

Gràfics i Visualització de Dades

T3: Mètodes projectius: ZBuffer

Anna Puig

Índex

3.1. Introducció a ZBuffer

3.2. Pipeline de visualització

3.3. Pipeline de visualització a GL

3.4. Il·luminació usant shaders

3.5. Textures

3.6. Reflexions i Transparències

3.4. Il·luminació usant shaders

En els mètodes projectius el model més utilitzat és el model del Blinn-Phong (veure tema 2b)

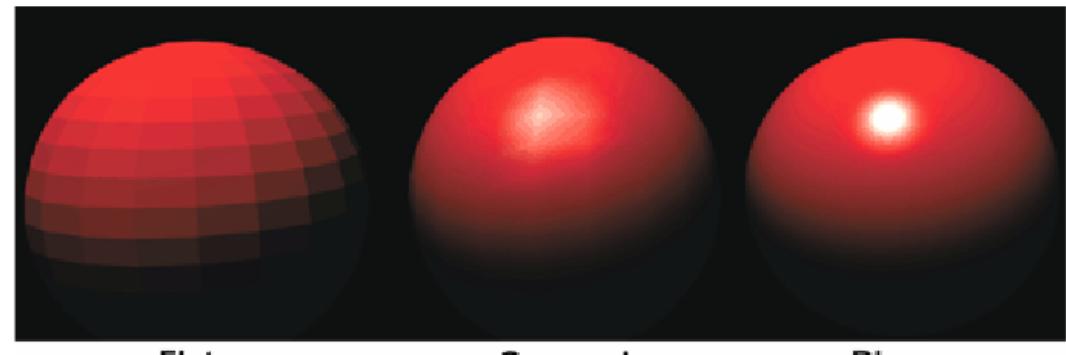
$$I_{total} = I_{a_{global}}K_a + \sum_{i=1}^{numLlums} \frac{1.0}{a_i d_i^2 + b_i d_i + c_i} (I_{d_i} K_d \max(L_i \cdot N, 0.0) + I_{s_i} K_s \max((N \cdot H_i), 0.0)^{\beta}) + I_{a_i} K_a$$

El shading en malles poligonals pot ser calculat de diferents maneres:

- **flat shading**
- **sau (smooth i Gouraud)**
- **Phong shading**

Es necessiten:

- les normals
- les llums
- la càmera
- les propietats dels materials



S'envia tota la informació als shaders

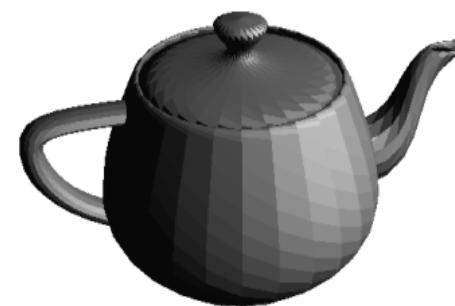
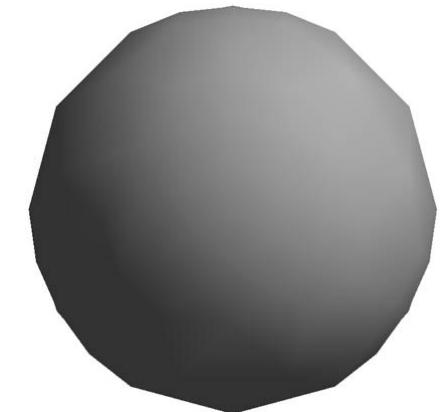
3.4. Il·luminació usant *shaders*

El shading poligonal pot ser:

- **Flat Shading**
- **Suau (smooth i Gouraud)**
- **Phong shading**

Es necesiten:

