

---

# OpenGL

## Vertex - Matrix

## Push/Pop - Trigo

---

Renaud Hélias

ST informatique services

---

---

---

# OpenGL : Introduction

---

Exemples d'application classique

---

# OpenGL : Introduction

---

## Exemples d'application classique

- dessin technique
  - pointillés, texte

# OpenGL : Introduction

---

## Exemples d'application classique

- dessin technique
  - pointillés, texte
- simulation
  - graphes, traits, points

# OpenGL : Introduction

---

## Exemples d'application classique

- dessin technique
    - pointillés, texte
  - simulation
    - graphes, traits, points
  - interaction avec la souris
    - zoom, rotation, déplacement
-

# OpenGL : Introduction

---

## Exemples d'application classique

- dessin technique
  - pointillés, texte
- simulation
  - graphes, traits, points
- interaction avec la souris
  - zoom, rotation, déplacement
- jeux d'arcade
  - vue orthogonale, calques superposés

# OpenGL : Introduction

---

## Exemples d'application classique

- dessin technique
    - pointillés, texte
  - simulation
    - graphes, traits, points
  - interaction avec la souris
    - zoom, rotation, déplacement
  - jeux d'arcade
    - vue orthogonale, calques superposés
  - Quake 3
    - moteur 3D
-



# OpenGL : Introduction

---

Outils générant des figures OpenGL

- animation : Maya, Poser, Cinema 4D
  - modélisation : 3ds Max, modeler Quake3
  - schématique : Catia, AutoCAD
-

# OpenGL : Introduction

---

Outils générant des figures OpenGL

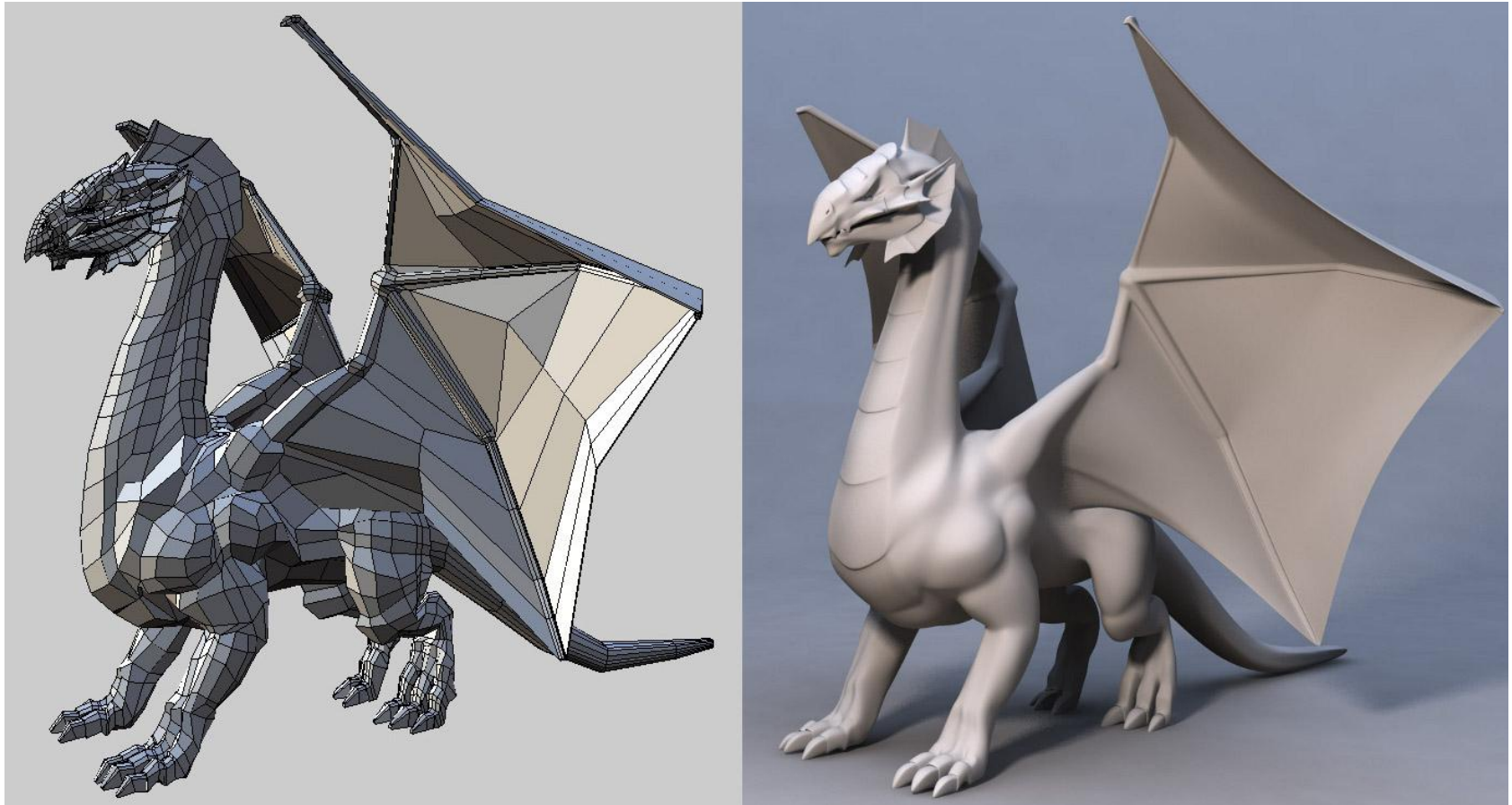
- animation : Maya, Poser, Cinema 4D
- modélisation : 3ds Max, modeler Quake3
- schématique : Catia, AutoCAD

Langages réutilisant les principes OpenGL

- Flex : `flash.geom.Matrix3D`
  - Java : `java.awt.Graphics2D`
-

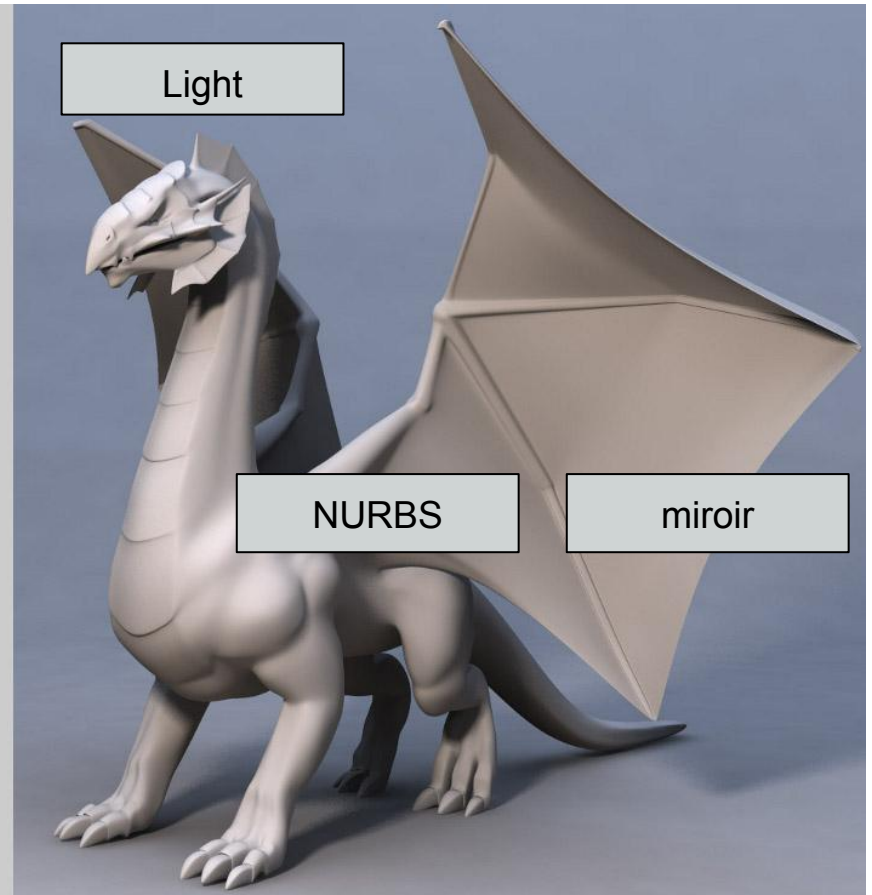
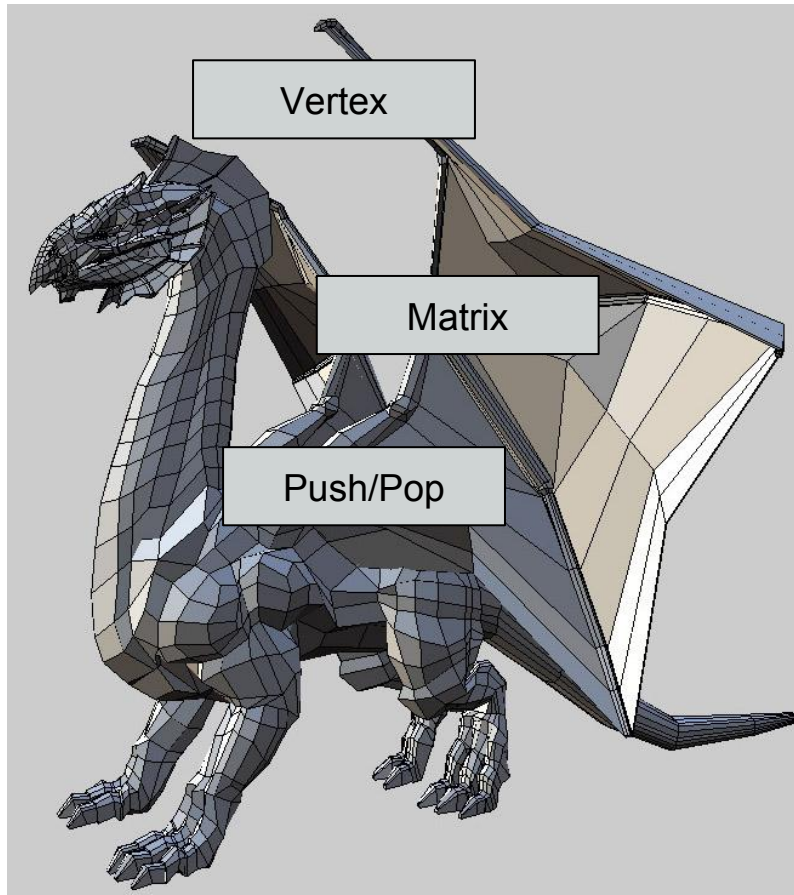
# OpenGL : Introduction

