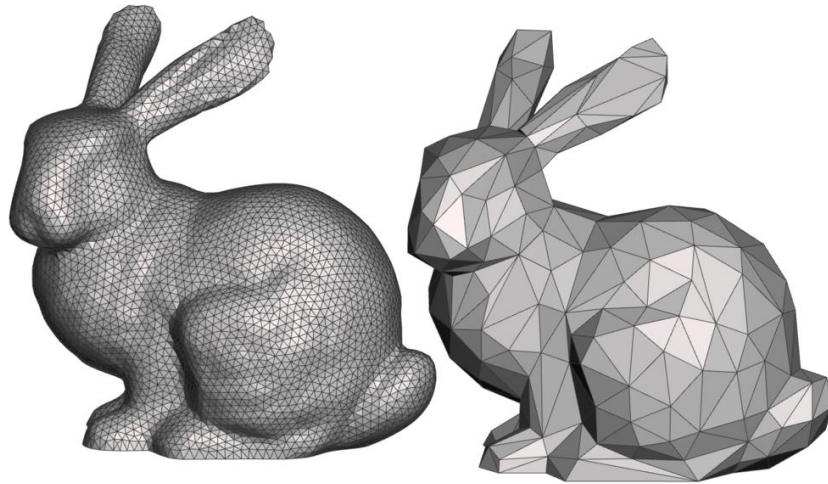


INDI / Classe 1.3



Classe 1.3: Contingut

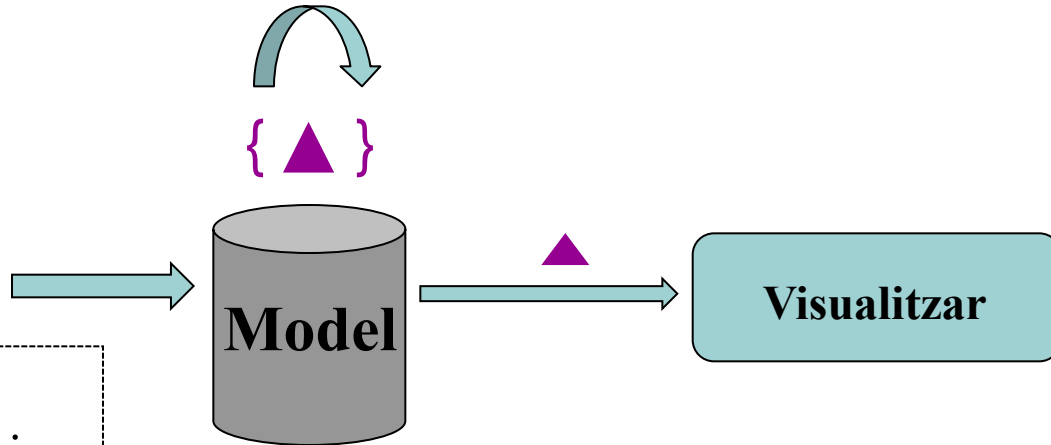
- Introducció a la Informàtica Gràfica
- Introducció a hardware gràfic de sortida
- **Models geomètrics**
 - **Un objecte**
 - Un conjunt d'objectes (escena)

Bibliografia del “Llibre en CD” els temes:

- Geometria2D i 3D.
- Representació d'objectes geomètrics

En el marc d'IDI...

Dades

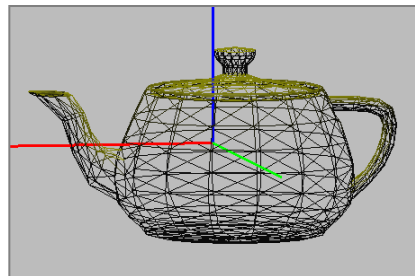
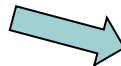


Sòlids

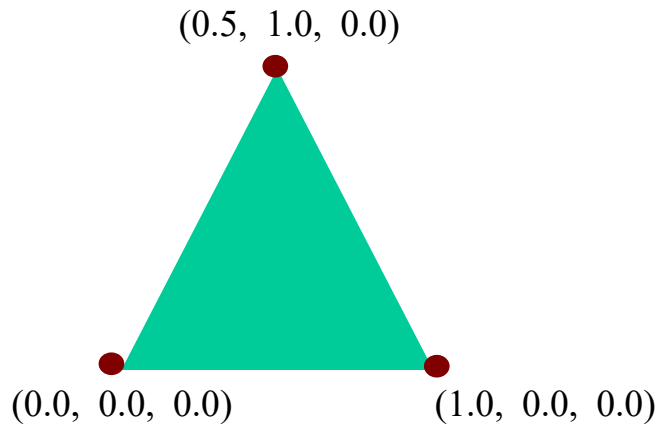
- Limitats per superfície
- Moviments no modifiquen forma

Models Geomètrics:

- Model de Fronteres
aproximació per cares planes => triangles
- Procedural, CSG, octrees,...



Models Geomètrics (intro)

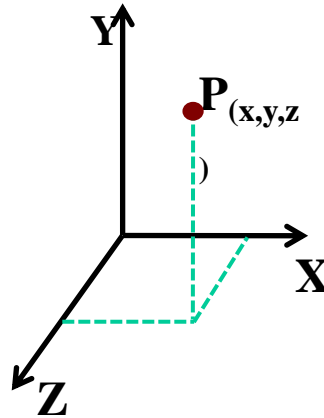


- ***Com guardem un triangle?***

Desem coordenades dels 3 vèrtex.

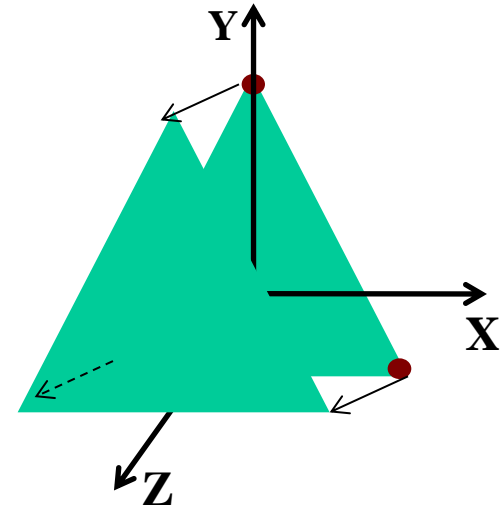
Ún vèrtex en 3D són 3 floats, que en el laboratori es representaran com a:

`glm::vec3`



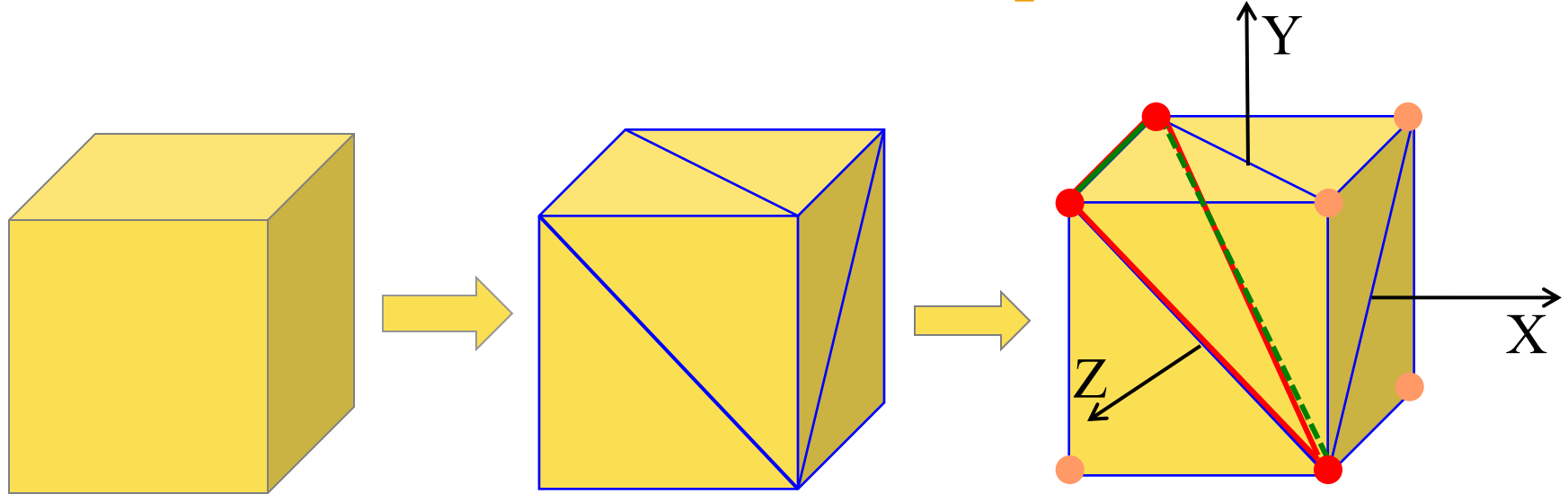
IMPORTANT !

Requerim un sistema de coordenades de referencia (SC). En el marc de l'assignatura utilitzarem SC cartesià orientat segons la mà dreta. (right-handed coordinate system)



Si modifiquem les coordenades, modifiquem la posició del triangle. Les transformacions geomètriques ens ajudaran a fer això.

Model Fronteres: Exemple Cub



Cal desar:

- Geometria
- Topologia (relació de connectivitat/adjacència entre els vèrtexs per formar triangles)

Solució *naïve*: taula de 8 vèrtex

Problemes:

1.- l'ordre determina la topologia.

3.- No es possible definir el cub només amb la llista de 8 vèrtex

Vèrtexs

x	y	z
-1	1	-1
-1	1	1
1	-1	1
1	-1	-1
1	1	-1
1	1	1
-1	-1	-1
-1	-1	1

`glm::vec3 vertex[8];`

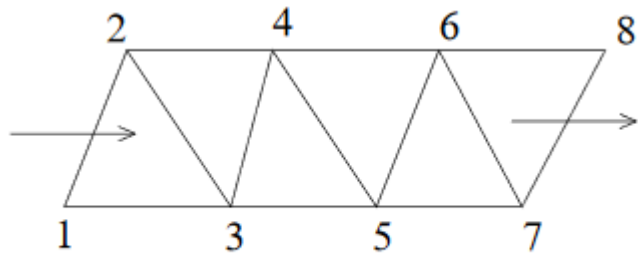


Figure 1: A Triangle Strip

Solució *triangle-strip*:
taula de 14 vèrtex

Problemes:

1.- l'ordre determina la topologia.

2.- Repetim algun vèrtex.

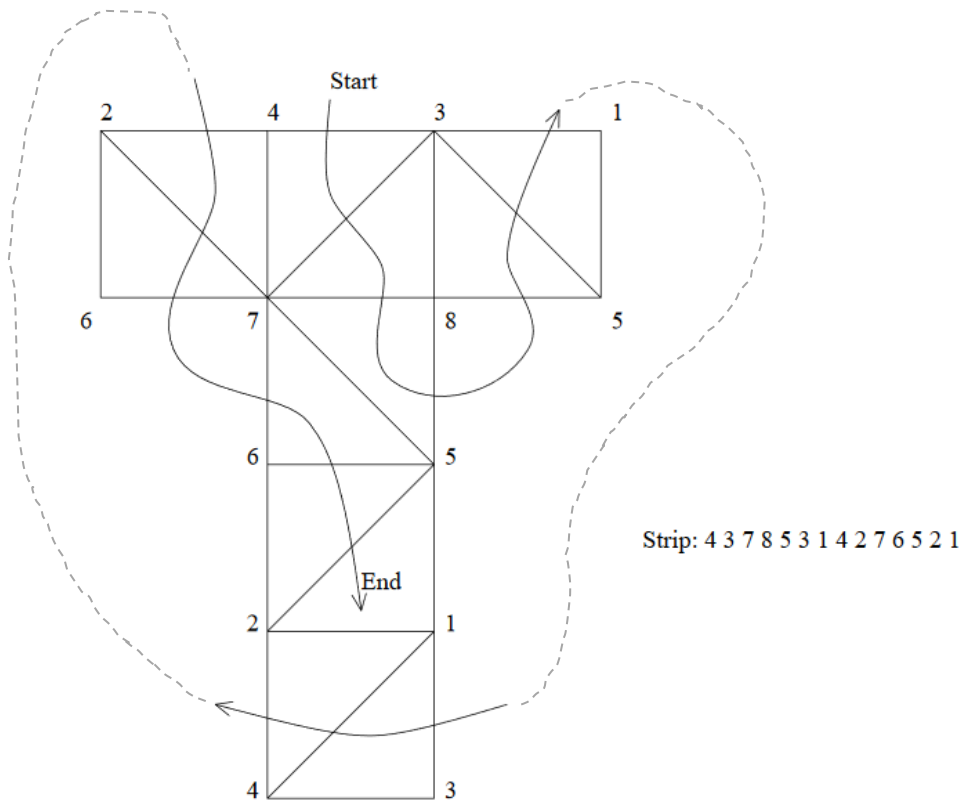


Figure 2: Triangulating a cube for one sequential strip.

```
glm::vec3 vertex[14];
```