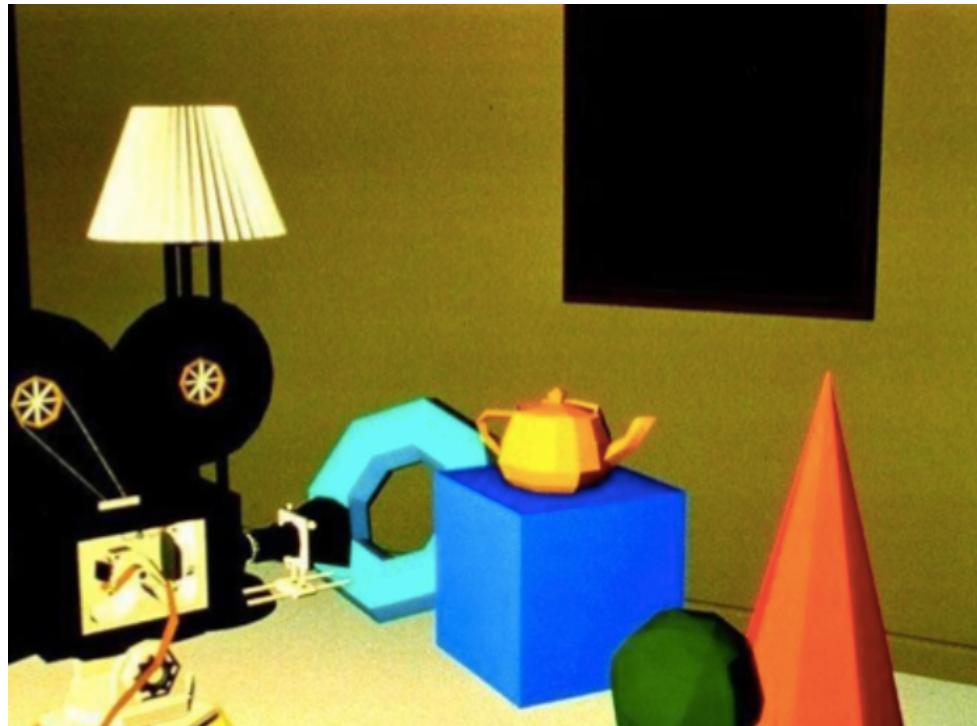
- les normals
- les llums
- les propietats dels materials



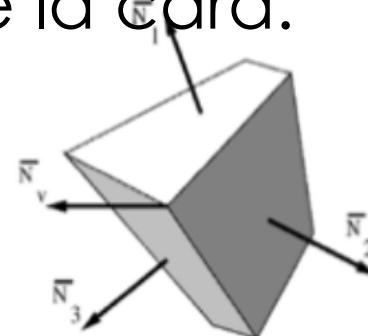
3.4. Il·luminació usant shaders

Flat shading:

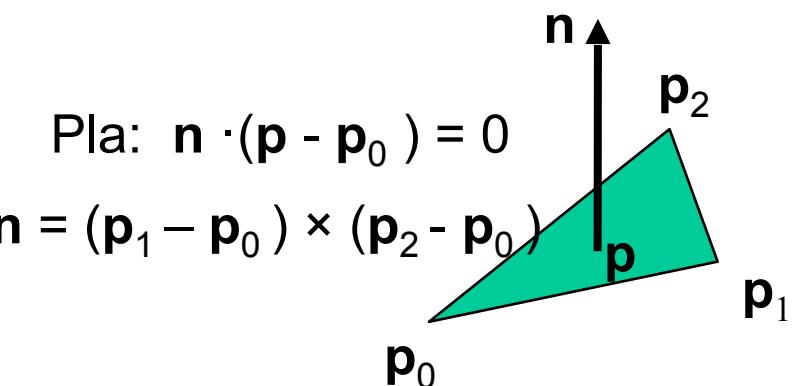
- tots els punts d'una cara es pinten d'un color molt similar, s'usa la normal al pla de la cara.



Mach banding problem



$$\text{Pla: } \mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$$
$$\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)$$