---



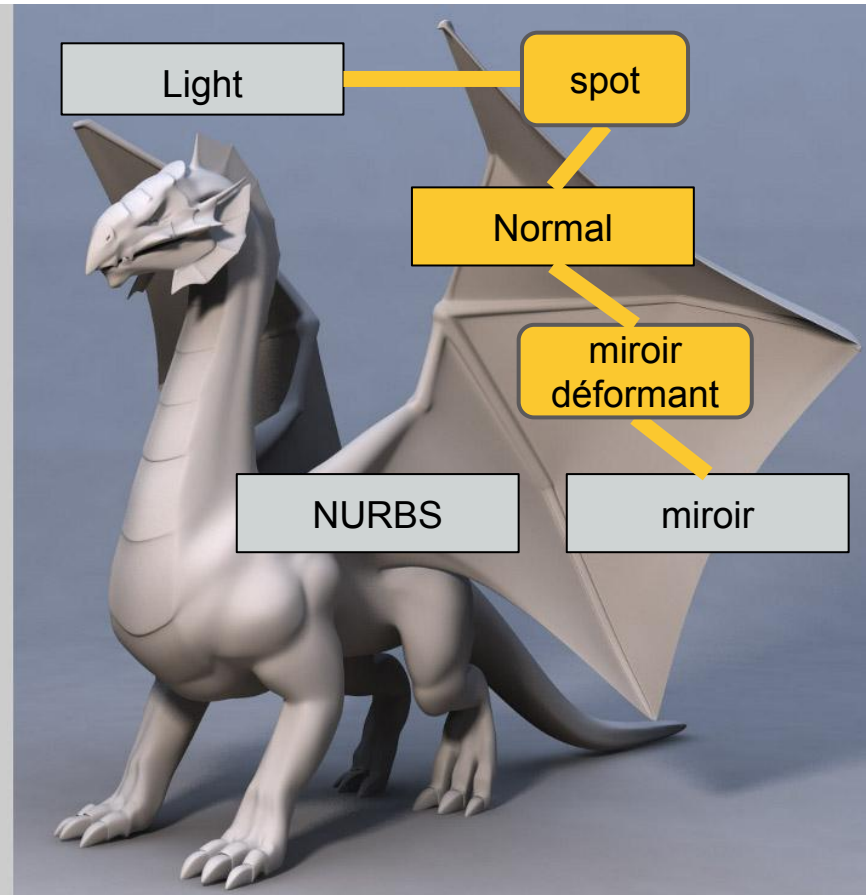
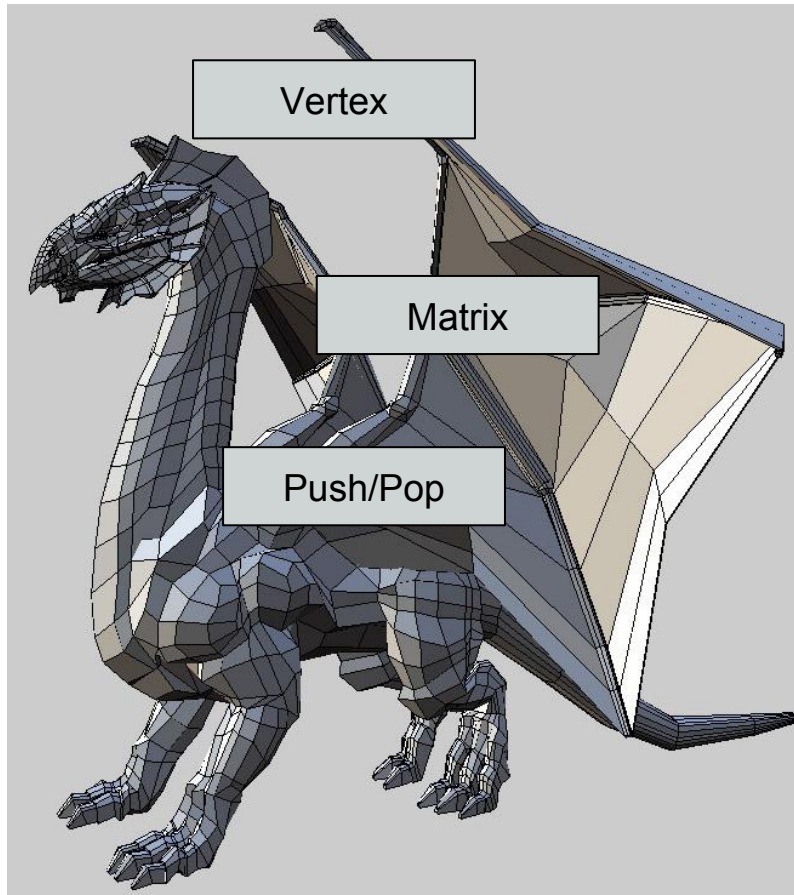
# OpenGL : Introduction