Strip: 4 3 7 8 5 3 1 4 2 7 6 5 2 1

Optimizing Triangle Strips for Fast Rendering

Francine Evans

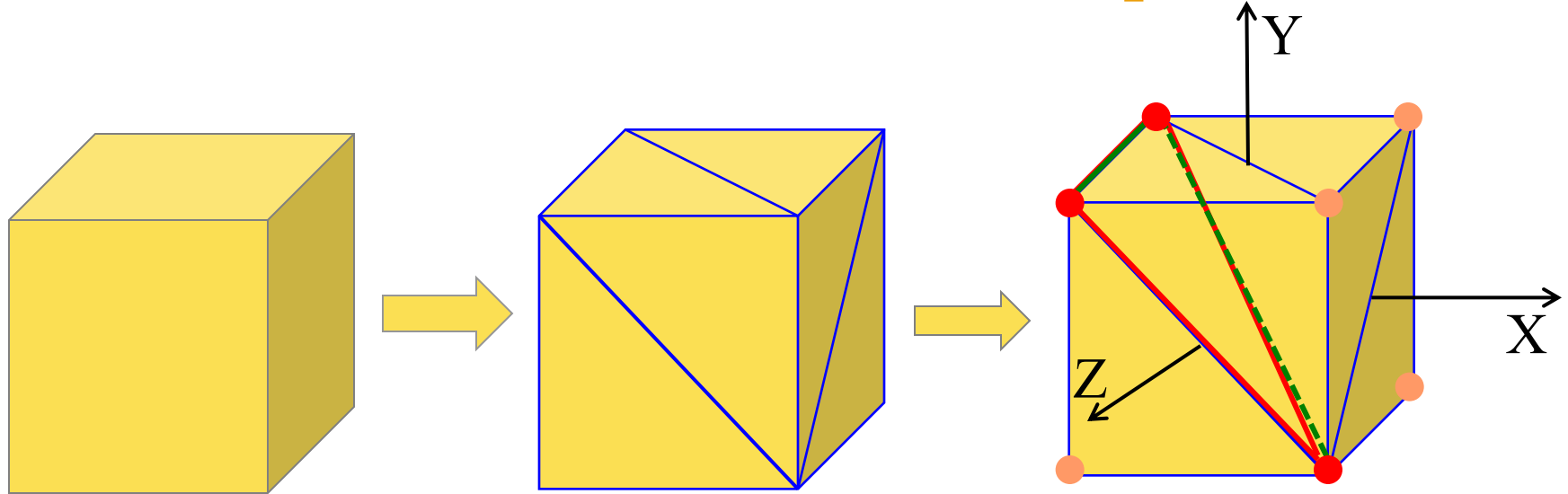
Steven Skiena

Amitabh Varshney

State University of New York at Stony Brook

INDI Q2 2020-2021

Model Fronteres: Model implícit



Solució implícita:

Desem per cada triangle
les coordenades dels tres
vèrtex.

Simple i efectiu.

Problema: Vèrtexs
repetits ☹️

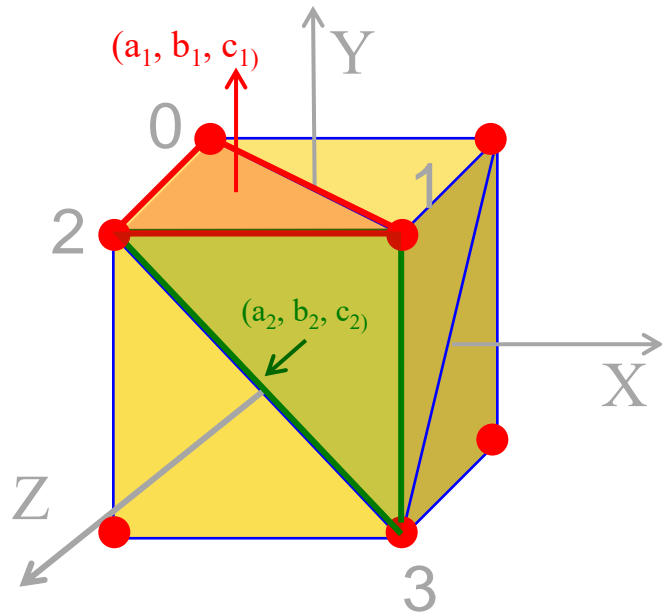
Vèrtexs

x	y	z
-1	+1	-1
-1	+1	+1
+1	+1	+1
...



```
glm::vec3 vertex[3*6*2];
```

Model Fronteres: Model explícit



Cares

normal	Id Vèrtexs
a_1, b_1, c_1	0,2,1
a_2, b_2, c_2	1,2,3
...	...

```
glm::uvec3 cares[6*2];  
glm::uvec3 normals[6*2];
```

Vèrtexs

x	y	z
-1	1	-1
-1	1	1
1	1	1
1	-1	1
1	-1	-1
1	1	-1
-1	-1	-1
-1	-1	1

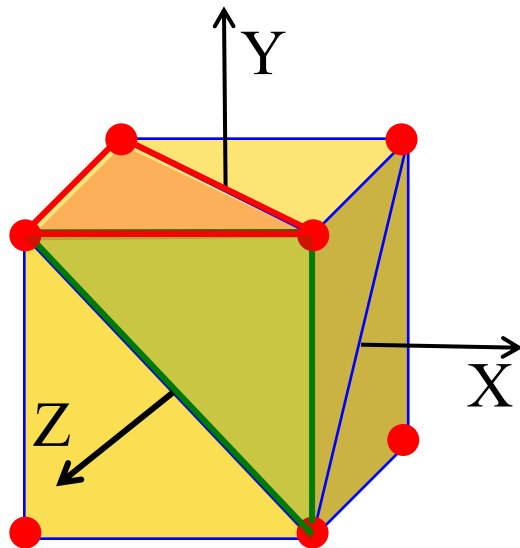
```
glm::vec3 vertex[8];
```

Solució *explícita*:

- Desem els vèrtex sense repeticions a una taula.
- Desem les cares amb els índexs dels vèrtex a una altra taula.
Afegim també les normals pels càlculs d'il·luminació.
- No hi ha redundància

Problema: Més complexitat

Model Fronteres: Opcions de generació



Vèrtexs

x	y	z
-1	1	-1
-1	1	1
1	1	1
-1	1	1
1	-1	1
1	1	1
.	.	.
.	.	.

Topologia implícita

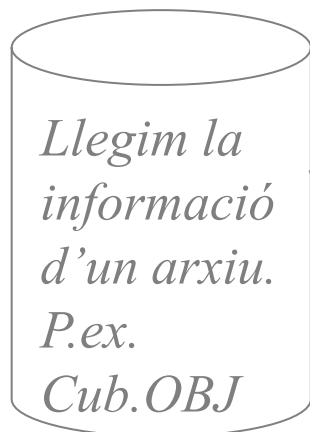
Cares

normal	Id Vèrtexs
a_1, b_1, c_1	0,2,1
a_2, b_2, c_2	1,2,3
...	...

Vèrtexs

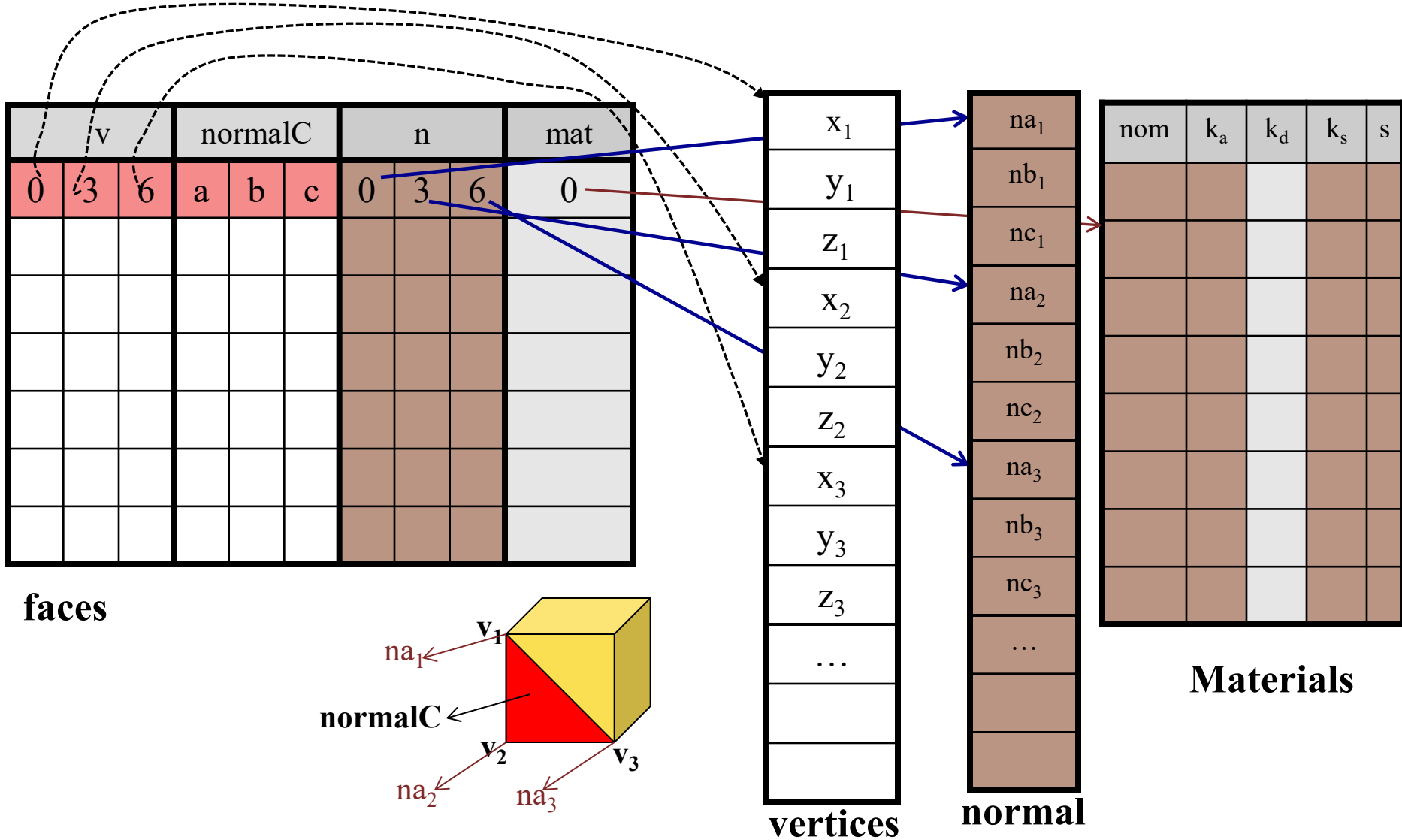
x	y	z
-1	1	-1
-1	1	1
1	1	1
1	-1	1
1	-1	-1
1	1	-1
-1	-1	-1
-1	-1	1

Topologia explícita



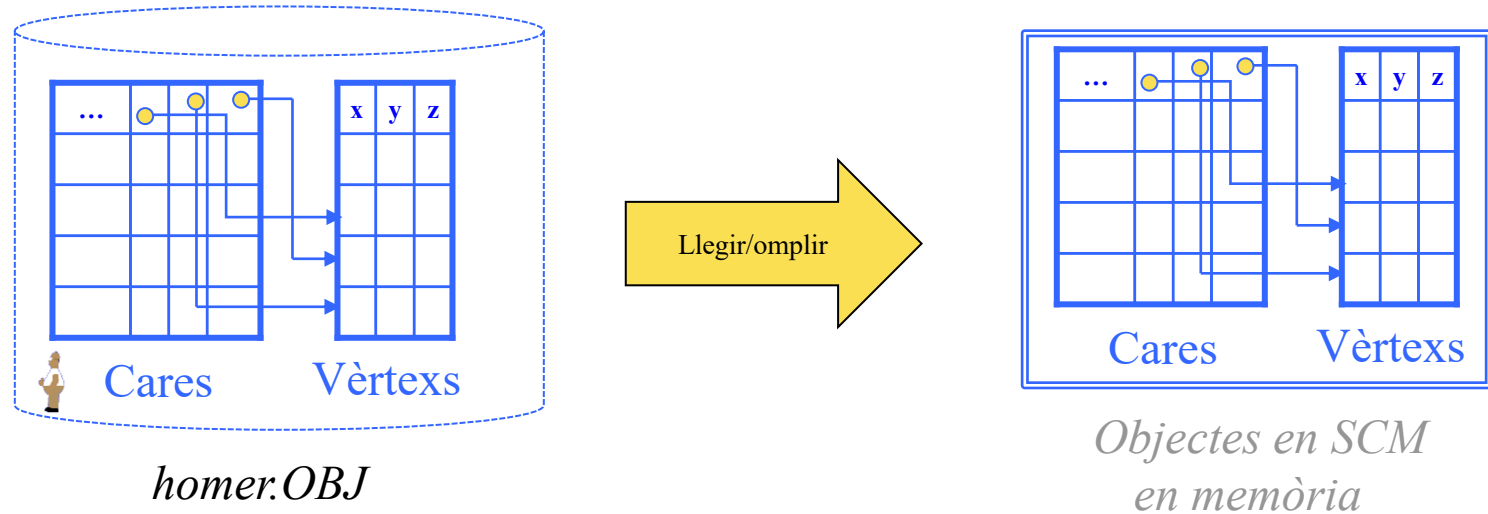
Generem valors en el codi (algorítmicament)