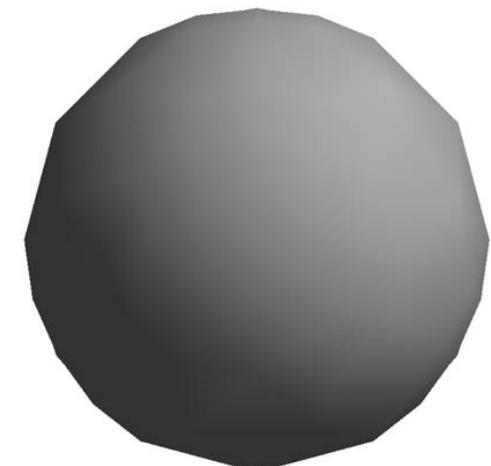
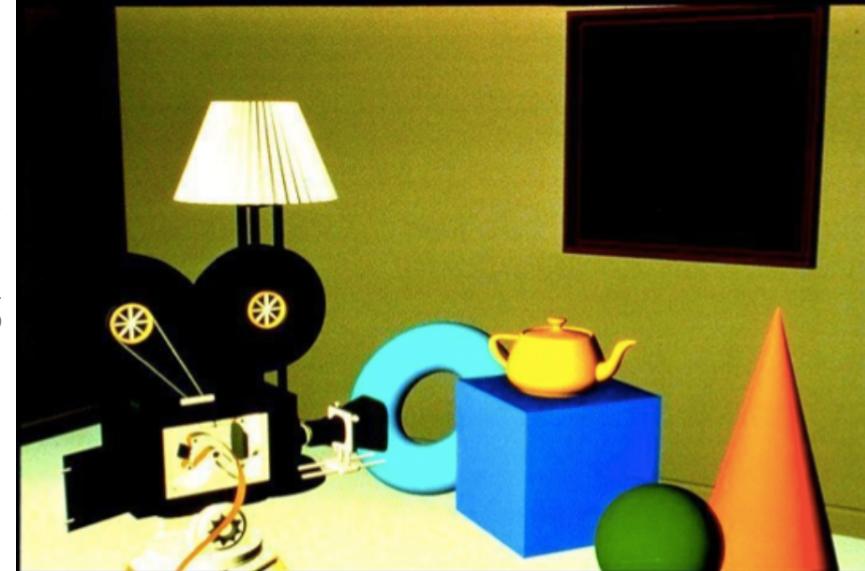


$$\text{normalització } \mathbf{n} = \mathbf{n} / |\mathbf{n}|$$

3.4. Il·luminació usant *shaders*

Smooth shading:

- Es calcula la il·luminació a cada vèrtex i s'interpola el color a cada punt de la cara (segons els seus vèrtexs frontera)
- Cas de superfícies **paramètriques** (com esferes, cilindres, etc.): es pot calcular la directament la normal a cada vèrtex



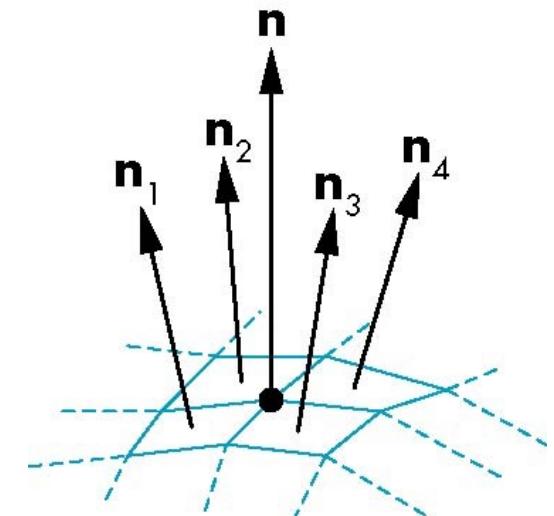
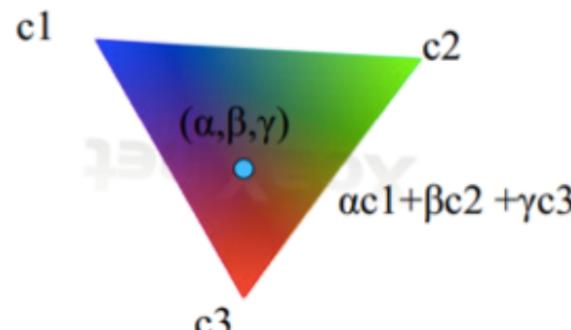
3.4. Il·luminació usant shaders

Gouraud Shading

Usat en models **poligonals** per calcular les normals a cada vèrtex:

1. Es calcula el promig de les normals a cada vèrtex
2. S'aplica el model de Blinn-Phong a cada vèrtex
3. S'interpolen les intensitats de cada vèrtex a cada polígon