---



# OpenGL : Introduction

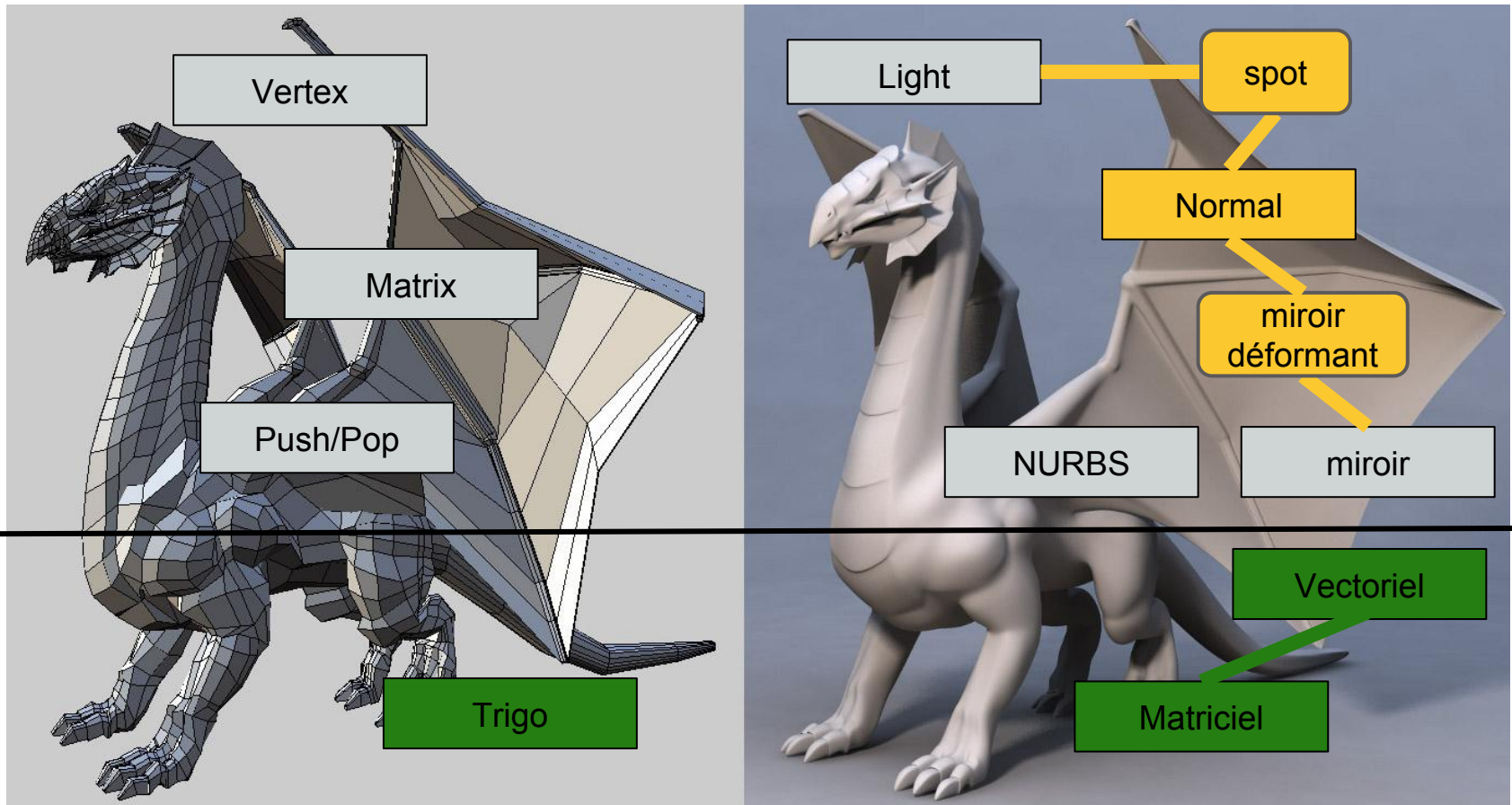
---





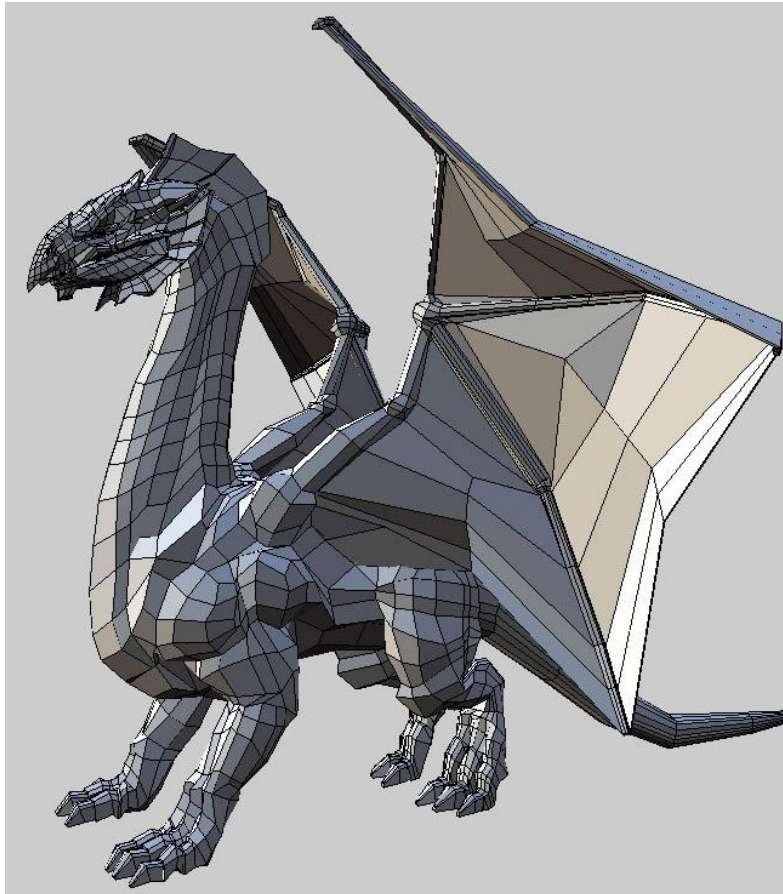
# OpenGL : Introduction

---



# OpenGL : Sommaire

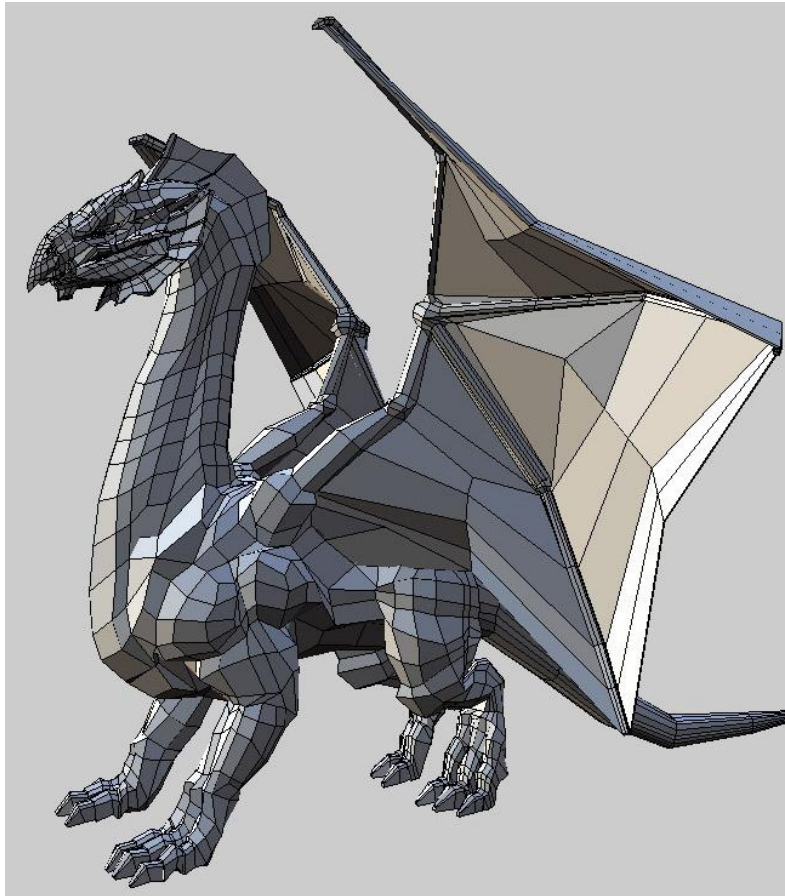
---



- Vertex
  - Introduction
  - vertex, triangles, texCoord
  - Exercice : mesh, sphere
- Matrix
  - Rotate and then Translate
  - Exemple : rubans
  - Exercice : cerf-volant
- Push/Pop
  - push/pop, arbres
  - Exemple : rubans au vent
  - Exercice : cerf-volant amélioré
- Trigonométrie
  - Cercle trigo
  - Pythagore
  - Exemple : rotation à la souris
  - Exercice : levier

# OpenGL : Vertex

---



- Vertex
  - Introduction
  - vertex, triangles, texCoord
  - Exercice : mesh, sphere