Exemple: Laboratori



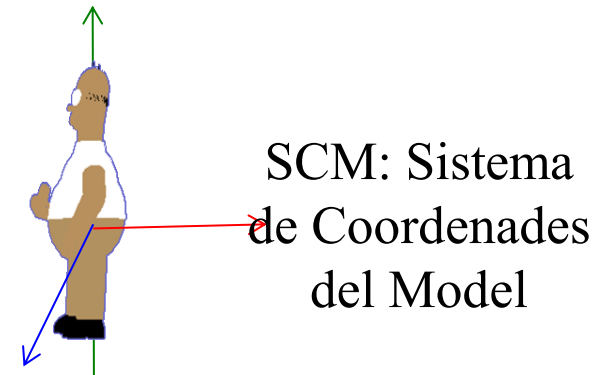
En el marc de les classes de teoria considerem:

- A) Disposem de models d'objectes ja creats i desats en un arxiu. Els triangles del model són respecte a un SC definit pel creador, i li direm SC de Mode o SCM.
- B) Assumirem que tenim un mètode per llegir i omplir una estructura de dades explícita (cares+vèrtex) per representar la superfície.



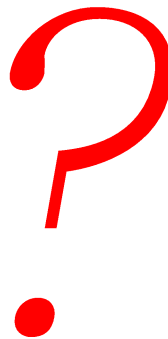
```
10499 f 1 2 3
10500 f 1 3 4
10501 f 2 5 6
10502 f 2 6 3
10503 f 5 7 8
10504 f 5 8 6
10505 f 7 9 10
10506 f 7 10 8
10507 f 9 11 12
```

```
1 # Blender v2.66 (sub 0) OBJ File: '6781.blend'
2 # www.blender.org
3 mtllib HomerProves.mtl
4 o homer
5 v -0.465557 0.169774 0.358141
6 v -0.462597 0.160639 0.365364
7 v -0.440276 0.161682 0.362064
8 v -0.440372 0.170949 0.354419
9 v -0.459846 0.149863 0.369920
10 v -0.440599 0.150763 0.367073
11 v -0.457495 0.138188 0.371498
```



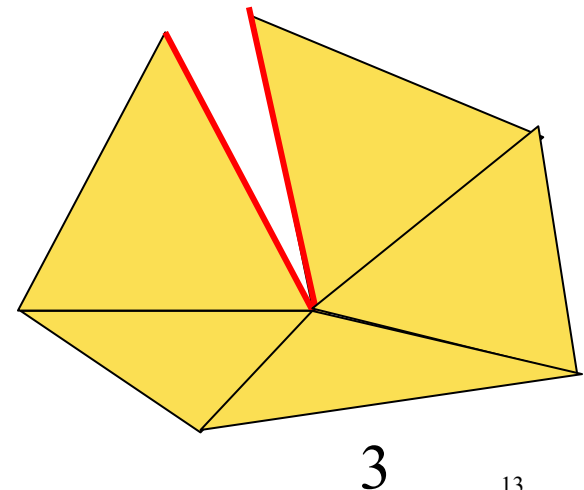
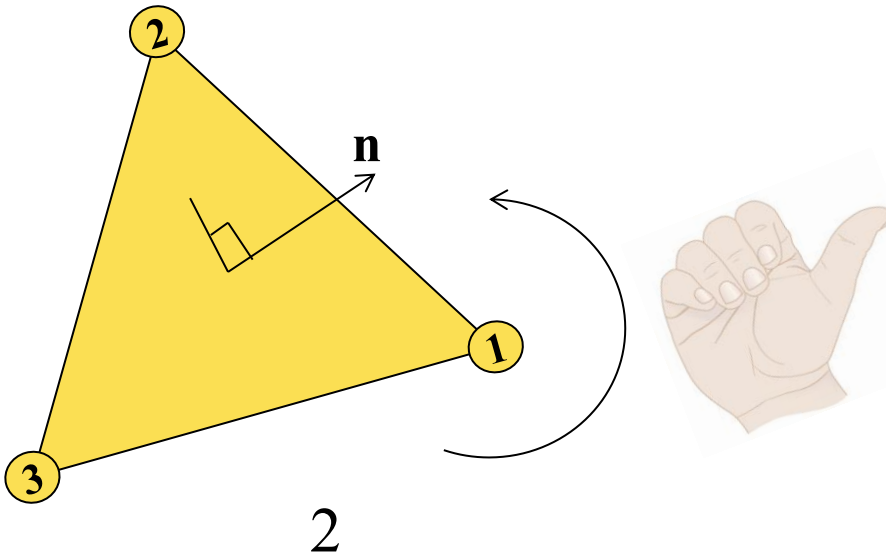
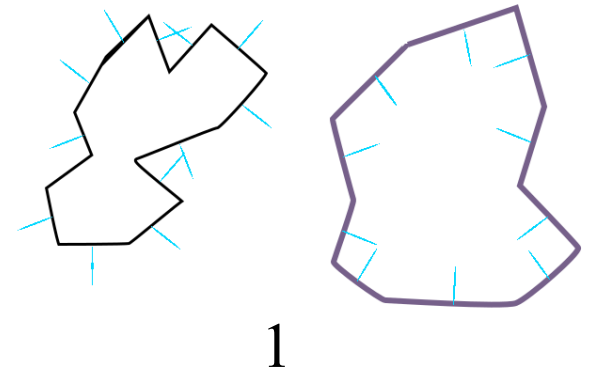
Com podem saber si el model és vàlid?

- O sigui que tenim una superfície que separa interior de l'exterior?



Model Fronteres: propietat de ser vàlid

1. Cares “orientades” coherentment (normals apuntant a dins o a fora, però totes igual).
2. Ordenació vèrtexs coherent amb l’orientació de les normals (regla mà dreta)
3. Cada aresta separa 2 cares.

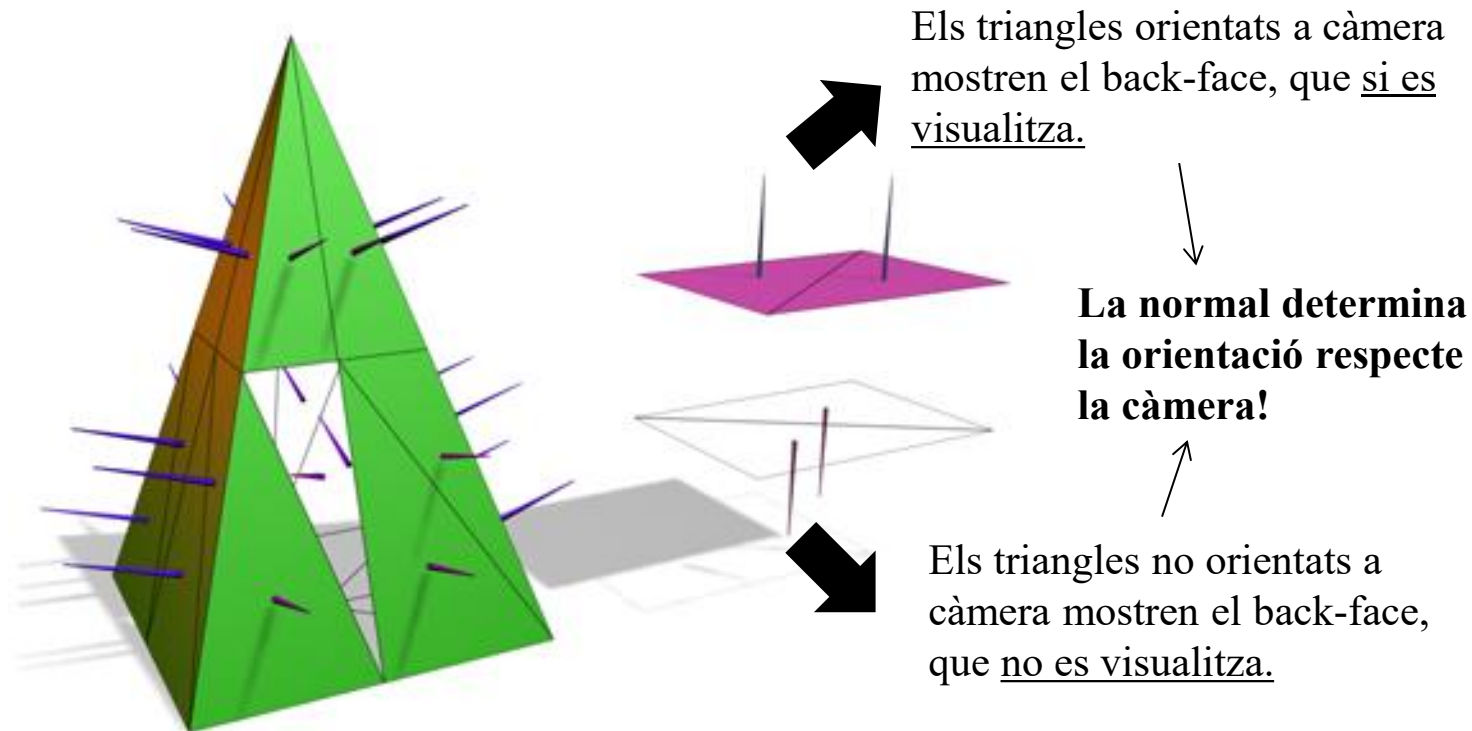


Model Fronteres: Que passa si el model no és vàlid

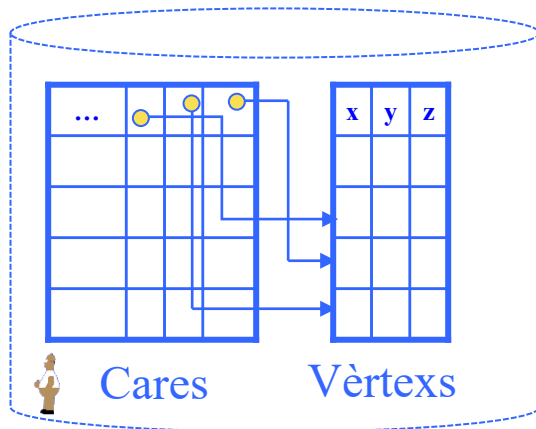
El càlcul de la visualització utilitza les normals.

Els algorismes pressuposen models vàlids.

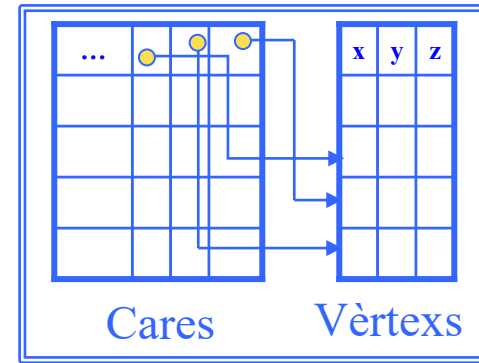
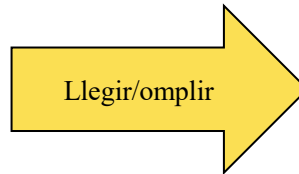
Això implica que el càlcul del color del triangle no serà correcte.



