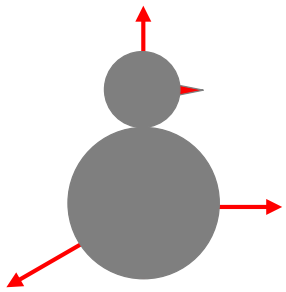
Bloc_2: Transformacions Geomètriques **i Models (sessió 2)**

Què heu fet en la 1ª sessió Bloc 2?

- Pintar algun objecte glut (secció 1)
- Utilitzar OpenGL per aplicar TG a un objecte (secció 2)
  - Entendre els paràmetres de les crides i composició d'operacions
  - Recordeu que OpenGL aplica la matriu del top de la pila MODELVIEW als vèrtexs i que les operacions amb matrius afecten a la matriu del top de la pila activa.
  - Utilitzar callbacks de teclat i ratolí per modificar TG
- Utilitzar OpenGL per a aplicar diferentes TG als diferentes objectes de l'escena
  - Cal Push/Pop Matrius
  - Gir dels dos triangles
- Crear una escena utilitzant objectes glut (secció 3)



Sessions i Objectius

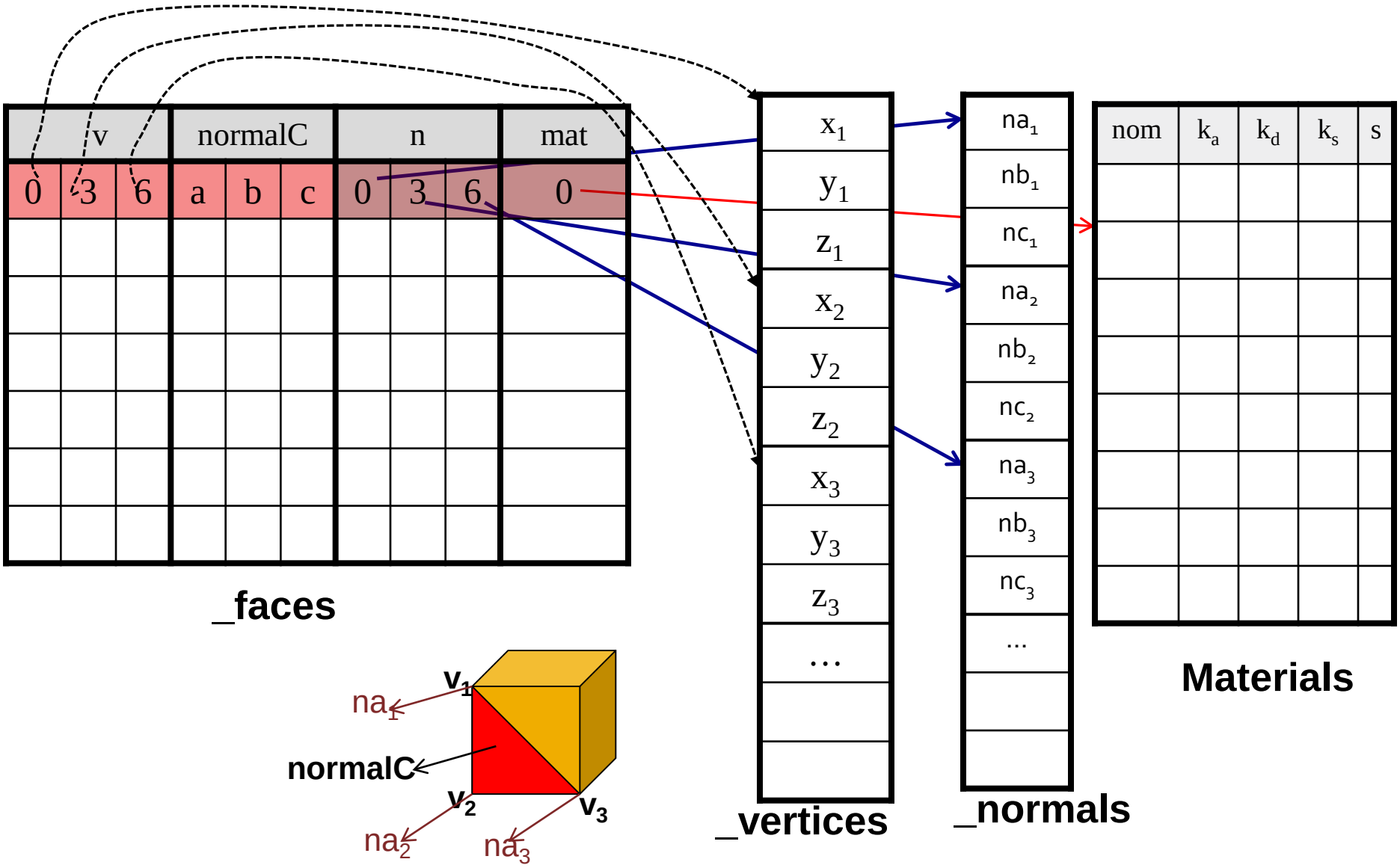
- Sessió 1 –seccions 1 a 3- : Transformacions Geomètriques
  - Objectes glut
  - Entendre el funcionament de les transformacions geomètriques per: posicionar i animar objectes.
  - Utilització en OpenGL.
  - Exercici: pintar ninot de neu i interactivament girar i escalar.