---

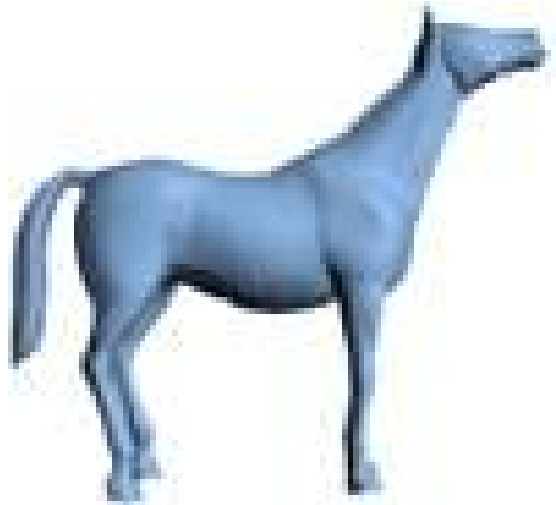
# OpenGL : Vertex

**Afficher une  
image 3D**

---

# OpenGL : Vertex

---



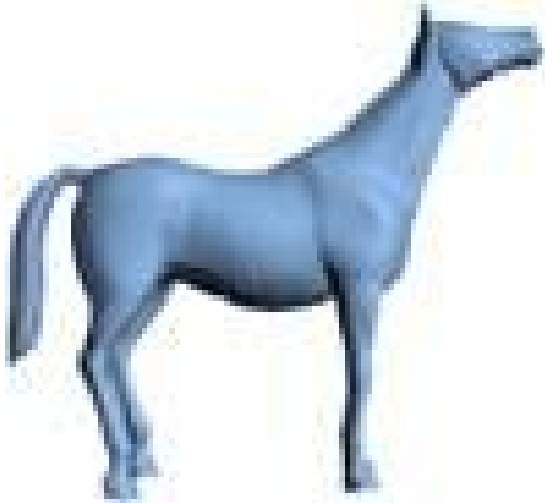
horse.mesh



hayden.md2

# OpenGL : Vertex

---



**horse.mesh** : format texte

**v** : vertex list

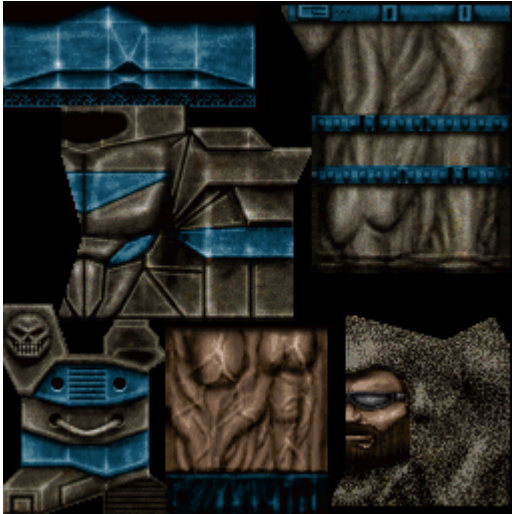
**vn** : vertex normal list

**t** : triangles list

horse.mesh

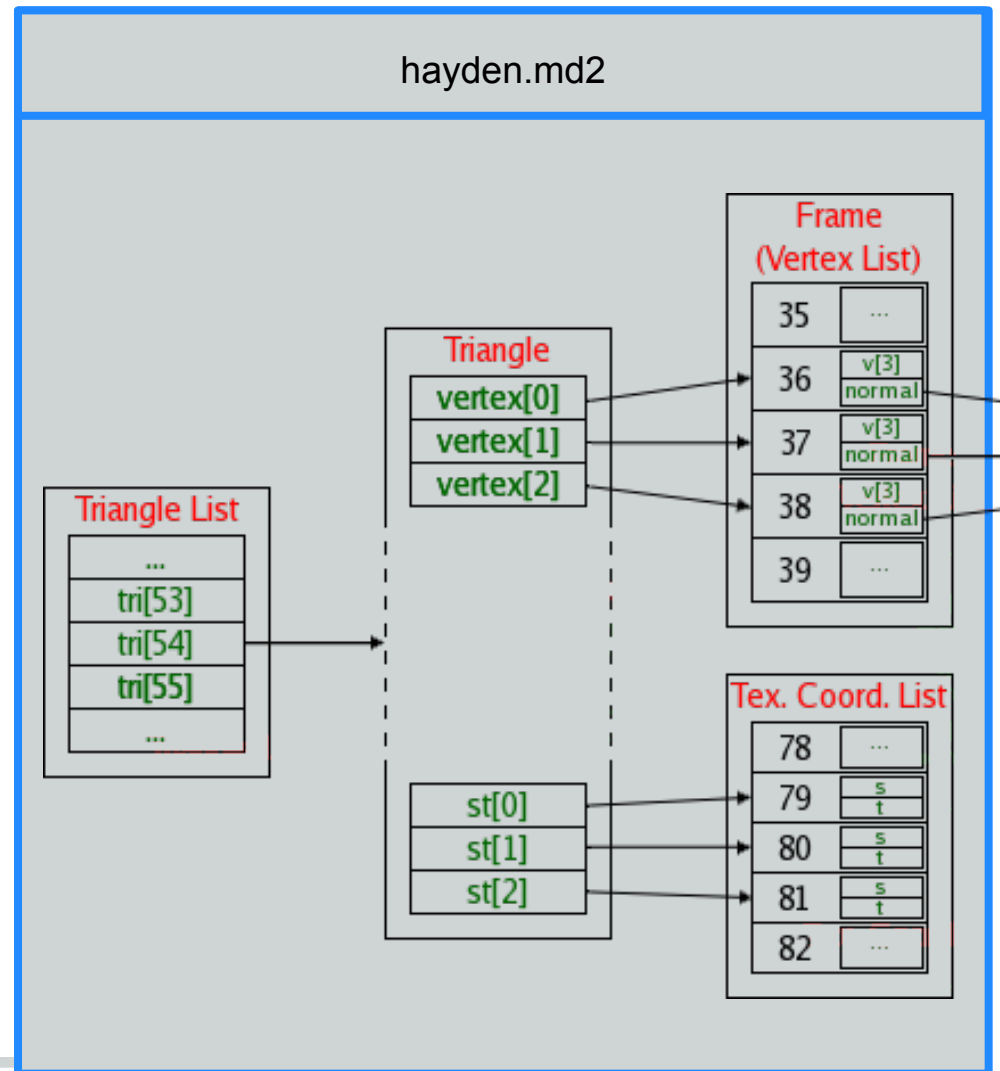
```
v -0.008161330 0.589476000 -0.378438000
v -0.014804900 0.577342000 -0.383194000
v -0.011917500 0.570075000 -0.390049000
v -0.002765810 0.562748000 -0.396552000
v -0.007911980 0.565378000 -0.394545000
v -0.004118260 0.592503000 -0.378059000
vn -0.996776900 0.030902918 0.074032571
vn 0.812284916 -0.3524165480.464753474
vn 0.933587627 -0.3051862870.187817658
vn 0.925693680 -0.3208014850.200443556
vn 0.804808802 -0.3593820860.472363535
vn 0.864660952 -0.169791321 -0.472792074
vn 0.818481317 -0.205028242 -0.536704530
vn 0.938309476 -0.240245450 -0.248711581
f 201//201 204//204 209//209
f 201//201 209//209 210//210
f 210//210 209//209 211//211
f 210//210 211//211 212//212
f 205//205 208//208 213//213
f 205//205 213//213 214//214
```

# OpenGL : Vertex



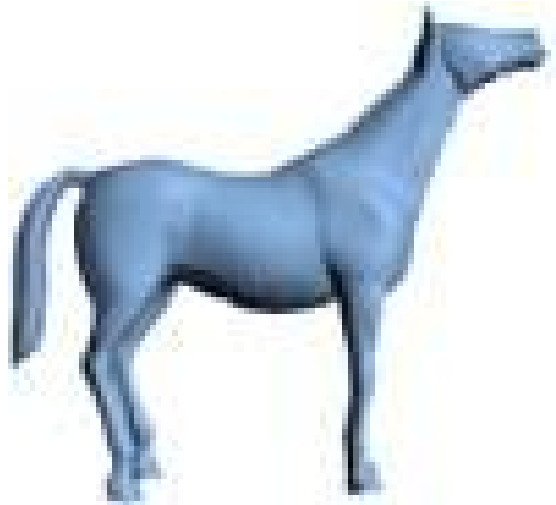
hayden.md2 : format binaire

hayden.png : texture



# OpenGL : Vertex

---



`vertexArray + triangleArray`



`texCoordArray`

# OpenGL : Vertex

---

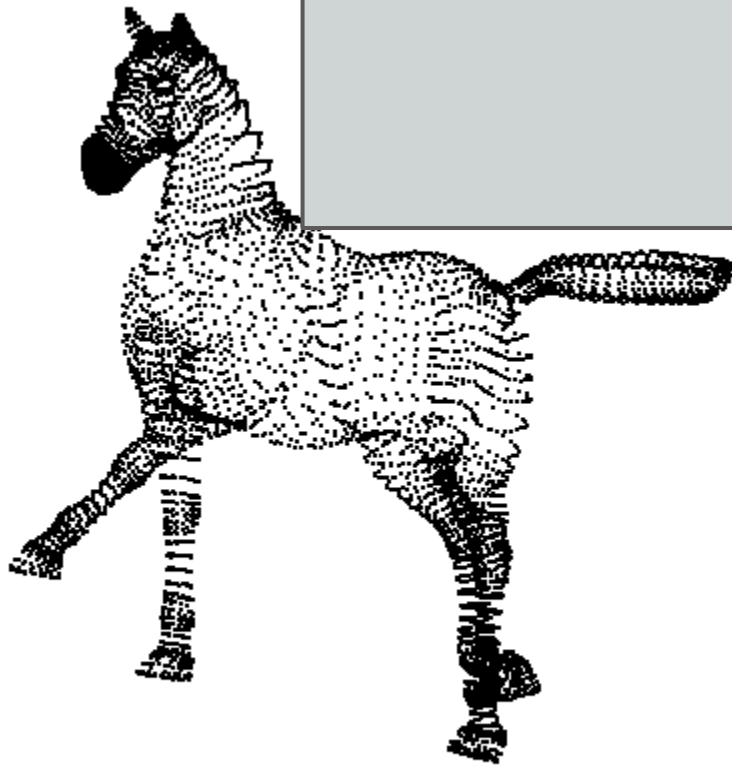
vertexArray : liste de points

```
float vertexArray[] = {  
0.3f,0.3f,0.0f, // x,y,z point 0  
0.3f,0.7f,0.0f, // x,y,z point 1  
0.7f,0.7f,0.0f, // x,y,z point 2  
0.7f,0.3f,0.0f // x,y,z point 3  
}
```

---

# OpenGL : Vertex

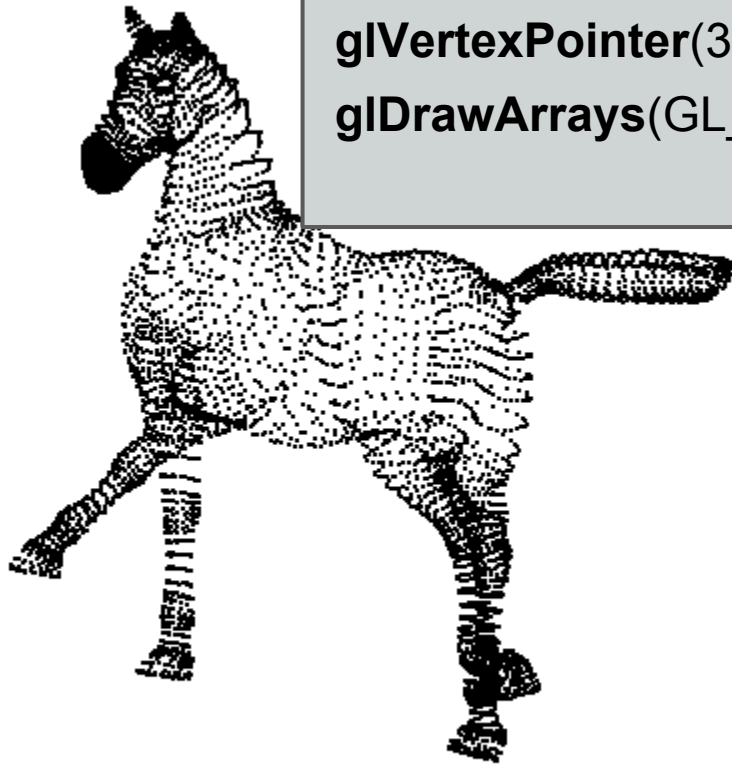
---



`vertexArray`

# OpenGL : Vertex

---



```
glVertexPointer(3, GL_FLOAT, 0, vertexArray);  
glDrawArrays(GL_POINTS, 0, sizeof(vertexArray));
```