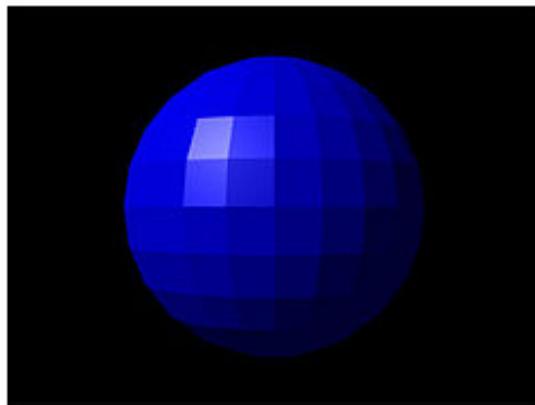
$$\mathbf{n} = (\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4) / |\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4|$$



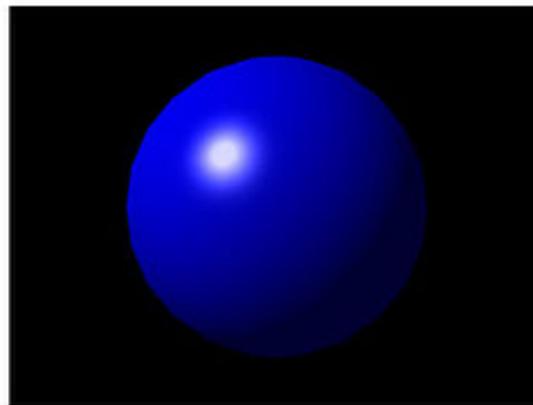
3.4. Il·luminació usant shaders

Phong shading

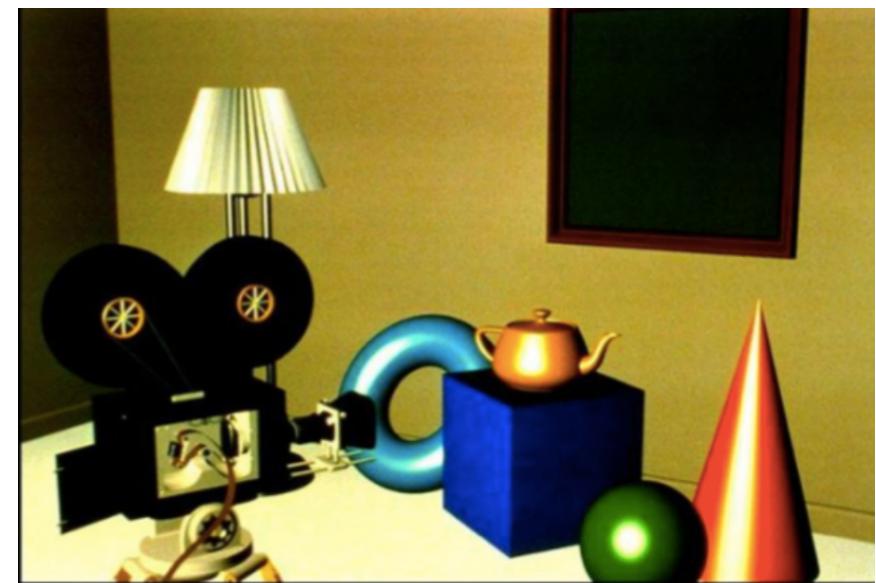
1. Es troben les normals a cada vèrtex
2. S'interpolen les normals en els punts interiors (fragments) del polígon