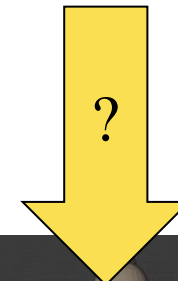
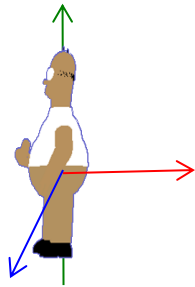
Però...com visualitzem el model?




homer.OBJ

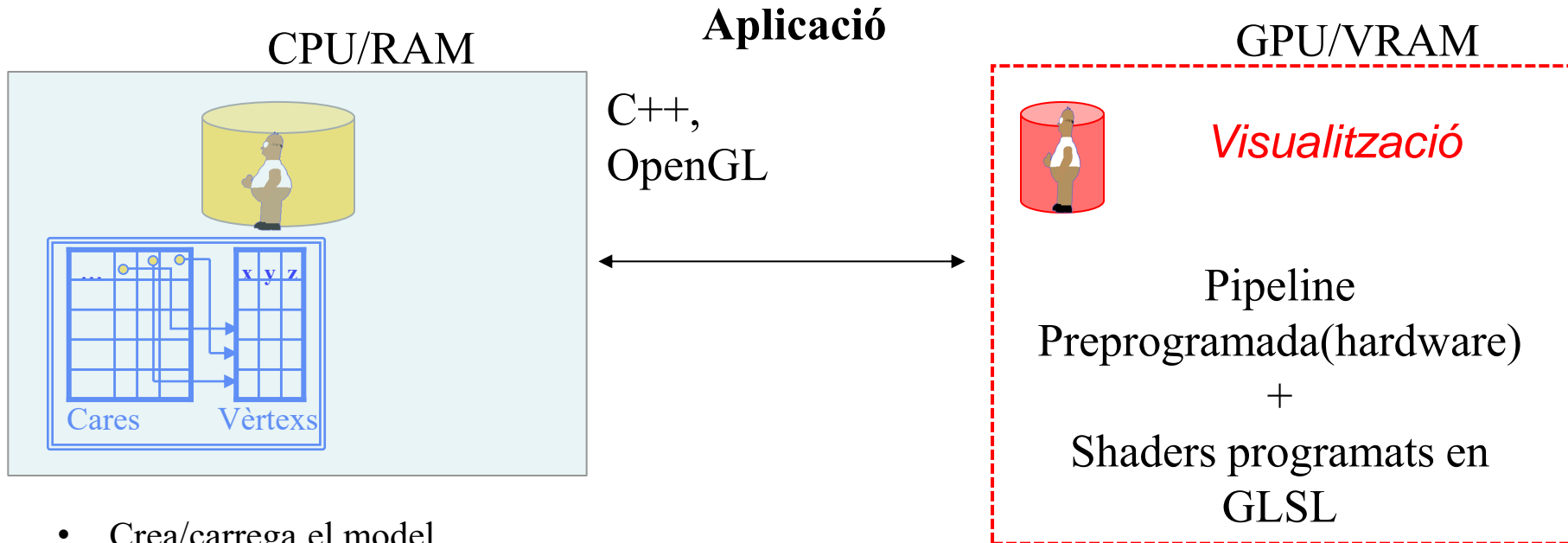


*Objectes en coord. de model
en memòria*

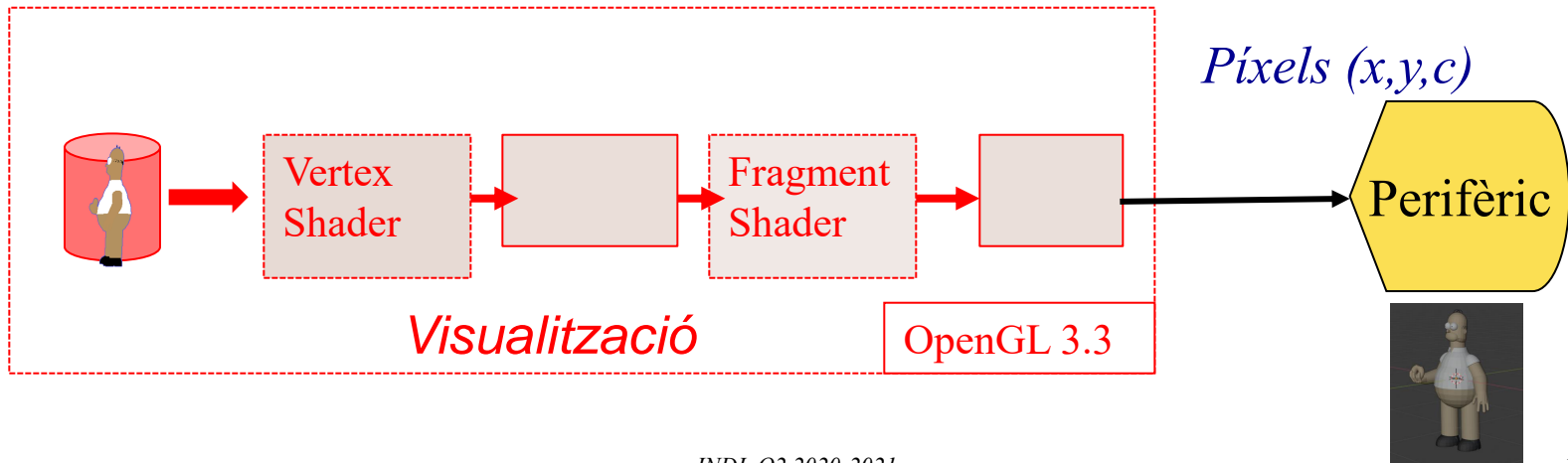


A INDI utilitzarem l' API/llibreria
 en versió 3.3 que ens ajudarà a
visualitzar el model 3D.

Pintar en OpenGL 3.3: “core” mode

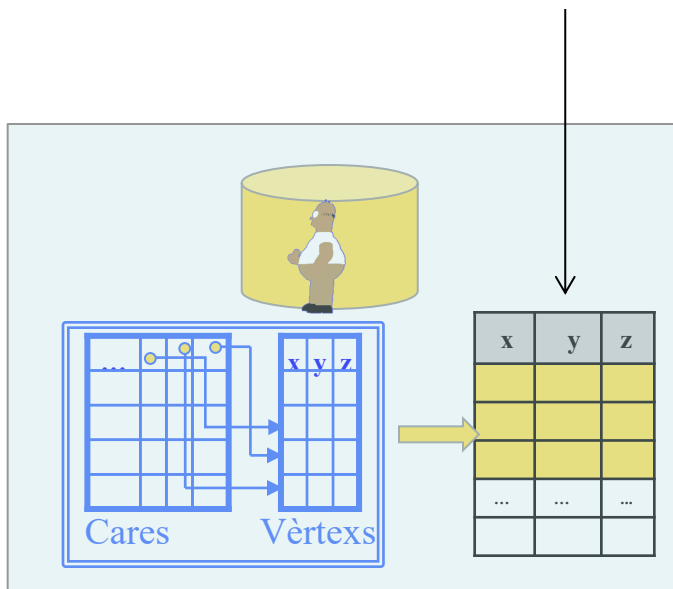


- Crea/carrega el model
- Decideix la posició a l'espai (transformació geomètrica)
- Defineix la interacció



Pintar en OpenGL 3.3: “core” mode

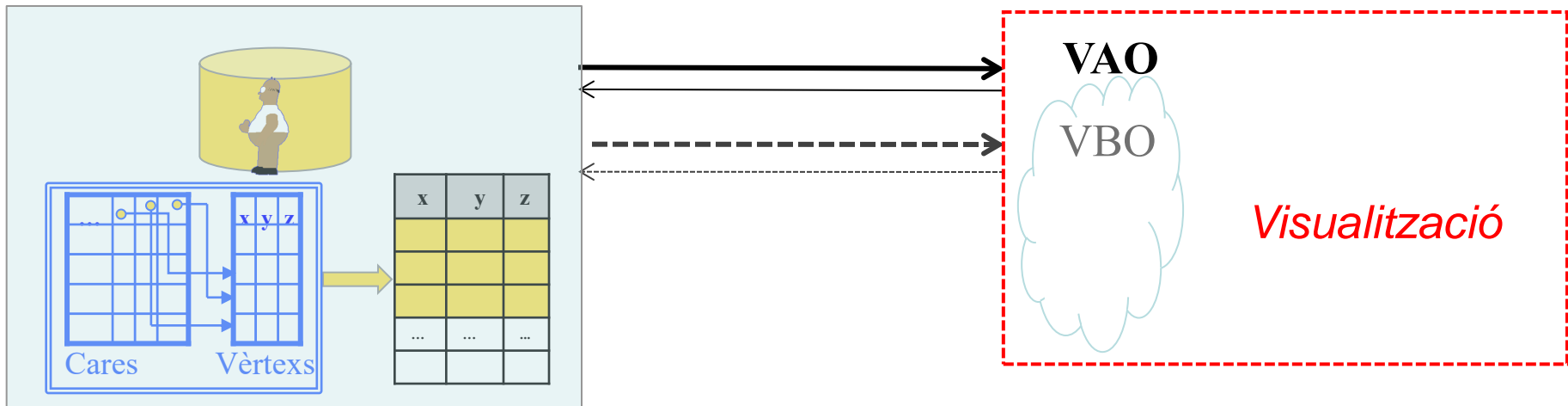
- **Un sol cop** cal enviar/passar el *model/geometria* a la GPU com una llista de triangles amb les coordenades dels vèrtexs i topologia implícita (o altres opcions com “strips”).
- Per tant, si ens cal, a partir del model haurem de crear una estructura auxiliar/temporal amb la informació en aquest format per poder enviar-la a la GPU.



Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL:

- un *VAO* (*Vertex Array Object*) que encapsularà dades del model (objecte).
- un *VBO* (*Vertex Buffer Object*) que guardarà les coordenades dels vèrtexs (potser calen d’altres per normals, colors,...)

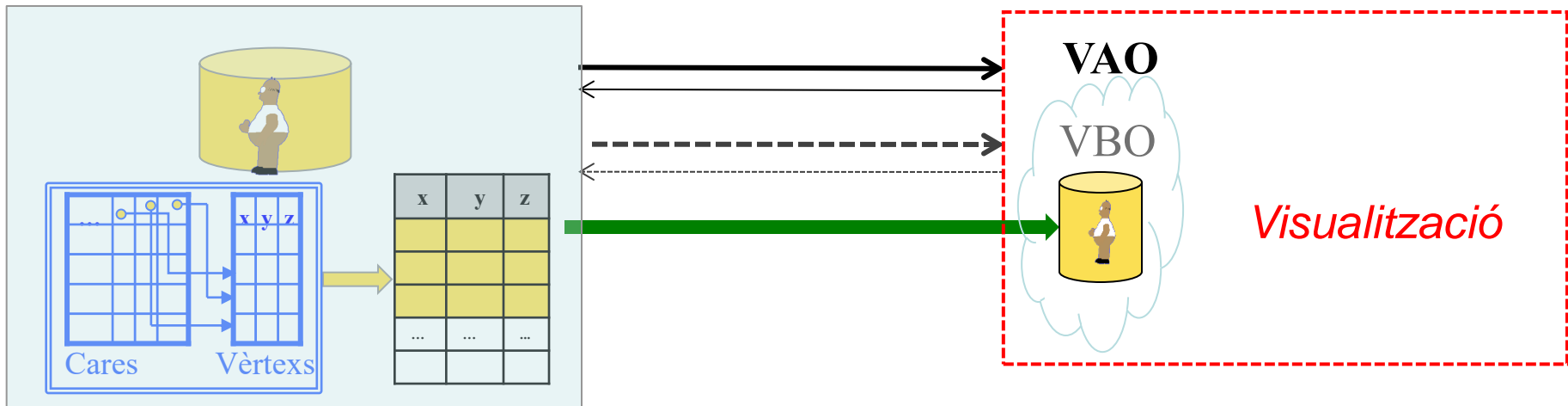


Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL:

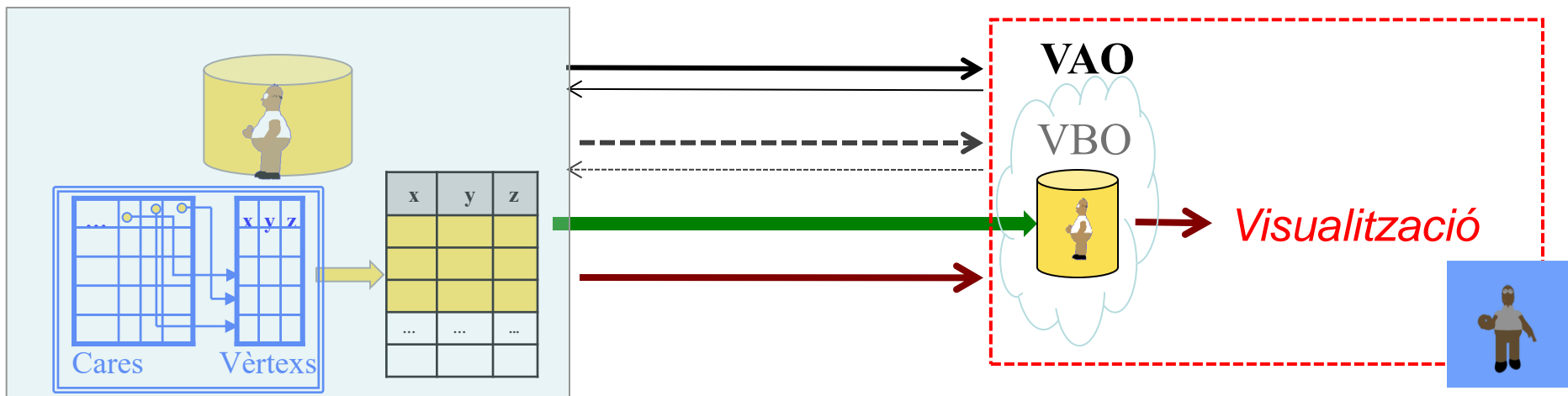
- un *VAO* (*Vertex Array Object*) que encapsularà dades del model (objecte).
- un *VBO* (*Vertex Buffer Object*) que guardarà les coordenades dels vèrtexs (potser calen d'altres per normals, colors,...)