# OpenGL : Vertex

---



```
glEnableClientState(GL_VERTEX_ARRAY);  
glVertexPointer(3, GL_FLOAT, 0, vertexArray);  
glDrawArrays(GL_POINTS, 0, sizeof(vertexArray));  
glDisableClientState(GL_VERTEX_ARRAY);
```

# OpenGL : Vertex

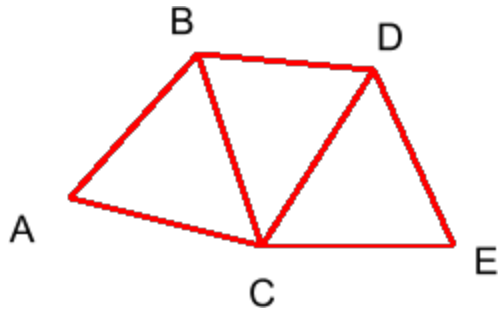
---

```
glEnableClientState(GL_VERTEX_ARRAY);  
glVertexPointer(3, GL_FLOAT, 0, vertexArray);  
glDrawArrays(GL_TRIANGLES, 0, sizeof(vertexArray));  
glDisableClientState(GL_VERTEX_ARRAY);
```

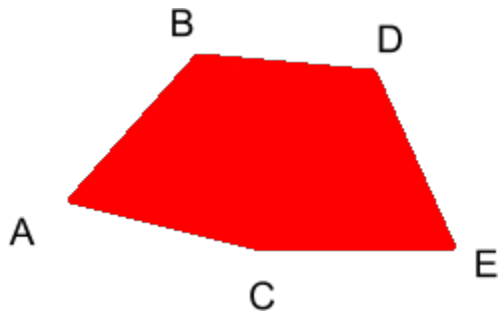


# OpenGL : Vertex

---



triangles = A,B,C , B,C,D , C,D,E



# OpenGL : Vertex

---

Index : liste de triangles (GL\_TRIANGLES)

```
unsigned int trianglesArray[] = {  
0,1,2, // relier points 0,1,2  
0,2,3 // relier points 0,2,3  
}
```

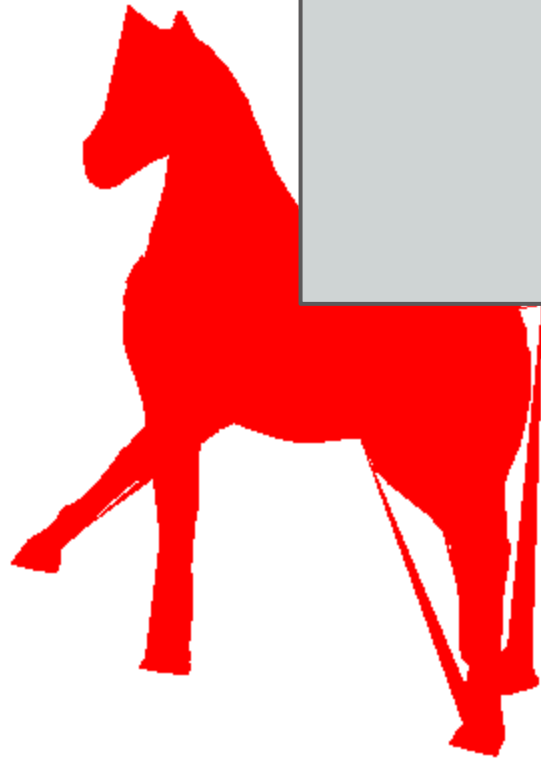
Index : liste de traits (GL\_LINES)

```
unsigned int traitsArray[] = {  
0,1 , 1,2 , 2,3 , 3,0 // quatre traits  
}
```

---

# OpenGL : Vertex

---



**vertexArray**

**trianglesArray**

# OpenGL : Vertex

---



```
glVertexPointer(3, GL_FLOAT, 0, vertexArray);  
glDrawElements(GL_TRIANGLES, sizeof(trianglesArray),  
GL_UNSIGNED_INT, trianglesArray);
```

# OpenGL : Vertex

---



```
glEnableClientState(GL_VERTEX_ARRAY);  
glVertexPointer(3, GL_FLOAT, 0, vertexArray);  
glDrawElements(GL_TRIANGLES, sizeof(trianglesArray),  
GL_UNSIGNED_INT, trianglesArray);  
glDisableClientState(GL_VERTEX_ARRAY);
```

# OpenGL : Vertex

---

texCoord : coordonnées de texture

```
float texCoordArray[] = {  
    0.0f,0.0f, // point 0  
    0.0f,1.0f, // point 1  
    1.0f,1.0f, // point 2  
    1.0f,0.0f // point 3  
}
```

---



# OpenGL : Vertex

---



**texCoordArray**

**vertexArray**

**trianglesArray**

# Opengl : Vertex

---



**texId**

**texCoordArray**

**vertexArray**

**trianglesArray**

# Opengl : Vertex

---



```
glBindTexture (GL_TEXTURE_2D, texId);
```

```
glTexCoordPointer(2, GL_FLOAT, 0, texCoordArray);
```

```
glVertexPointer(3, GL_FLOAT, 0, vertexArray);
```

```
glDrawElements(GL_TRIANGLES, sizeof(trianglesArray),  
GL_UNSIGNED_INT, trianglesArray);
```

# Opengl : Vertex

---



```
glEnable (GL_TEXTURE_2D);  
glBindTexture (GL_TEXTURE_2D, texId);  
glEnableClientState(GL_VERTEX_ARRAY);  
glEnableClientState(GL_TEXTURE_COORD_ARRAY);  
glTexCoordPointer(2, GL_FLOAT, 0, texCoordArray);  
glVertexPointer(3, GL_FLOAT, 0, vertexArray);  
glDrawElements(GL_TRIANGLES, sizeof(trianglesArray),  
GL_UNSIGNED_INT, trianglesArray);  
glDisableClientState(GL_TEXTURE_COORD_ARRAY);  
glDisableClientState(GL_VERTEX_ARRAY);  
glDisable (GL_TEXTURE_2D);
```

# A vous de jouer

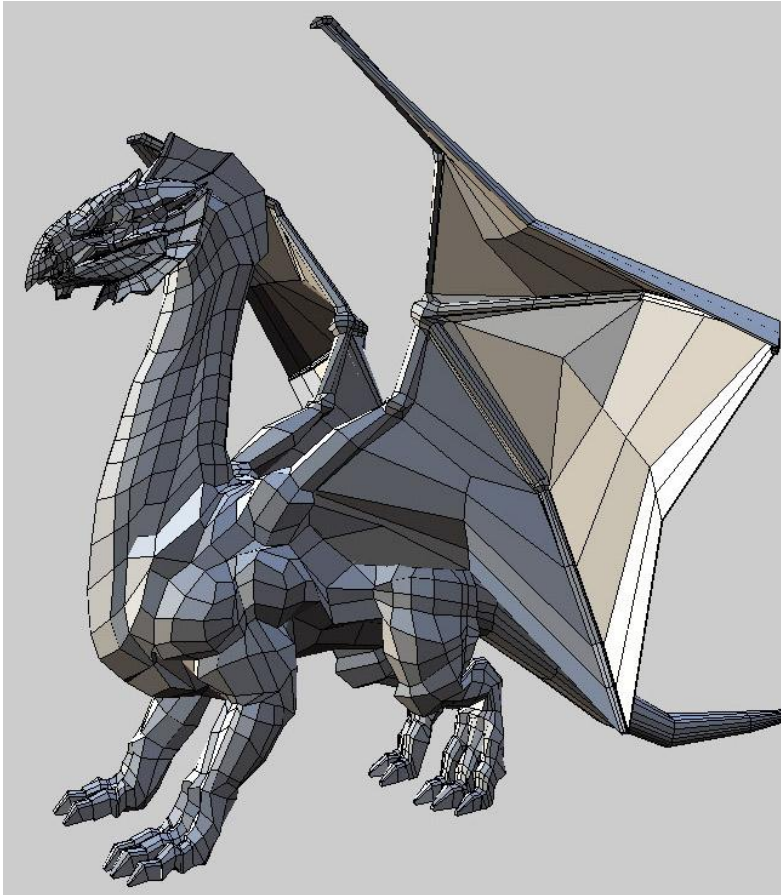
---

- Extraire une image 3D d'un fichier
    - L'afficher avec une texture
  - Dessiner un Cerf-volant
    - Avec un point d'attache en  $(0,0,0)$
    - Peut comprendre quelques ficelles
  - Sphère
    - Fonction récursive pour générer les points
    - Boucles imbriquées
    - Algorithme type "diviser pour régner"
      - point au centre d'un triangle
      - un triangle transformé en trois triangles
      - appliquer un rayon de taille fixe
-



# OpenGL : Matrix

---



- Vertex
  - Introduction
  - vertex, triangles, texCoord
  - Exercice : mesh, sphere
- Matrix
  - Rotate and then Translate
  - Exemple : rubans
  - Exercice : cerf-volant

