generar el resto del modelo (barriendo el espacio con el perfil).

Si el barrido se realiza girando el perfil se obtiene una superficie con simetría de revolución. La Figura 10.14 muestra dos ejemplos de superficies creadas por revolución de un perfil, haciéndolo girar respecto al eje Y. En la parte superior el perfil es una circunferencia y la superficie generada es un toro. La imagen inferior es una forma cerrada creada por una curva abierta que toca al eje Y en sus extremos.

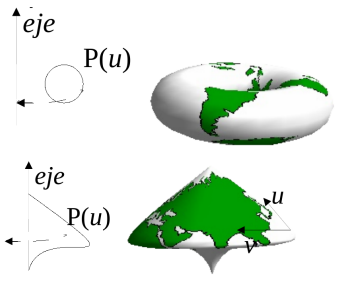

Figura 10.14: Ejemplos de objetos creados por revolución.

Si se quiere crear la malla de revolución utilizando el eje Y como eje de simetría y se parte de un perfil contenido en el plano $Z = 0$ como el de la parte superior de la Figura 10.15 formado por cuatro vértices, crearemos varias copias del perfil rotándolo respecto al eje Y:

$$V_{ik}.x = V_{0k}.x\ cos(2\pi i/N)$$

$$V_{ik}.z = V_{0k}.x\ sin(2\pi i/N)$$

siendo N el número de perfiles utilizados.

Una vez creada la lista de vértices se construyen los elementos (cuadriláteros o triángulos) uniendo los vértices de perfiles consecutivos. Si construimos triángulos podemos formar los siguientes[2]:

$$(V_{ik}, V_{(i-1)k}, V_{i(k+1)}\ \ y\ \ (V_{(i-1)k}, V_{i(k+1),V_{(i-1)(+1)}}\ \ i \in (1,N), k \in (0,M-1)$$

En el cierre de la malla se puede no duplicar los vértices del perfil original, utilizando como valor k en la ecuación anterior $k\,\%(M-1)$.

El calculo de los vectores normales de vértices se puede realizar como se vio en el capítulo anterior 2.3.1 o calculándolas a partir de la rotación perfil lo que asegura que se correspondan con la superficie de revolución que se pretende aproximar. La normal del perfil de partida es:

$$(N_{0k}.x = V_{0(k-1)}, V_{0(k+1)}\ y\ (V_{(i-1)k}, V_{i(k+1),V_{(i-1)(+1)}}\ i \in (1,N), k \in (0,M-1)$$

El mismo proceso se puede seguir para crear mallas por barrido lineal, esto es, trasladando el perfil a lo largo de un vector. En la Figura 10.16 se ha creado una superficie cilíndrica trasladando la circunferencia $P(u)$ con el vector **r**.

Este proceso se puede generalizar haciendo que la trayectoria que sigue el perfil sea una curva, desplazando y orientando el perfil a lo largo de la curva. La Figura 10.17 muestra la superficie creada por barrido de una circunferencia a lo largo de un perfil helicoidal[3].

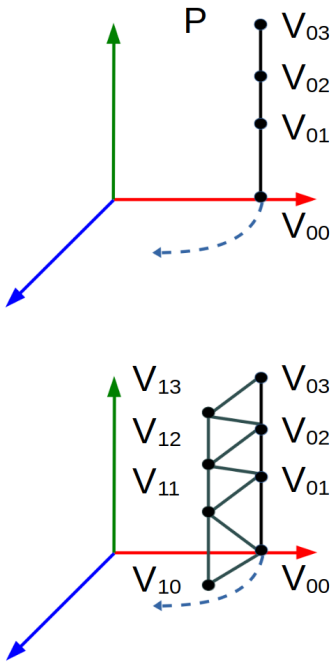

Figura 10.15: Creación de una malla de revolución.

2: Se debe tener en cuenta que el sentido de recorrido debe ser antihorario.