2. Guardar la llista de vèrtexs en el *VBO* (i si cal, color i normal en els seus *VBO*)

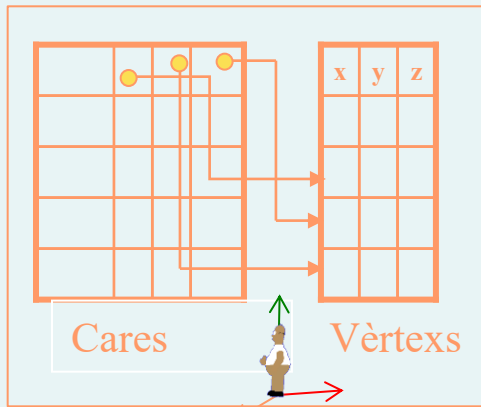


Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL:
 - un *VAO* (*Vertex Array Object*) que encapsularà dades del model (objecte).
 - un *VBO* (*Vertex Buffer Object*) que guardarà les coordenades dels vèrtexs (potser calen d'altres per normals, colors,...)
2. Guardar la llista de vèrtexs en el *VBO*
3. Cada cop que es requereix pintar (acció `pinta_model()` a teoria):
 - a) indicar el *VAO* a pintar: `glBindVertexArray(VAOi)`;
 - b) dir que es pinti: `glDrawArrays(GL_TRIANGLES...)`.

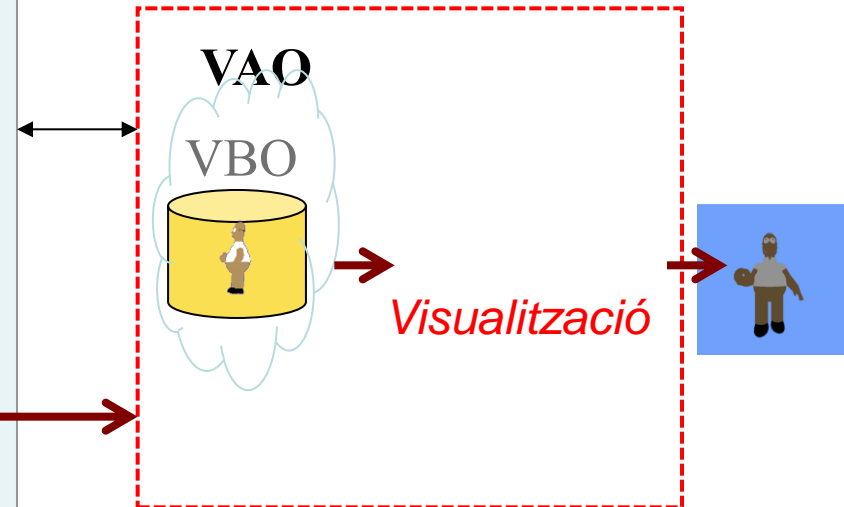


Pintar en OpenGL 3.3: “core” mode



```
// un sol cop  
“crear” i omplir un VAO i VBOi
```

```
/*cada cop que cal refrescar finestra  
activa VAO i glDrawArrays(...)*/  
pinta_model()
```



OpenGL 3.3: Creació del model

```
//-----  
// Creació del Vertex Array Object (VAO) que usarem per pintar  
glGenVertexArrays(1, &VAO1);  
glBindVertexArray(VAO1);
```

```
//-----  
// Creació del VBO de vèrtexs  
// Tres vèrtexs amb X, Y i Z  
glm::vec3 Vertices[3];  
Vertices[0] = glm::vec3(-1.0, -1.0, 0.0);  
Vertices[1] = glm::vec3(1.0, -1.0, 0.0);  
Vertices[2] = glm::vec3(0.0, 1.0, 0.0);
```

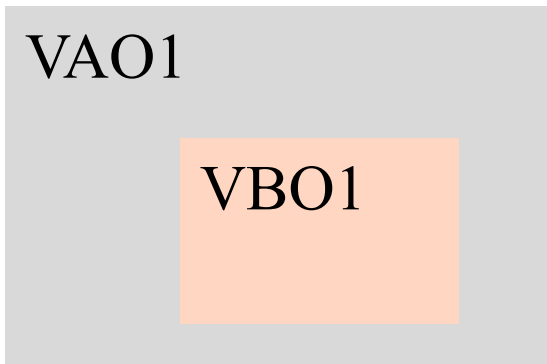
VAO1

VBO1

```
// VBO1 contindrà els vèrtexs  
GLuint VBO1;  
glGenBuffers(1, &VBO1);  
glBindBuffer(GL_ARRAY_BUFFER, VBO1);  
glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);  
// Activem l'atribut que farem servir des del Vertex Shader  
glVertexAttribPointer(vertexLoc, 3, GL_FLOAT, GL_FALSE, 0, 0);  
glEnableVertexAttribArray(vertexLoc);
```

OpenGL 3.3: Visualització del model

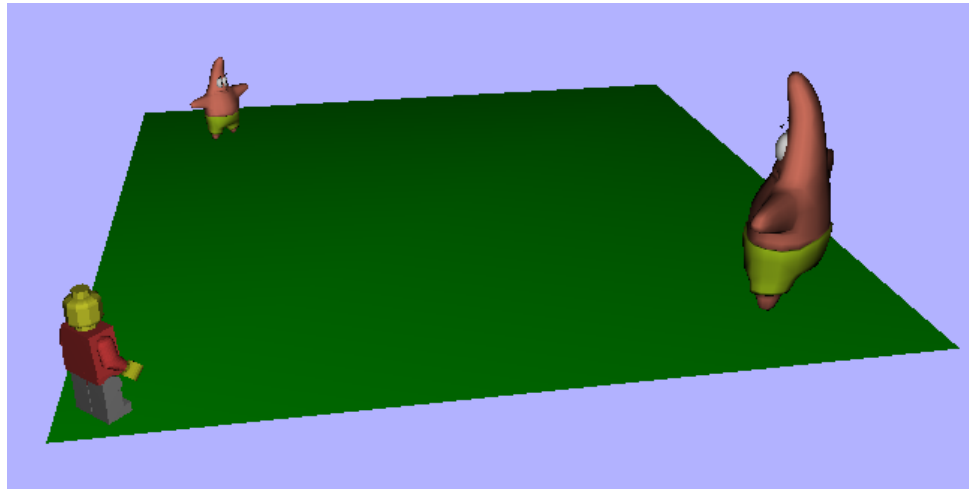
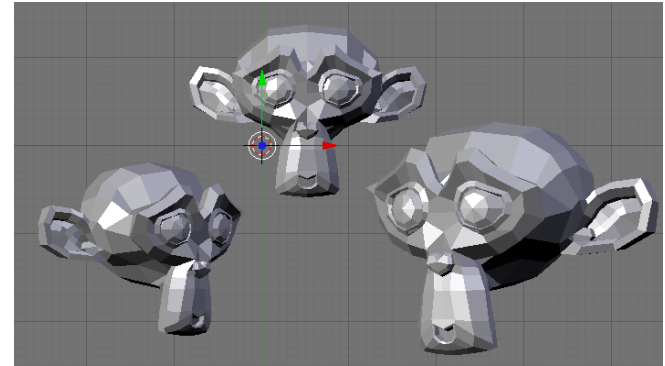
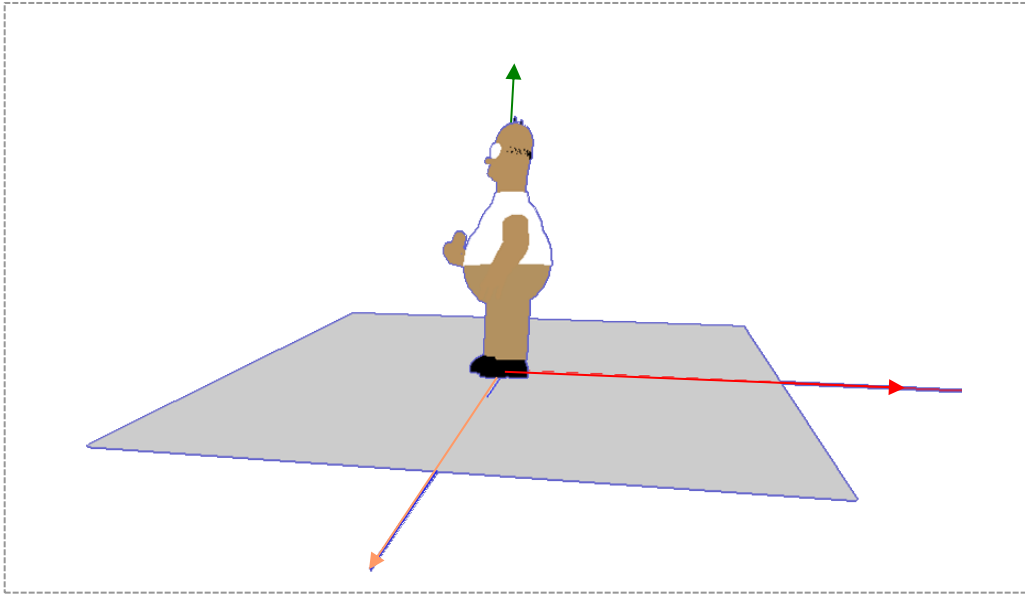
```
//-----  
// Activem el Vertex Array Object 1 (VAO1)  
glBindVertexArray(VAO1);  
// Pintem l'escena  
glDrawArrays(GL_TRIANGLES, 0, 3);
```



Classe 1: Contingut

- Introducció a la Informàtica Gràfica
- **Models geomètrics**
 - Un objecte
 - **Un conjunt d'objectes (escena)**

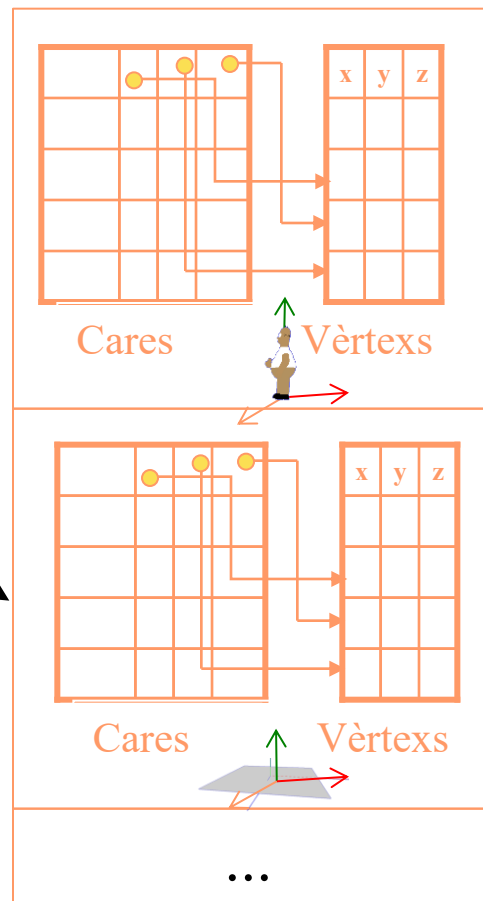
Com guardar i pintar “escenes”?