---

# **OpenGL : Matrix**

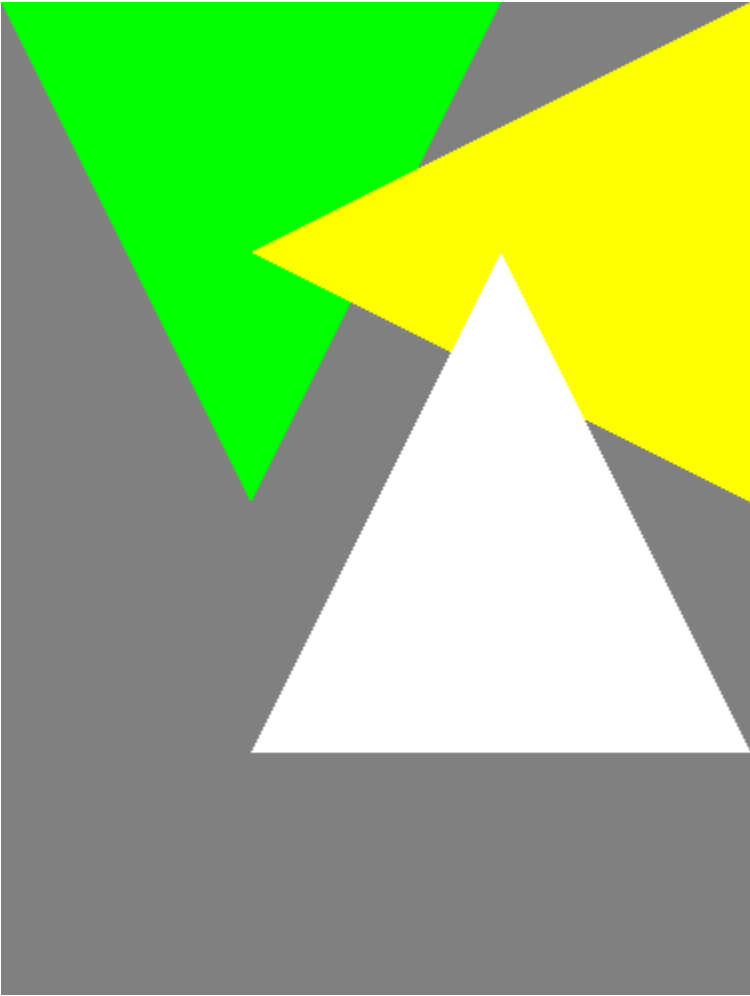
**Rotate and then  
Translate**

---



# OpenGL : Matrix

---

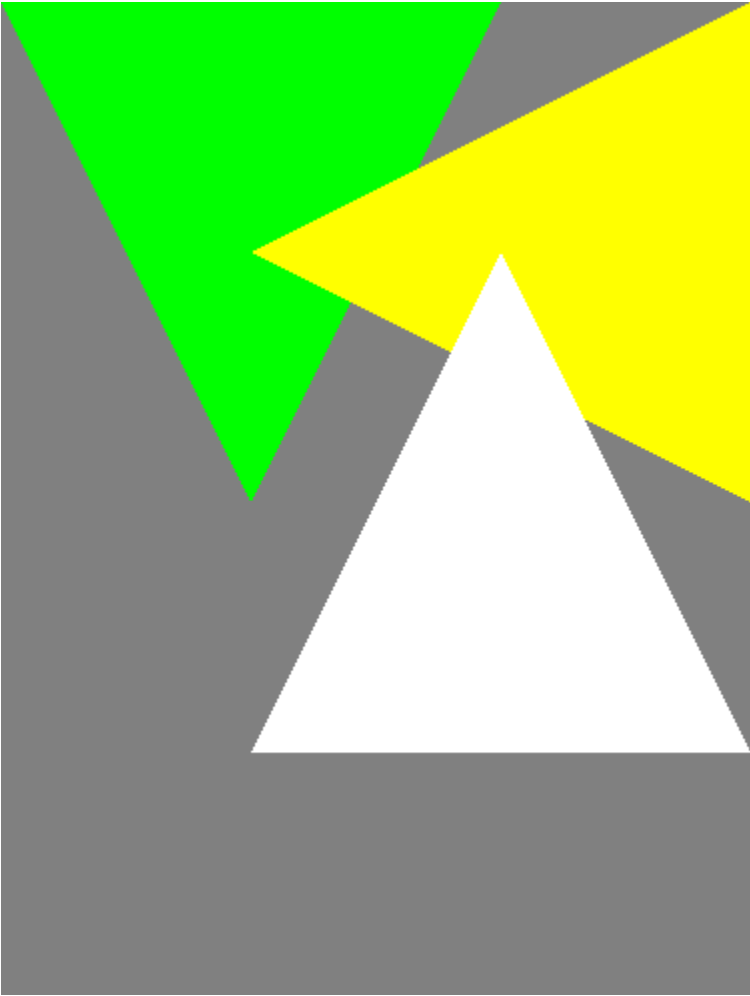


```
glColor3d(1,1,1);  
triangle();
```

```
glColor3d(1,1,0);  
glTranslated(0.0,1.0,0.0);  
glRotated(90.0,0.0,0.0,1.0);  
triangle();
```

```
glColor3d(0,1,0);  
glTranslated(0.0,1.0,0.0);  
glRotated(90.0,0.0,0.0,1.0);  
triangle();
```

# OpenGL : Matrix



```
glColor3d(1,1,1);  
triangle();
```

```
glColor3d(1,1,0);  
glTranslated(0.0,1.0,0.0);  
glRotated(90.0,0.0,0.0,1.0);  
triangle();
```

```
glColor3d(0,1,0);  
glTranslated(0.0,1.0,0.0);  
glRotated(90.0,0.0,0.0,1.0);  
triangle();
```

# OpenGL : Matrix

---

Exemple :

- Découper une image en carrés
  - Les colonnes forment des rubans
  - Chaque ruban est tenu par le haut
  - Faire bouger les rubans
-

# OpenGL : Matrix

---

Cadeau :

- `srand(47)` permet d'avoir une suite `rand()` déterministe
- un timer peut incrémenter une variable

`angle=(timerValue+rand())%360`

---

# OpenGL : Matrix

---



```
// rotation  
// afficher  
// contre-rotation  
//un pas en bas
```

# OpenGL : Matrix

---



```
//pour chaque ligne  
// rotation  
// afficher  
// contre-rotation  
//un pas en bas  
//fin pour
```

# OpenGL : Matrix

---



```
//pour chaque ligne  
// rotation  
// afficher  
// contre-rotation  
//un pas en bas  
//fin pour  
//un pas à droite  
//8 pas en haut
```

# OpenGL : Matrix



```
//pour chaque colonne  
//pour chaque ligne  
// rotation  
// afficher  
// contre-rotation  
//un pas en bas  
//fin pour  
//un pas à droite  
//8 pas en haut  
//fin pour
```



# OpenGL : Matrix

---

Conclusion :

- La matrice déplace le repère, sur lequel est appliqué le vertex, sur lequel est appliqué l'image

# A vous de jouer

---

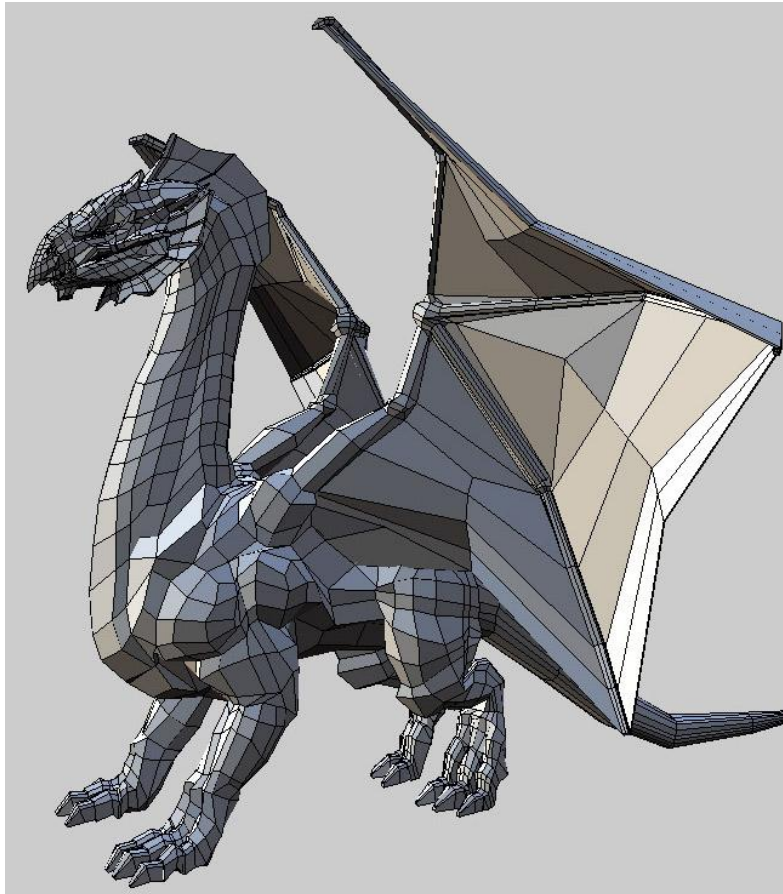
- Faire voler un cerf-volant
  - Attacher une corde en bas
  - Remonter la corde pas à pas, avec un rand rotate
  - Une fois en haut de la corde, poser le cerf-volant