FLAT SHADING



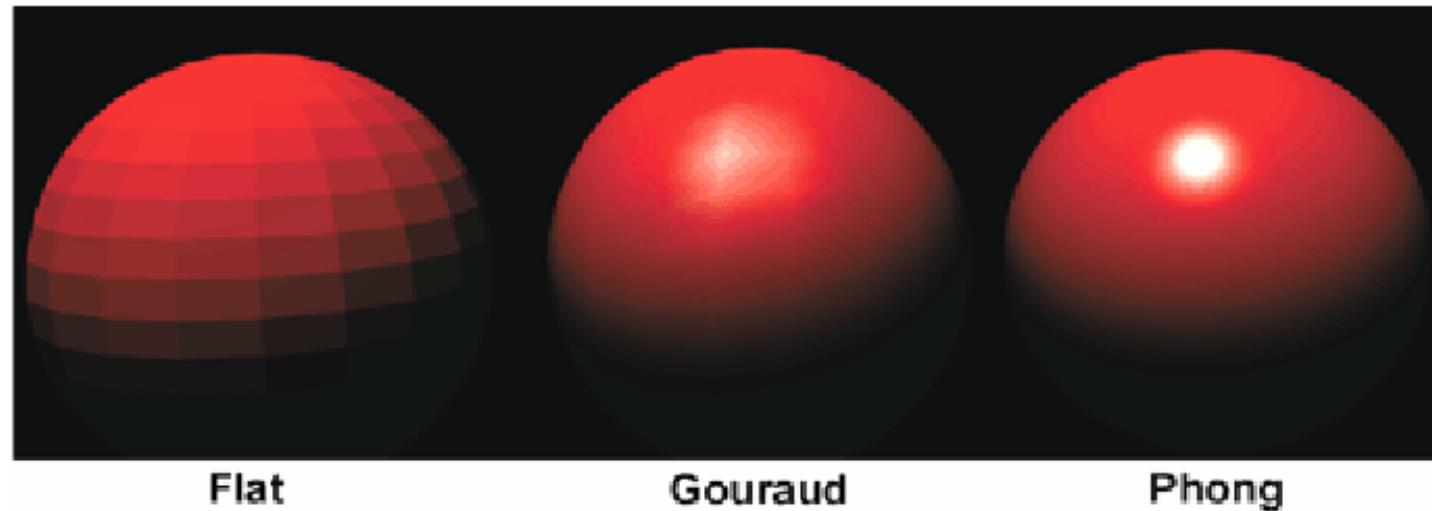
PHONG SHADING



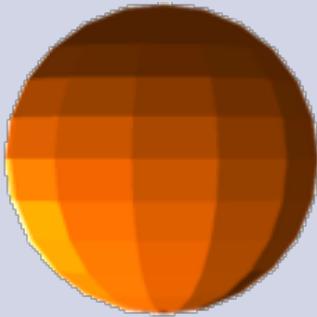
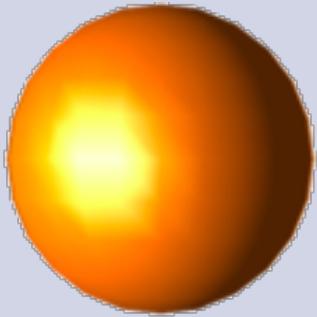
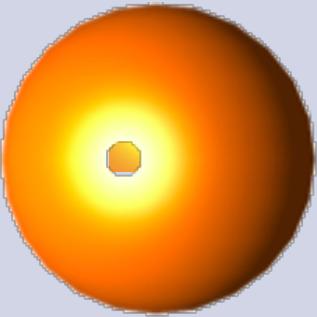
3.4. Il·luminació usant *shaders*

Si els polígons aproximen una superfície amb **curvatures molt altes**, el Phong shading es veu més suau que el Gouraud

El Phong shading implica més càlculs que el Gouraud shading



Comparació de mètodes

	Flat shading	Gouraud Shading	Phong shading
Avantatges	Senzill i ràpid	Encara ràpid No hi ha discontinuïtats de 1er ordre a les arestes	Fa highlights més suaus
Inconvenients	Poc realisme, Mach bands	Problemes amb quadrilàters* Problemes en highlights	Costós de calcular
Resultats			

*

3.4. Il·luminació usant shaders

Vertex shader

```
#version 330

layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vColor;

uniform mat4 model_view;
uniform mat4 projection;

out vec4 color;

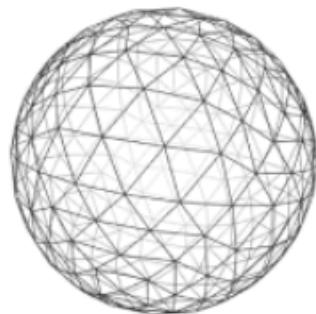
void main()
{
    gl_Position = projection*model_view*vPosition;
    gl_Position = gl_Position/gl_Position.w;
    color = vColor;
}
```

Fragment shader

```
#version 330

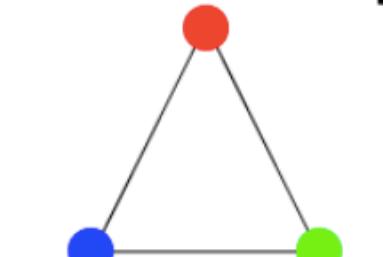
in vec4 color;
out vec4 colorOut;

void main()
{
    colorOut = color;
}
```



Mesh

Attributes →



Vertex Shader



gl_Position

Uniforms



Varyings →

out/in



Fragment Shader



colorOut

Per a calcular Blinn-Phong i el shading corresponent, les dades uniform (comuns a tots els vèrtexs) són les següents:

Posició i orientació de la llum
Color de la llum
Posició de la càmera
Coeficients de material (ambient, difús, especular i brillantor)

3.4. Il·luminació usant shaders

Per a calcular Blinn-Phong i el shading corresponent:

- Quines dades són uniform (comuns a tots els vèrtexs)?
- Quines dades són IN/OUT?
- Càlculs en el Vertex shader o en el Fragment shader?
- Quan es passen a la GPU?