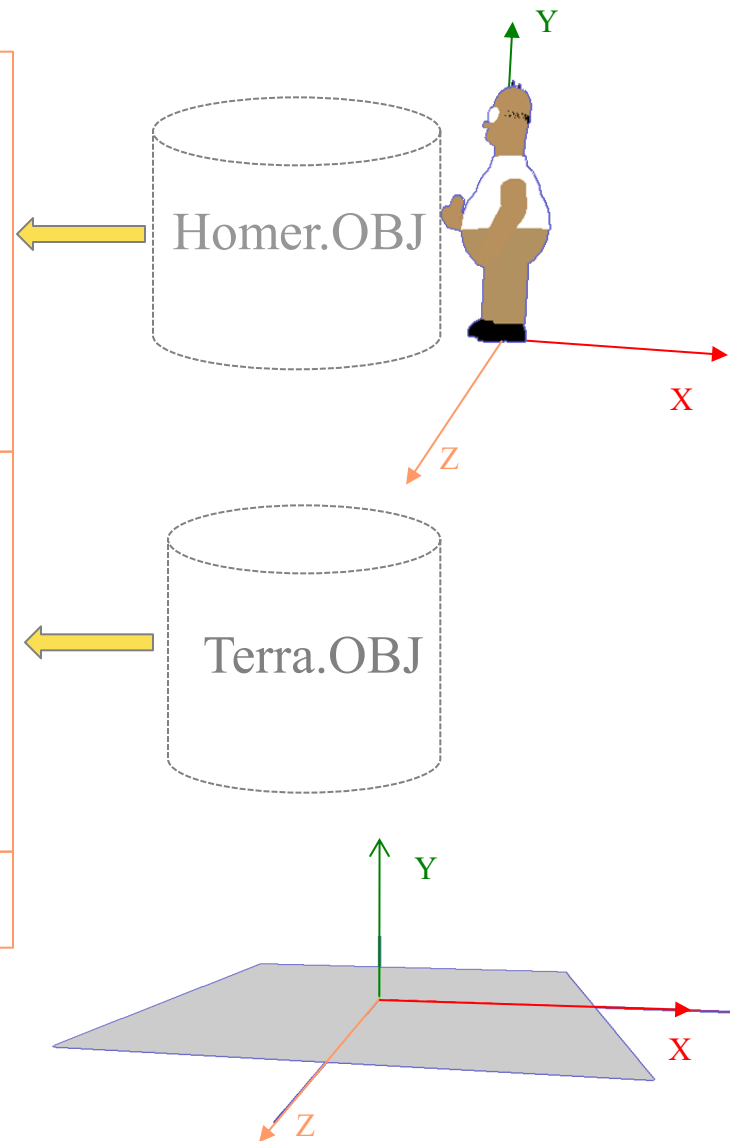
Escena : {objectes}

nom	model
Homer		
Terra		

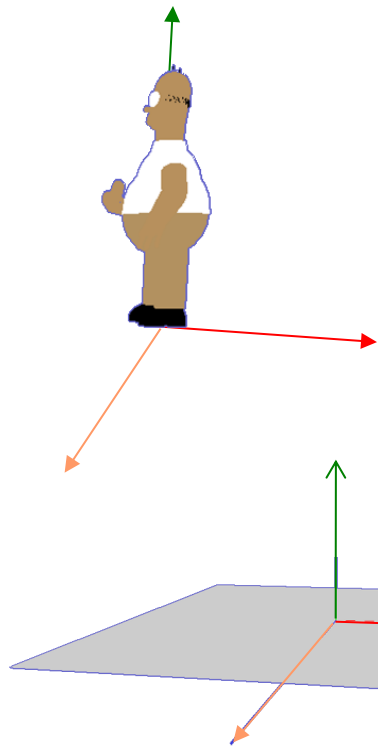
Objectes



Models



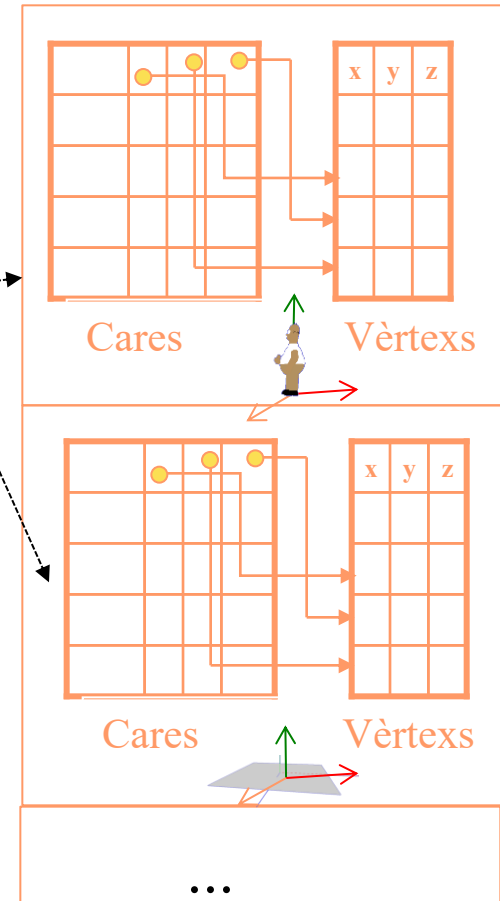
Escena : {objectes} ... com visualitzar?



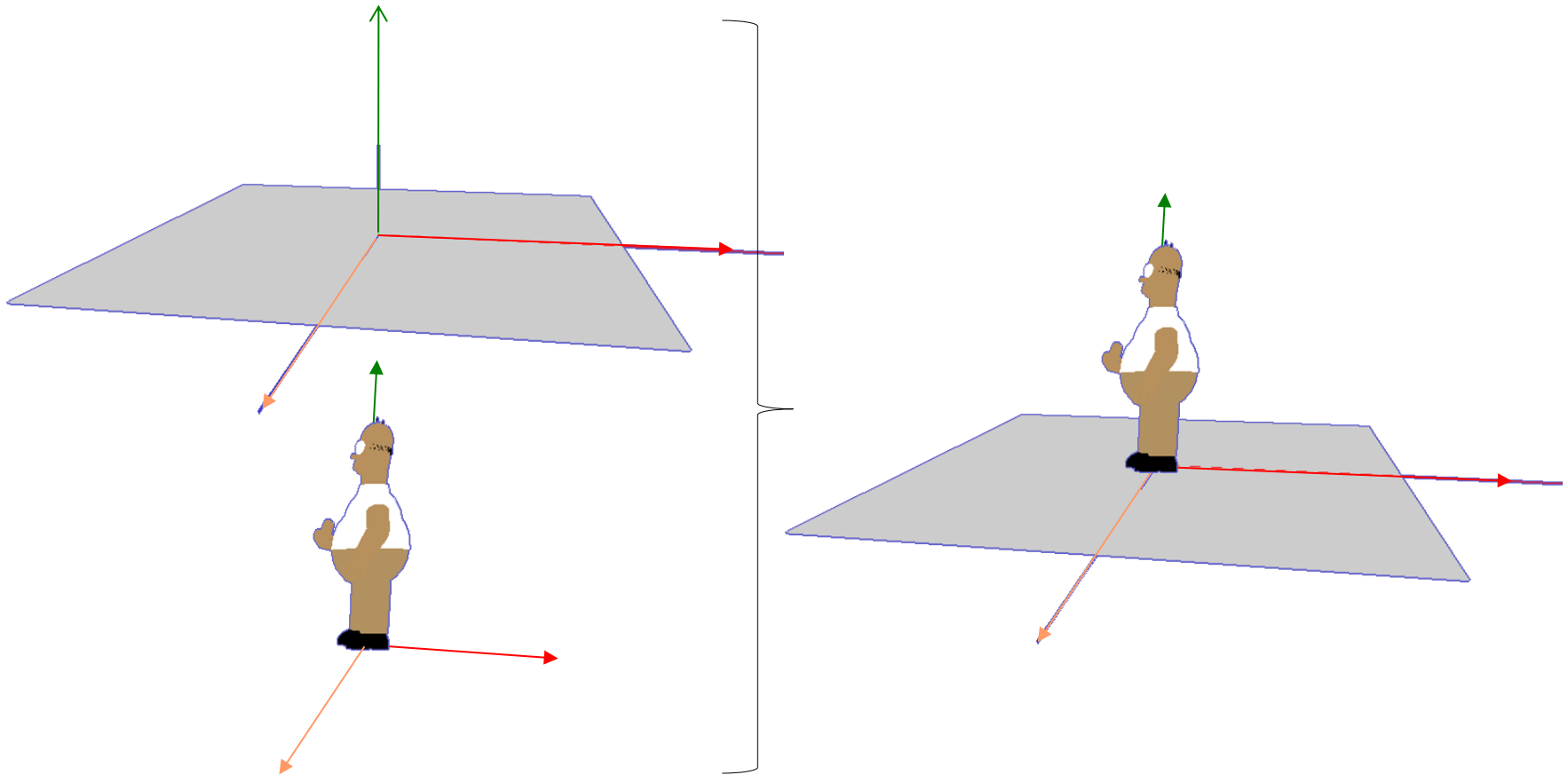
nom	model
Homer		
Terra		

```
// un sol cop  
per cada model  
crear i omplir VAOi i VBOj  
fper
```

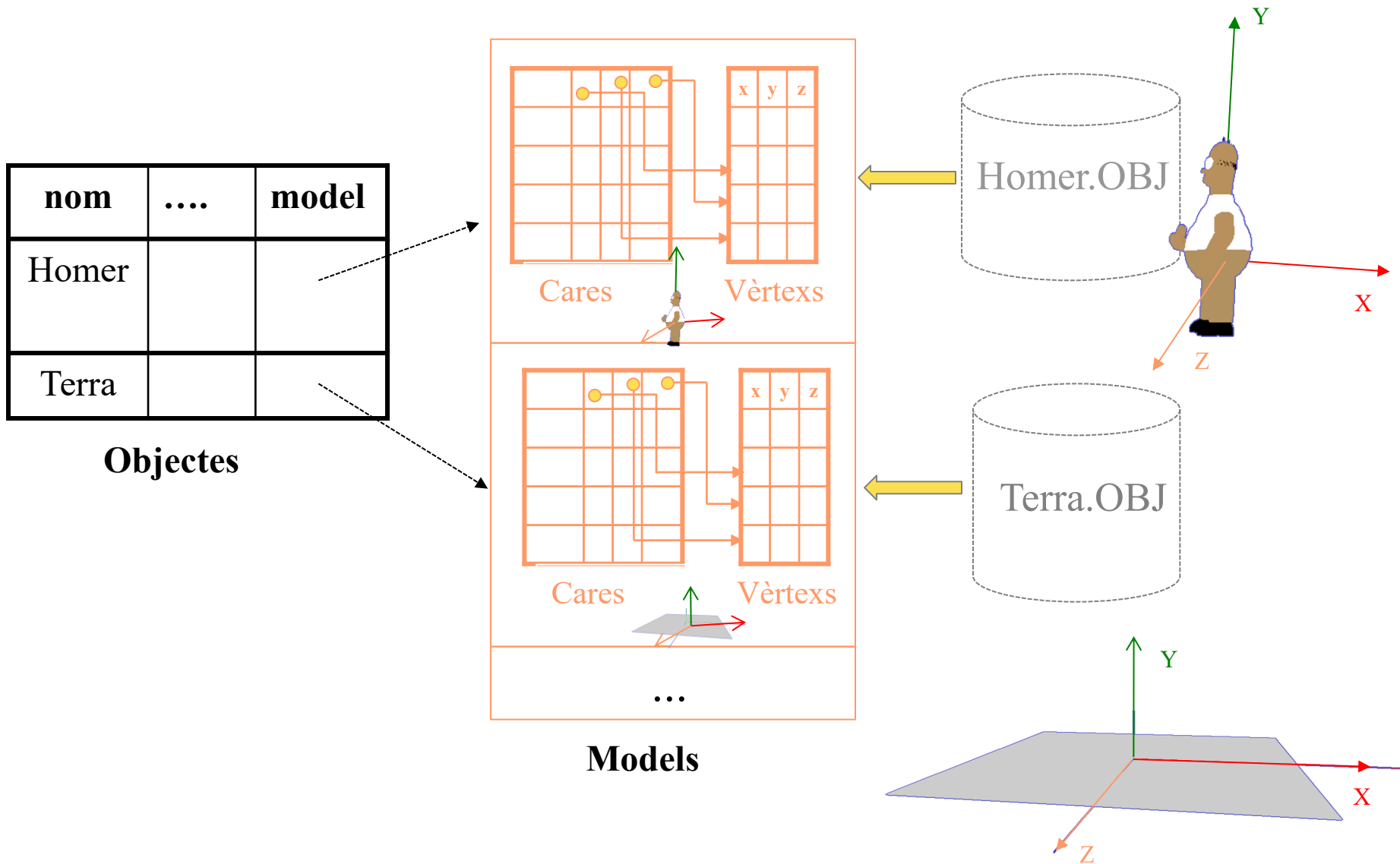
```
//cada cop que cal refrescar finestra  
per cada objecte  
/*pinta el seu model: activa VAO  
i glDrawArrays(...)*/  
pinta_modeli()  
fper
```