---

---

# OpenGL : Push/Pop

---



- Vertex
  - Introduction
  - vertex, triangles, texCoord
  - Exercice : mesh, sphere
- Matrix
  - Rotate and then Translate
  - Exemple : rubans
  - Exercice : cerf-volant
- Push/Pop
  - push/pop, arbres
  - Exemple : rubans au vent
  - Exercice : cerf-volant amélioré

---

# OpenGL : Push/Pop

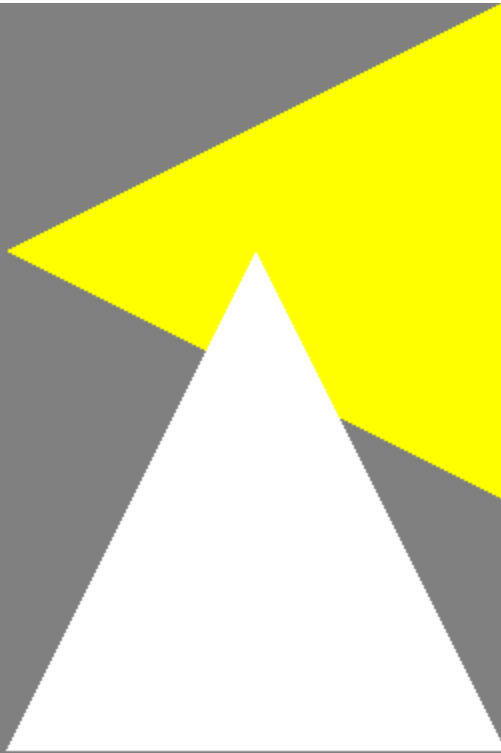
**Se perdre dans la  
matrice, c'est permis**

---

---

# OpenGL : Push/Pop

---

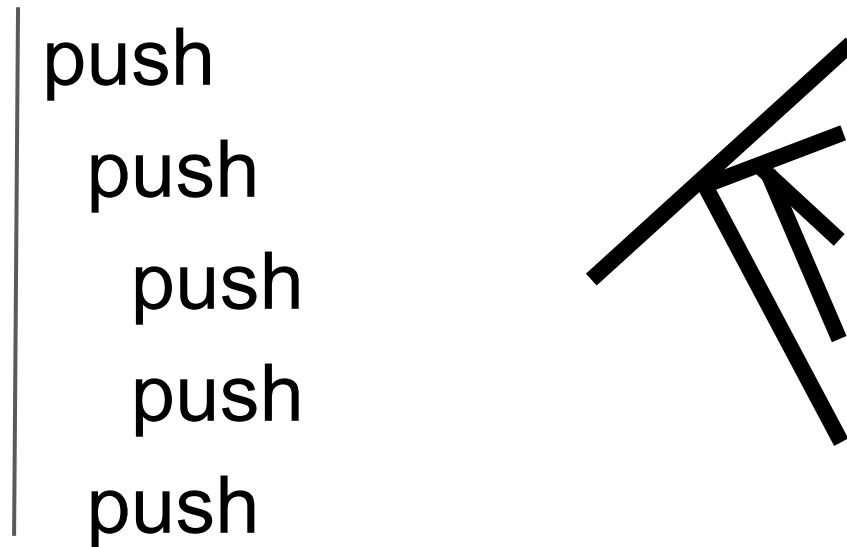


```
glPushMatrix();  
    glColor3d(1,1,1);  
    triangle();  
glPopMatrix();  
glPushMatrix();  
    glColor3d(1,1,0);  
    glTranslated(0.0,1.0,0.0);  
    glRotated(90.0,0.0,0.0,1.0);  
    triangle();  
glPopMatrix();  
glPushMatrix();  
    glColor3d(0,1,0);  
    glTranslated(0.0,1.0,0.0);  
    glRotated(90.0,0.0,0.0,1.0);  
    triangle();  
glPopMatrix();
```

# OpenGL : Push/Pop

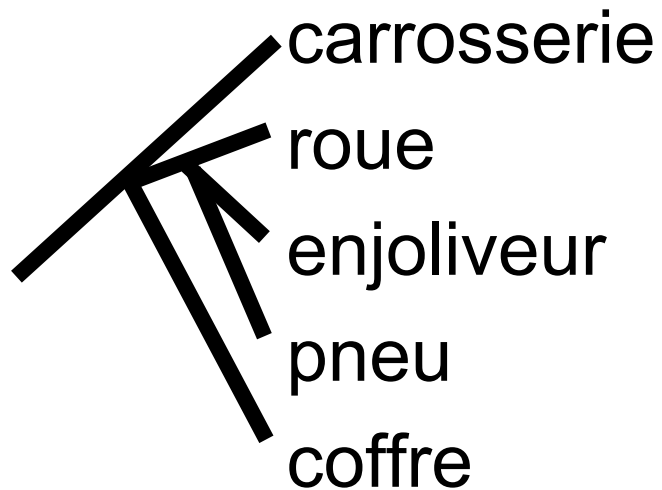
---

push,push,push,pop,push,pop,pop,push,pop...



# OpenGL : Push/Pop

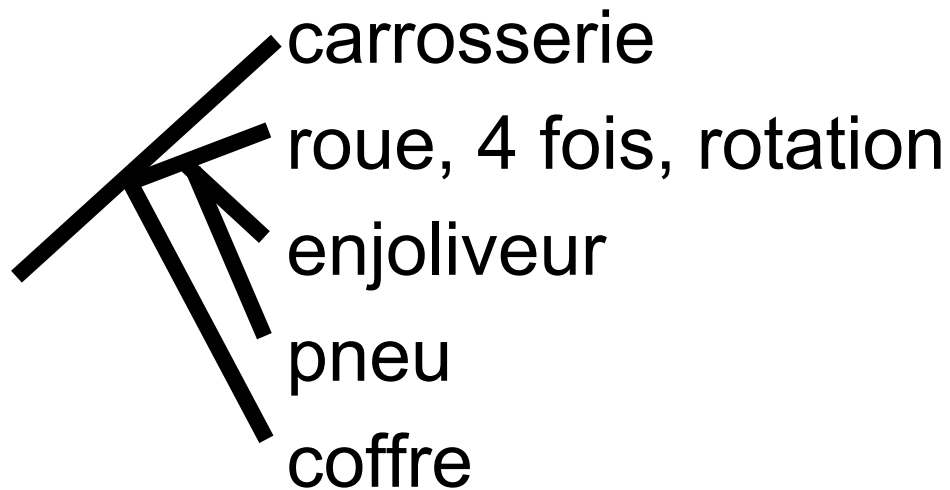
---





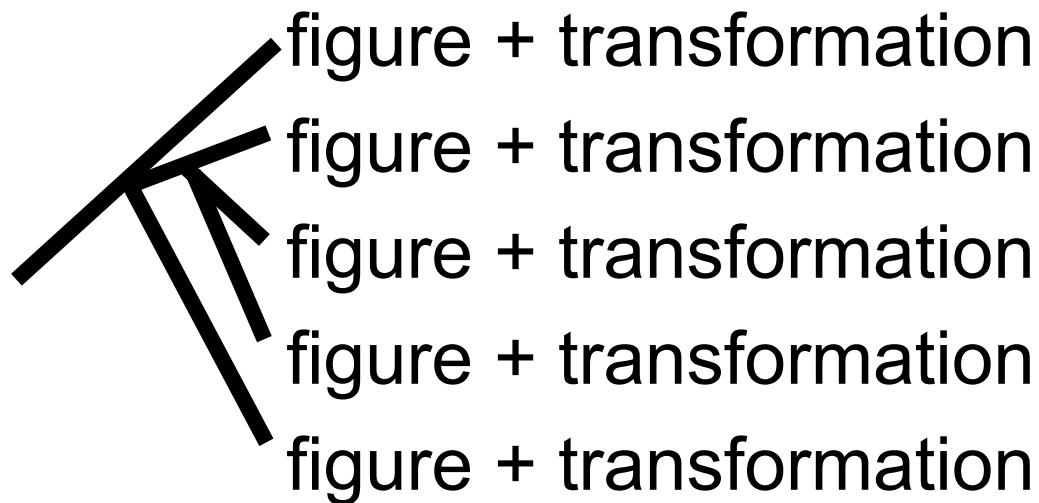
# OpenGL : Push/Pop

---



# OpenGL : Push/Pop

---



# OpenGL : Push/Pop

---

Exemple :

- . Découper une image en carrés
  - . Les colonnes forment des rubans
  - . Chaque ruban est tenu par le haut
  - . Faire voler les rubans au vent
-