Les dades IN/OUT són les següents:

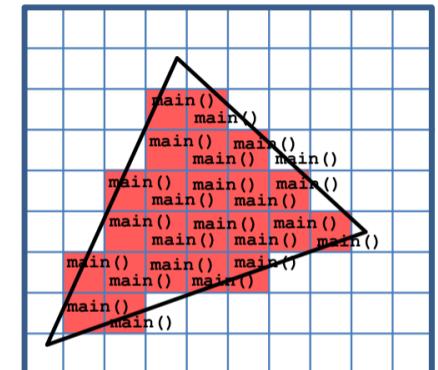
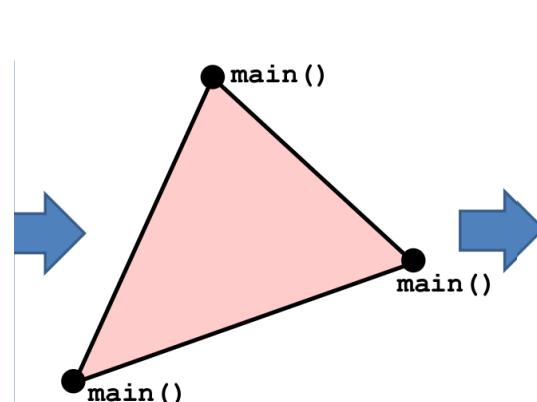
Posició dels vèrtexs (IN)
Normal dels vèrtexs (IN)
Coordenades de textura (IN)
Color dels vèrtexs (IN)
Color final del píxel (OUT)

Els càlculs es realitzen en el Fragment shader, ja que els càlculs de llum són específics per a cada píxel.

Les dades es passen a la GPU en el moment que es dibuixa la malla de triangles. Les dades uniform es passen una sola vegada i es mantenen constants per a tots els vèrtexs i píxels. Les dades IN es passen per a cada vèrtex i es interpolen per a cada píxel en el fragment shader.

$$I_{total} = I_{a_{global}} K_a + \sum_{i=1}^{numLlums} \frac{1.0}{a_i d_i^2 + b_i d_i + c_i} (I_{d_i} K_d \max(L_i \cdot N, 0.0) + I_{s_i} K_s \max((N \cdot H_i), 0.0)^{\beta}) + I_{a_i} K_a$$

`glDrawArrays(GL_TRIANGLES, 0, Index);`



Com calcular les normals?

Representació explícita

t	v_i	v_j	v_k
0	1.0, 1.0, 1.0	-1.0, 1.0, -1.0	-1.0, -1.0, 1.0
1	1.0, 1.0, 1.0	-1.0, -1.0, 1.0	1.0, -1.0, -1.0
2	1.0, 1.0, 1.0	1.0, -1.0, -1.0	-1.0, 1.0, -1.0
3	1.0, -1.0, -1.0	-1.0, -1.0, 1.0	-1.0, 1.0, -1.0

Vèrtexs indexats

v	x	y	z
0	1.0	1.0	1.0
1	-1.0	1.0	-1.0
2	-1.0	-1.0	1.0
3	1.0	-1.0	-1.0

t	i	j	k
0	0	1	2
1	0	2	3
2	0	3	1
3	3	2	1

Adjacència de cares

v	x	y	z	l
0	1.0	1.0	1.0	0
1	-1.0	1.0	-1.0	0
2	-1.0	-1.0	1.0	0
3	1.0	-1.0	-1.0	1

t	i	j	k	n_0	n_1	n_2
0	0	1	2	3	1	2
1	0	2	3	3	2	0
2	0	3	1	3	0	1
3	3	2	1	0	2	1

Winged edge