- **Sessió 2 –seccions 4 i 5-:**
  - **Carregar models geomètrics (OBJ) i visualitzar en OpenGL.**
  - **Aplicació resum de conceptes: crear una escena concreta, poder girar l'escena, poder moure un dels objectes.**

Càrrega de Models (1)

- Classe Model: permet carregar *objecte.obj*
  - /assig/idi/Model
  - Analitzeu el model.h (classe Model)
  - `Model::load(std::string filename)`

    Inicialitza les estructures de dades a partir d'un model en format OBJ-Wavefront en disc
- Nou make:
  g++ -o bloc2 bloc2.o model.o –lGLU –lGL –lglut
- En */assig/vig/models* o */assig/idi/Model* trobareu models d'objectes.
  - Si els copieu a un directori local, per cada .obj copieu també (si existeix) el .mtl → definició dels materials corresponents.
- Més models els podeu trobar a la xarxa



Analitzeu l'arxiu model.h

Compte!! amb el nom dels camps de Material que en l'esquema són simbòlics; p.e. k_d és float diffuse[4]

Classe Model: observacions

- Totes les cares són triangles. Les cares es triangulen en el moment de llegir-se.

- Sempre podeu fer servir la normal per cara *normalC*
  - Model::load() l'haurà inicialitzat amb un vector unitari perpendicular al triangle.

- El vector de normals -per vèrtex- pot ser buit (si el fitxer original no el tenia).

Classe Model

- Tres std::vector<T> de la stl:
  - Un amb la informació de les cares: _faces
  - Un amb les components de normals per vèrtex: _normals
  - Un amb les coordenades dels vèrtexs: _vertices
- Declaració:
  - Model m;
  - vector <Model> models;
- Càrrega d'un model:
  - m.load ("…");
  - m.load (argv[1]);

Classe Model

- Hi ha mètodes consultors que retornen const
  - El codi en què les feu servir haurà de ser "const-correcte"

```
const std::vector< Face>&  faces() const {
        return _faces; }
```

- Exemples

// accés a info de la cara 13

```
const Face &f = m.faces()[12];
```

// accés a les coordenades del segon vèrtex de la cara i exemple d'ús.

```
glVertex3dv(&m.vertices() [f.v[1]]);
```

// accés al color de la cara 13 i exemple d'utilització

```
glColor4fv (Materials[f.mat].diffuse);
```

Classe Model

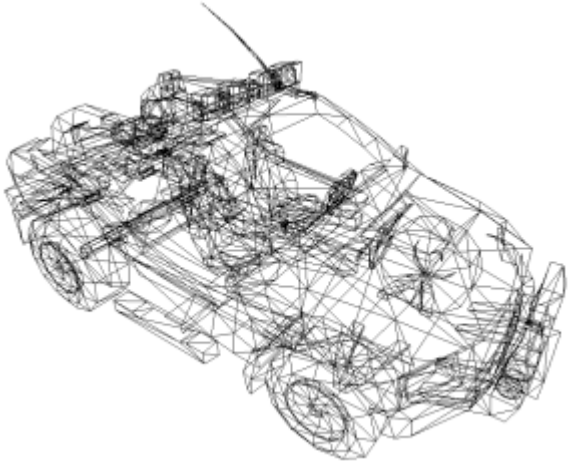
- Un altre exemple…. Recorregut del vector de vèrtexs