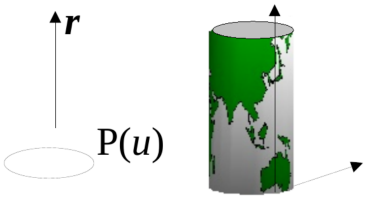

Figura 10.16: Creación de una malla por barrido lineal.

3: (https://www.computeraideddesignguide.com/basic-tools-in-3d-in-autocad)

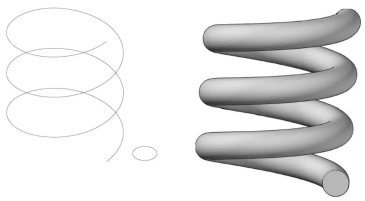

Figura 10.17: Creación de una malla por barrido lineal.

4: Obviamente esto no lo notaremos si el modelo es pequeño.

10.4. Mallas indexadas en OpenGL

El mecanismos de dibujo que hemos visto (glBegin // glVertex // glEnd) transfiere los vértices de los elementos a dibujar de la CPU a la GPU en cada frame. Esto hace que el rendimiento de la aplicación tenga un cuello de botella por la velocidad de transferencia entre ambas [4]. Desde que las GPU tienen memoria suficiente es mucho mas eficiente alojar los modelos en la GPU y realizar la transferencia de datos de la CPU solamente cuando se cambia el modelo.

El mecanismo de dibujo en OpenGL ha evolucionado en paralelo al desarrollo de las tarjetas gráficas para mejorar la eficiencia reduciendo la comunicación entre CPU y GPU, basados en transferir la información a dibujar empaquetada (usando estructuras denominadas **Vertex Buffer Objet**, VBO).

Una revisión histórica de la evolución de estos mecanismos se puede leer en la web The history of opengl vertex data.

10.5. Ejercicios

1. Recorrer la estructura de la figura 10.7 para encontrar los vértices de la cara 1.
2. Calcula el espacio ocupado por estructura de aristas aladas en función del número de elementos (vértices, caras y aristas) y del espacio ocupada por cada coordenada y cada índice.
3. La formula de Euler establece una relación entre los números de caras (C), aristas (A) y vértices (V) de un objeto poliédrico

$$V - A + C = 2$$

   Comprueba que se cumple en el poliedro de la figura 10.7.
4. Diseña un algoritmo para buscar fisuras en aristas aladas, que devuelva la poligonal que describe en contorno de cada fisura.
5. Diseña un algoritmo calcular normales de vértice en una estructura de aristas aladas.
6. Diseña un algoritmo que a partir de una poligonal contenida en el plano $Z = 0$ genere una malla por revolución respecto al eje Y.
7. ¿En que influye el número de perfiles usado en una superficie de revolución?
8. La caja englobante de un objeto es el mínimo paralelepípedo alineado con los ejes que lo contiene. Escribe usando seudocódigo un algoritmo calcular caja englobante de una malla.
9. Diseña una estructura para representar mallas de triángulos que permita localizar en tiempo constante cualquier elemento adyacente.
10. En el proceso descrito para generar mallas por revolución ¿Que pasa si el perfil es abierto y acaba y/o empieza en el eje Y?.
11. ¿Se puede generar la superficie de la figura 10.16 como malla de revolución?
12. Diseña un algoritmo para comprobar si una malla representada usando aristas aladas es non-manifol.
13. Genera el grafo de navegación de una representación de mallas de triángulos que consta de dos lista:

    Triángulos que contiene para cada triángulo sus tres vértices y sus tres triángulos vecinos $(v_0, v_1, v_2, t_0, t_1, t_2)$.

    Vértices que contiene las coordenadas del vertice y su normal (x, y, z, n_x, n_y, n_z).
14. Diseña un algoritmo para generar la lista de aristas de la representación anterior, conteniendo para cada aristas los índices de sus dos vértices y sus dos caras
15. Diseña un algoritmo para determinar si la orientación de todas las caras es consistente en la representación anterior.