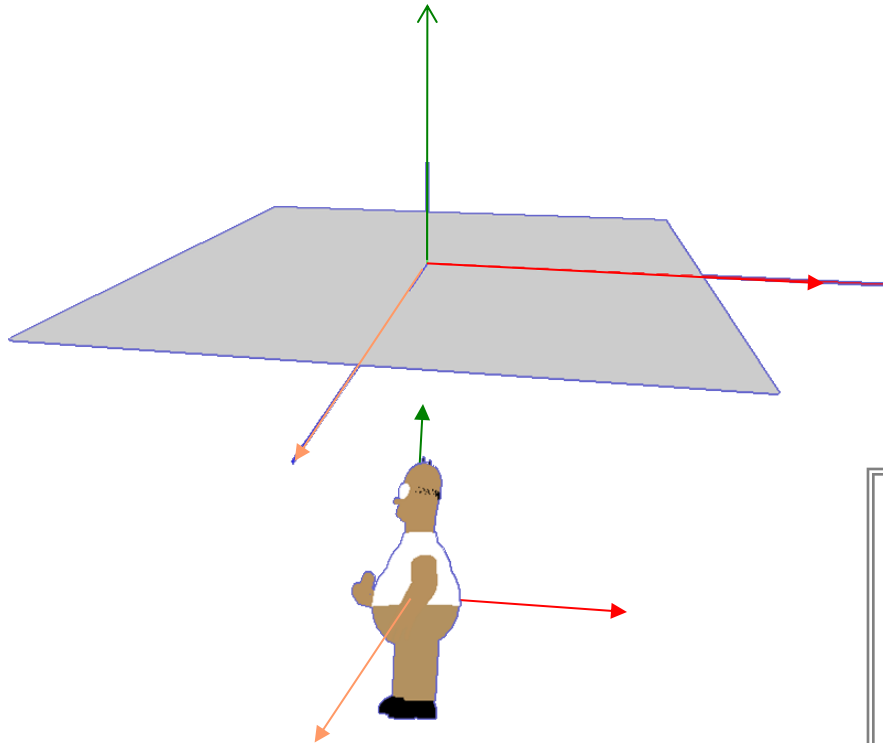
Imatge obtinguda...



Autre exemple: Homer definit en SC centrat en Homer



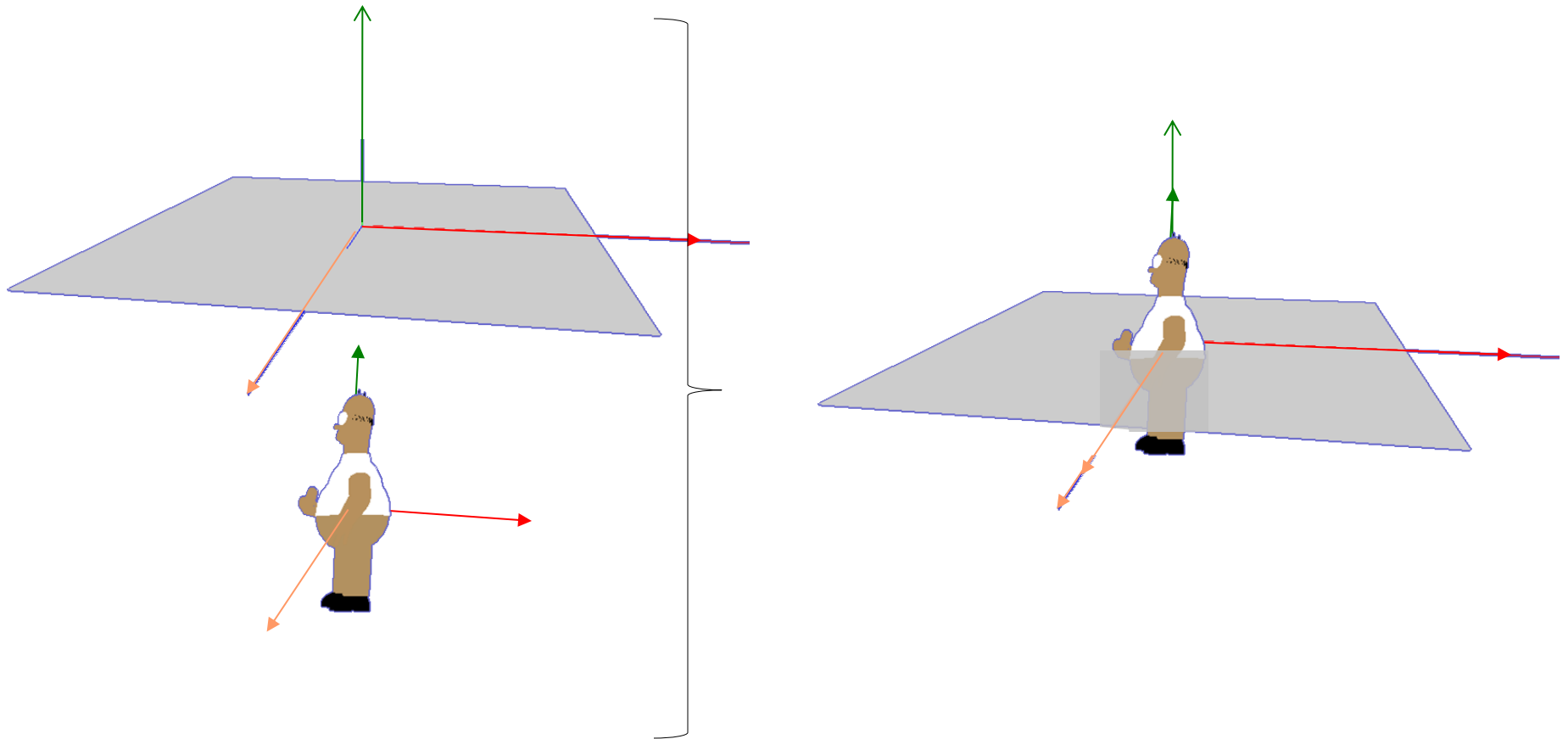
Igual que abans, per visualitzar...



```
// un sol cop  
per cada model  
    crear i omplir VAOi i VBOj  
fper
```

```
//cada cop que cal refrescar finestra  
per cada objecte  
    pinta_modeli()  
fper
```

Resultat....



Repasar Àlgebra i TG (Transformacions Geomètriques)

Si volem així...

Moure/pujar
 $\text{Trans}(0,+h,0)$

*SCA: SC de l'Aplicació
(o d'escena o World)*

- Cal aplicar TG que modifica les coordenades dels vèrtexs
- TG queda definida per matriu 4x4:

$$\mathbf{V}_A = \text{TG} \cdot \mathbf{V}_m = \text{T}(0,+h,0) \mathbf{V}_m$$

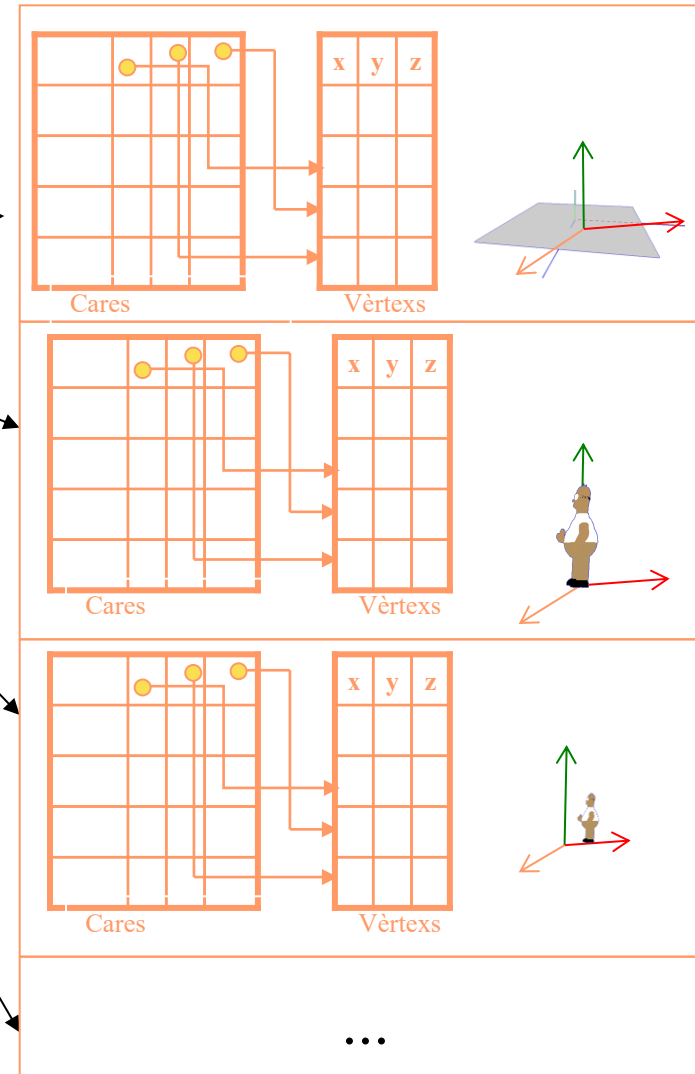
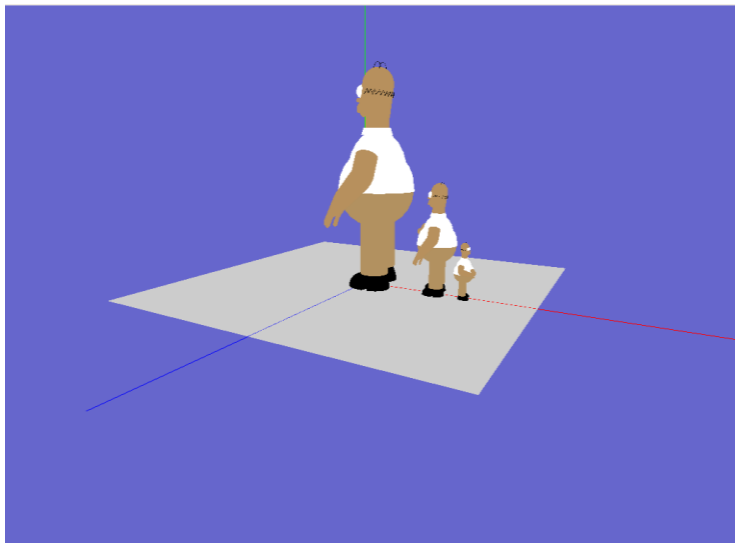
$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Però...quan transformem vèrtexs de SCM en SCA?

Escena: {objectes} i models en SCA

nom	model
terra		
h1		
h2		
h3		

Objectes



Models

Escenes: Objectes en SCA. Esquema de visualització

per cada objecte

```
llegir_Model();
```

```
modelTransformi(TGi);
```

```
aplicar_TG_a_model(TGi);
```

fper

```
/* CreaBuffers que crea un VAOi i  
VBOj per cada objecte*/
```

per cada objecte

```
crear i omplir VAOi, VBOj
```

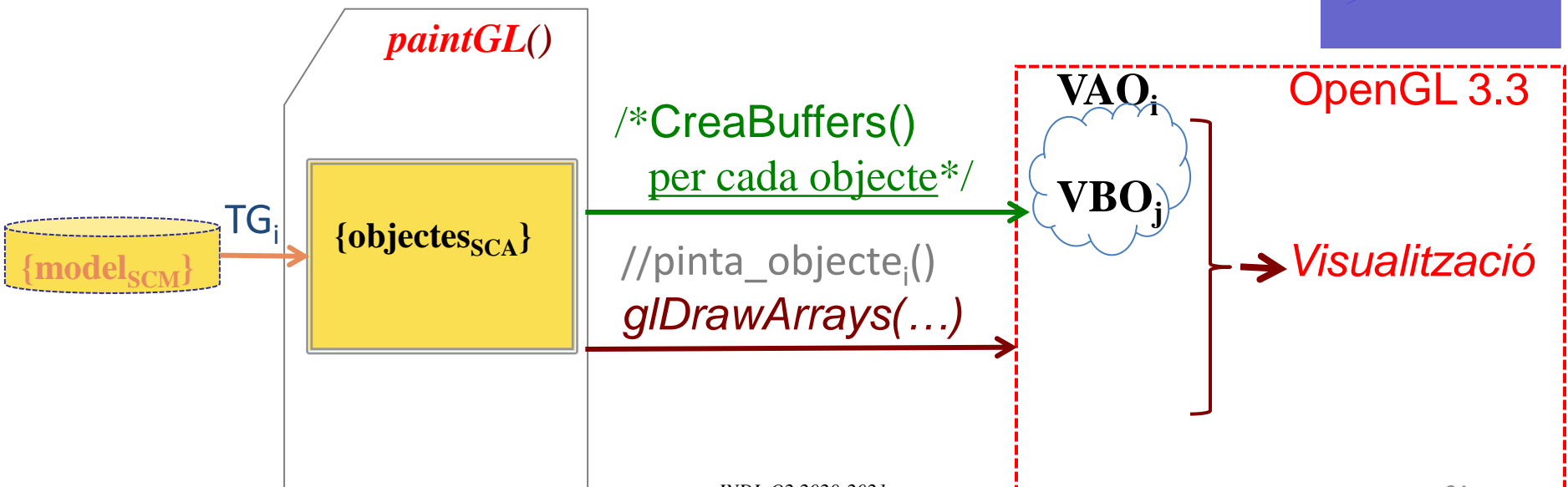
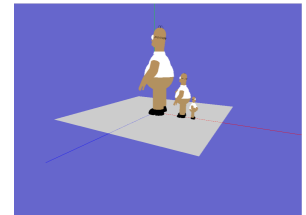
fper

```
//paintGL ();
```

```
per cada objectei
```

```
pinta_objectei(); //el seu VAOi
```