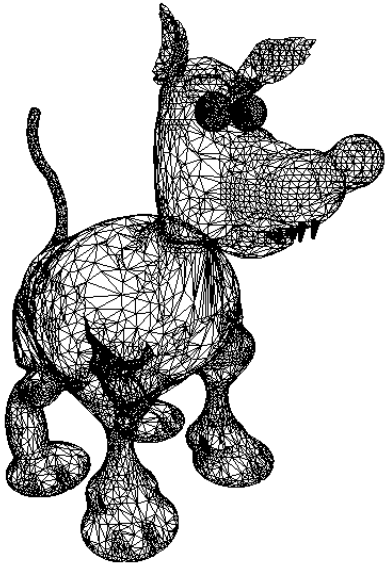
```
for (int i=0; i<m.vertices().size; i=i+3) {
    const Vertex &v=m.vertices()[i];
    // Aquí fer el que ens calgui accedint al vèrtex v,
    // per exemple si el volguéssim usar per pintar-lo faríem:
    // glVertex3dv (&v);
    // O una altra alternativa d'ús
    // const double *v = &m.vertices()[i];
    // glVertex3dv (v)
}
```

OpelGL: glPolygonMode

- void glPolygonMode (GLenum face, GLenum mode)
  - face: les cares a les que ens referim
    - GL_FRONT_AND_BACK → aquest
    - GL_FRONT
    - GL_BACK
  - mode: mode de dibuix
    - GL_POINT
    - GL_LINE
    - GL_FILL

OpenGL: glPolygonMode





OpenGL: Depth test

- Algoritme de z-buffer
  - *glEnable(GL_DEPTH_TEST);*
  - Esborrar el buffer de profunditat:
    glClear(…. | GL_DEPTH_BUFFER_BIT);
  - En glutInitDisplayMode afegir:
    | GLUT_DEPTH

- *Recordeu que qualsevol comanda OpenGL ha d'anar després d'haver creat la finestra gràfica (amb glut).*

Començant a treballar...Secció 4

- Mètode que permet carregar OBJ
  - Proveu inicialment: HomerProves.OBJ
- refresh()
  - Modifiqueu/completeu per a pintar el model (podeu fer un mètode `pinta_model()`.
    - Ha de recórrer totes les cares del model i enviar-les a pintar .
    - Inicialment, pinteu totes les cares del mateix color.
    - Després cada cara del color indicat en el camp “difusse” del seu material.

Començant a treballar...Secció 4

- Feu un mètode que:
  - a) carrega el model de disc
  - b) calcula la seva capsa mínima contenidora. La podeu guardar en variables globals o afegir un camp a la classe Model.

- Proveu carregar qualsevol OBJ i que surti sempre centrat, inicialment sense retallar i sense deformació =>
  - Completeu el mètode anterior per a calcular l'escalat, girs i translació que cal aplicar. Podeu guardar els paràmetres en un camp nou del model o com vulgueu.
  - Completeu refresh () => per a calcular la TG a partir dels paràmetres (vigileu en l'orde de les crides!) just **abans** d'enviar a pintar el model. *Recordeu* que la pila activa ha de ser GL_MODELVIEW i que als diferents objectes els hi podeu aplicar diferents TG (=> pushs i pops):

Secció 5: Exercici a lliurar

- Heu de crear l'escena de la darrera versió de l'enunciat :
  - Terra: quadrat amb centre (0,-0.4,0) i aresta 1.5.

    Observació: en la vista inicial, potser no es veu el terra ➔ cap problema ☺.
  - Ninot de neu d'exercici 3.
  - legoman.obj amb alçada 0.5 i amb el vèrtex (**xmin**,ymin,zmax) de la base de la seva capsa (considerant la nova mida) en el punt (-0.75,-0.4,0.75).
- S'ha de poder girar interactivament l'escena (en les figures ho està).
  - *Observació: al girar l'escena pot quedar retallada pel volum de visió ➔ cap problema ☺.*
- S'ha de poder desplaçar el legoman sobre el terra => mireu guió.

Bloc_2: Transformacions Geomètriques **i Models (sessió 2)**