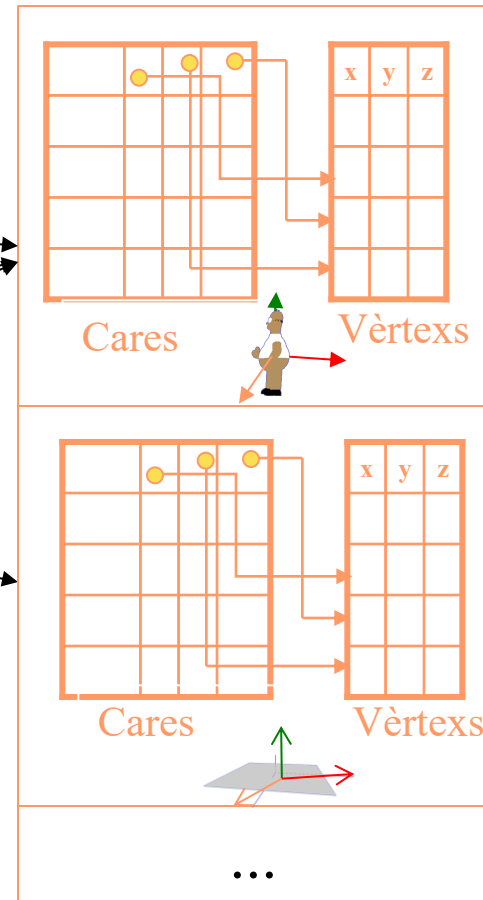
```
fper
```



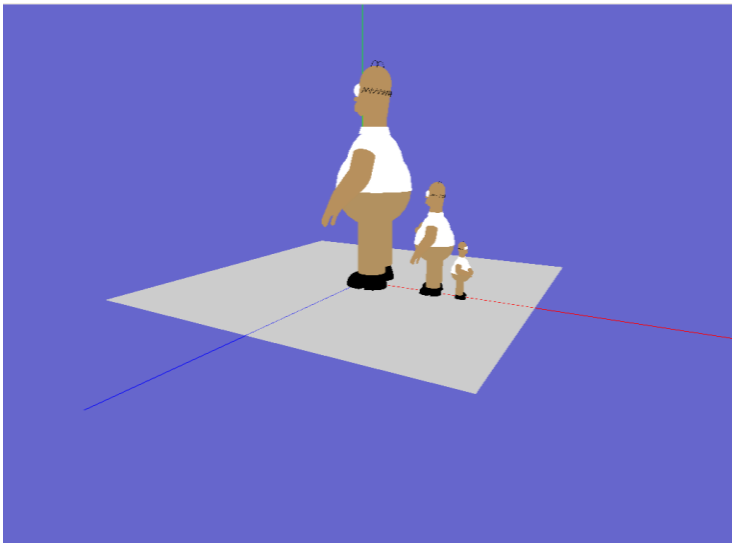
Reutilització de VAO/VBO (i)

Objectes

nom	...	TG/param	model
h1			
h2			
h3			
terra			



Un VAO/VBO
Compartit per
diferents
objectes!
Es “pinta” el
mateix VAO
vàries vegades.

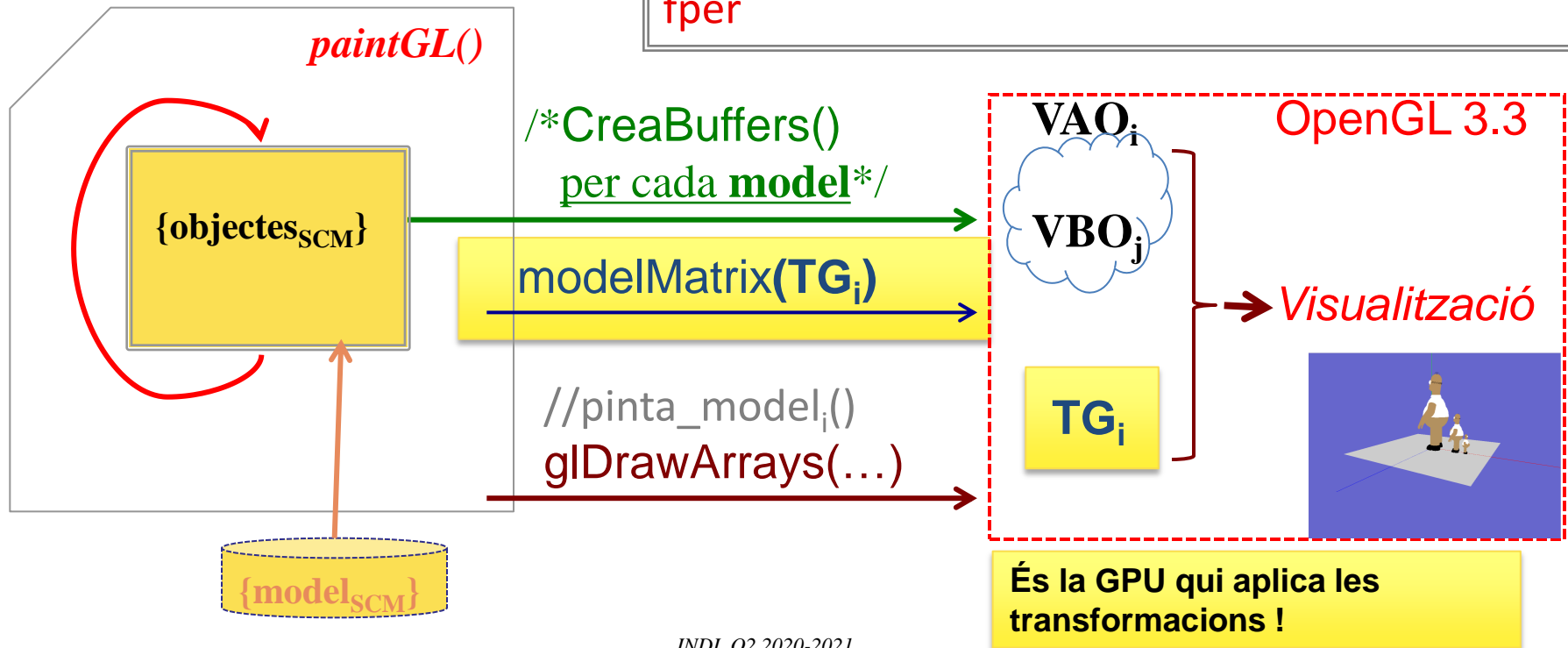


Models

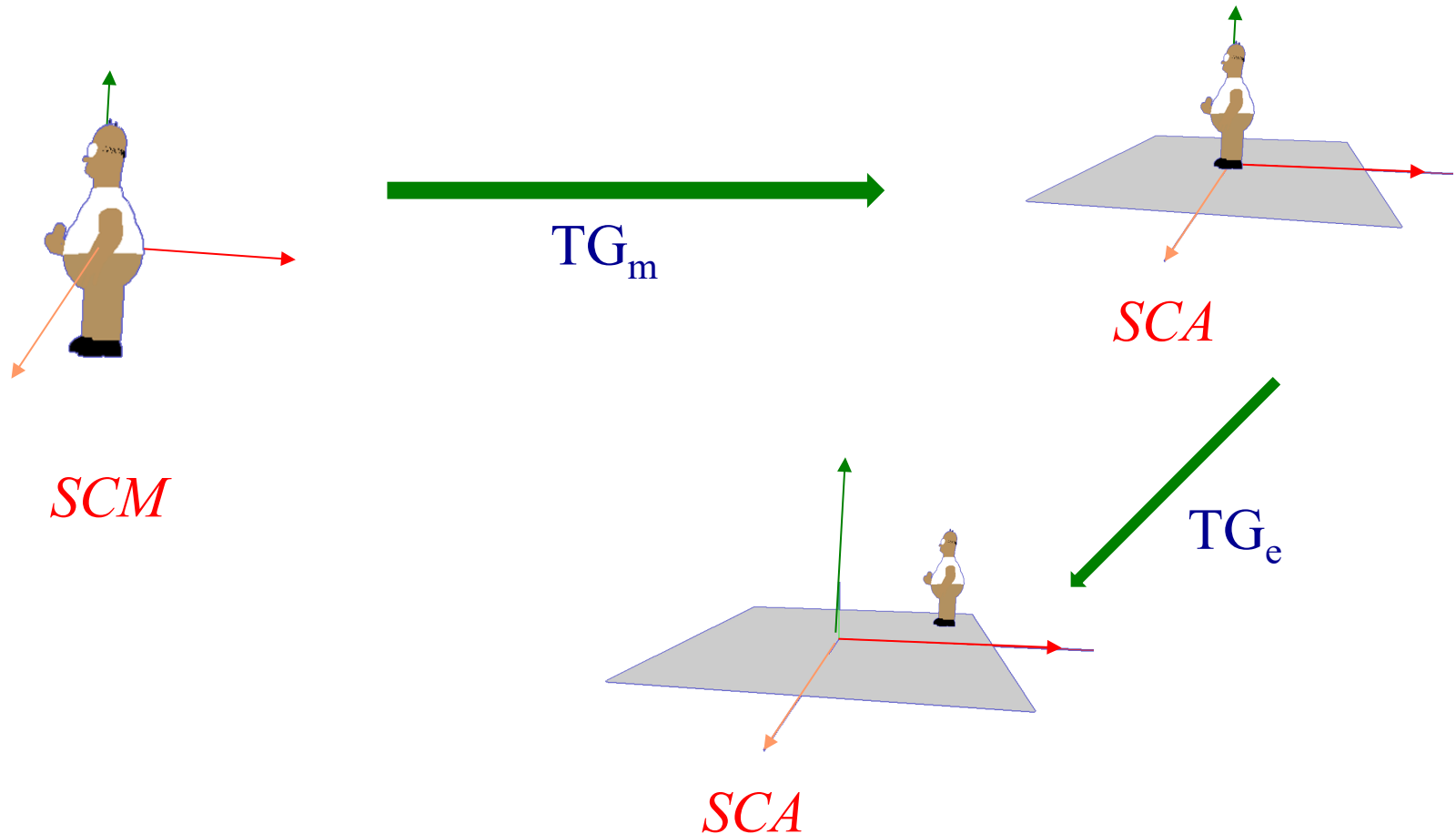
Reutilització de VAO/VBO (ii): Enviem TG

```
/*crear un únic VAOi i VBOj per cada  
model, en CreaBuffers()*/  
per cada model  
    llegir_Model();  
    crear i omplir VAOi, VBOj  
fper
```

```
//paintGL ();  
per cada objectei  
    /*Calcular la TGi a aplicar a model i  
    indicar a OpenGL la TGi */  
    modelTransformi(TGi);  
    modelMatrix(TGi); //envia “uniform”  
    pinta_modeli(); //el VAOi del seu model  
fper
```



Escenes: TG d'Aplicació



- La TG d'aplicació es pot descomposar en dues transformacions:

$$\mathbf{V}_A = \mathbf{TG}_e \cdot \mathbf{TG}_m \cdot \mathbf{V}_m$$

Classe 1: conceptes

- Model de fronteres: com guardar un triangle.
- Topologia implícita i explícita.
- Model vàlid.
- Filosofia de visualització en OpenGL 3.3: programes en CPU i GPU, VAO, VBO, ...
- Escena = conjunt d'objectes.
- SCM i SCA.
- Possibles estructures de dades per escenes.
- Filosofia de visualització d'escenes en OpenGL 3.3.

Classe 1: Contingut

- Introducció a la Informàtica Gràfica
- Models geomètrics
 - Un objecte
 - Un conjunt d'objectes (escena)

Bibliografia del “Llibre en CD” els temes:

- Geometria2D i 3D.
- Representació d'objectes geomètrics