# OpenGL : Push/Pop

---



```
// rotation x/y/z  
// afficher  
// un pas en avant
```

# OpenGL : Push/Pop

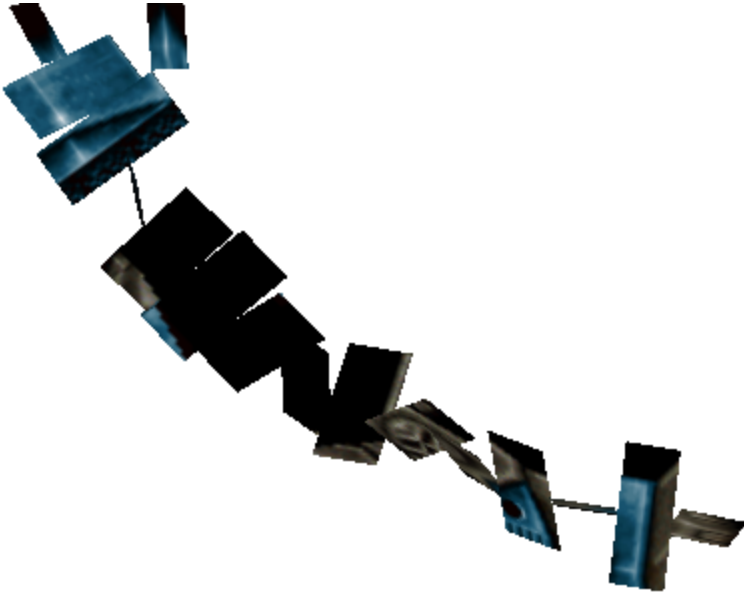
---



```
//pour chaque ligne  
// rotation x/y/z  
// afficher  
// un pas en avant  
//fin pour  
// je me suis perdu
```

# OpenGL : Push/Pop

---



```
//push  
//aller à la colonne suivante  
//pour chaque ligne  
// rotation x/y/z  
// afficher  
// un pas en avant  
//fin pour  
// je me suis perdu  
//pop
```

# OpenGL : Push/Pop

---



```
//pour chaque colonne  
//push  
//aller à la colonne suivante  
//pour chaque ligne  
// rotation x/y/z  
// afficher  
// un pas en avant  
//fin pour  
// je me suis perdu  
//pop  
//fin pour
```

# A vous de jouer

---

- Décoration du cerf-volant
  - Ajouter plusieurs queues au cerf-volant

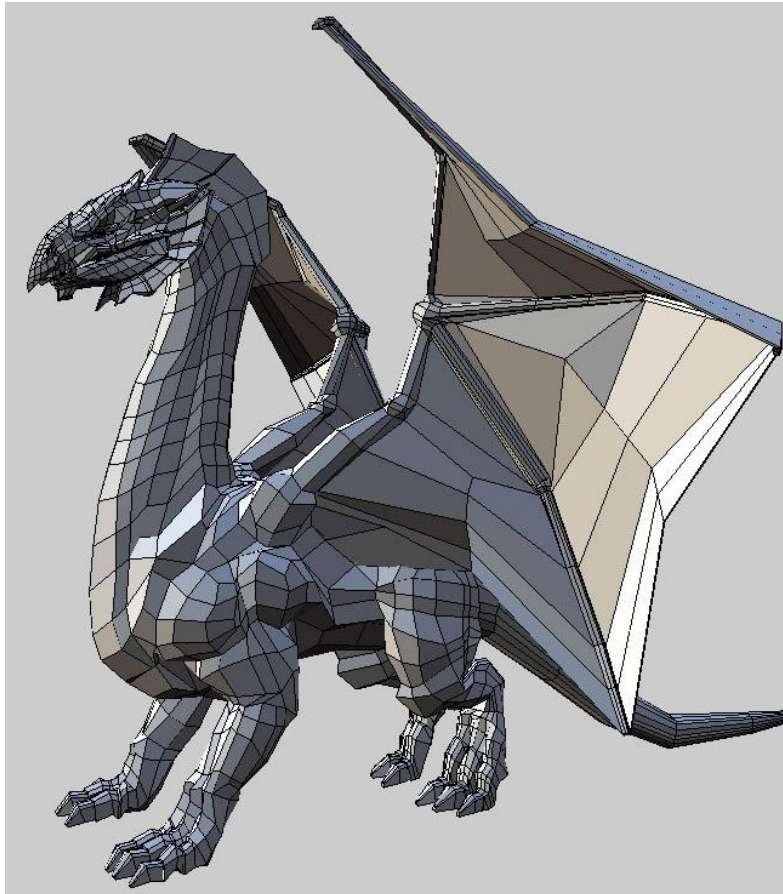


---

---

# OpenGL : Trigonométrie

---



- Vertex
  - Introduction
  - vertex, triangles, texCoord
  - Exercice : mesh, sphere
- Matrix
  - Rotate and then Translate
  - Exemple : rubans
  - Exercice : cerf-volant
- Push/Pop
  - push/pop, arbres
  - Exemple : rubans au vent
  - Exercice : cerf-volant amélioré
- Trigonométrie
  - Cercle trigo
  - Pythagore
  - Exemple : rotation à la souris
  - Exercice : levier

---

# **OpenGL : Trigo**

## **Images**

## **paramétrables**

---

---

# OpenGL : Trigonométrie

---

Exemples d'application classique

---

# OpenGL : Trigonométrie

---

Exemples d'application classique

- Détecter une zone à la souris
  - sélectionner un objet

# OpenGL : Trigonométrie

---

## Exemples d'application classique

- Détecter une zone à la souris
  - sélectionner un objet
- Déplacements à la souris
  - déplacements relatif
  - autour d'un point
  - suivant un plan

# OpenGL : Trigonométrie

---

## Exemples d'application classique

- Détecter une zone à la souris
  - sélectionner un objet
- Déplacements à la souris
  - déplacements relatif
  - autour d'un point
  - suivant un plan
- Construire une figure
  - selon les spécifications données

# OpenGL : Trigonométrie

---

Transformation

---



# OpenGL : Trigonométrie

---

## Transformation

- déplacement
  - coordonnée => coordonnée

# OpenGL : Trigonométrie

---

## Transformation

- déplacement
  - coordonnée => coordonnée
- agrandissement
  - coordonnée => coordonnée

# OpenGL : Trigonométrie

---

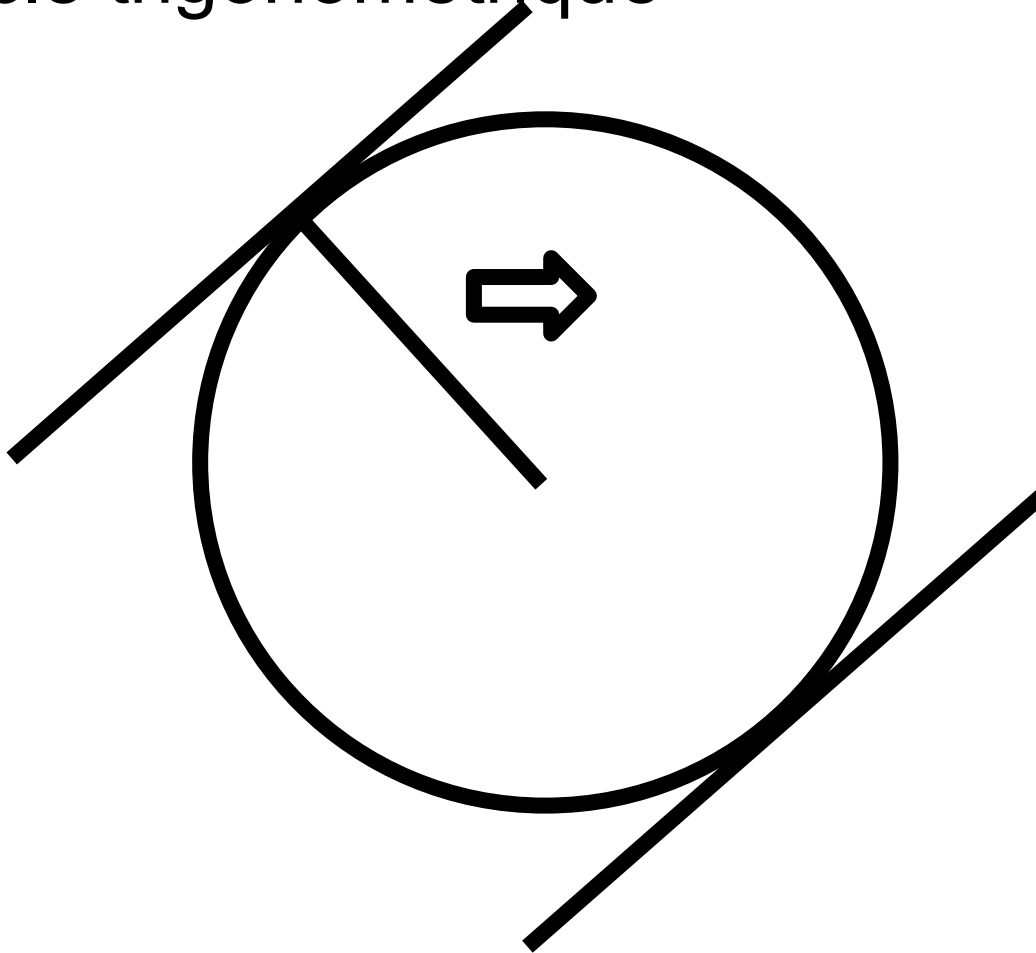
## Transformation

- déplacement
  - coordonnée => coordonnée
- agrandissement
  - coordonnée => coordonnée
- rotation
  - coordonnée => angle
  - angle => coordonnée

# OpenGL : Trigonométrie

---

Cercle trigonométrique



$$\cos(0) = ?$$

$$\sin(0) = ?$$

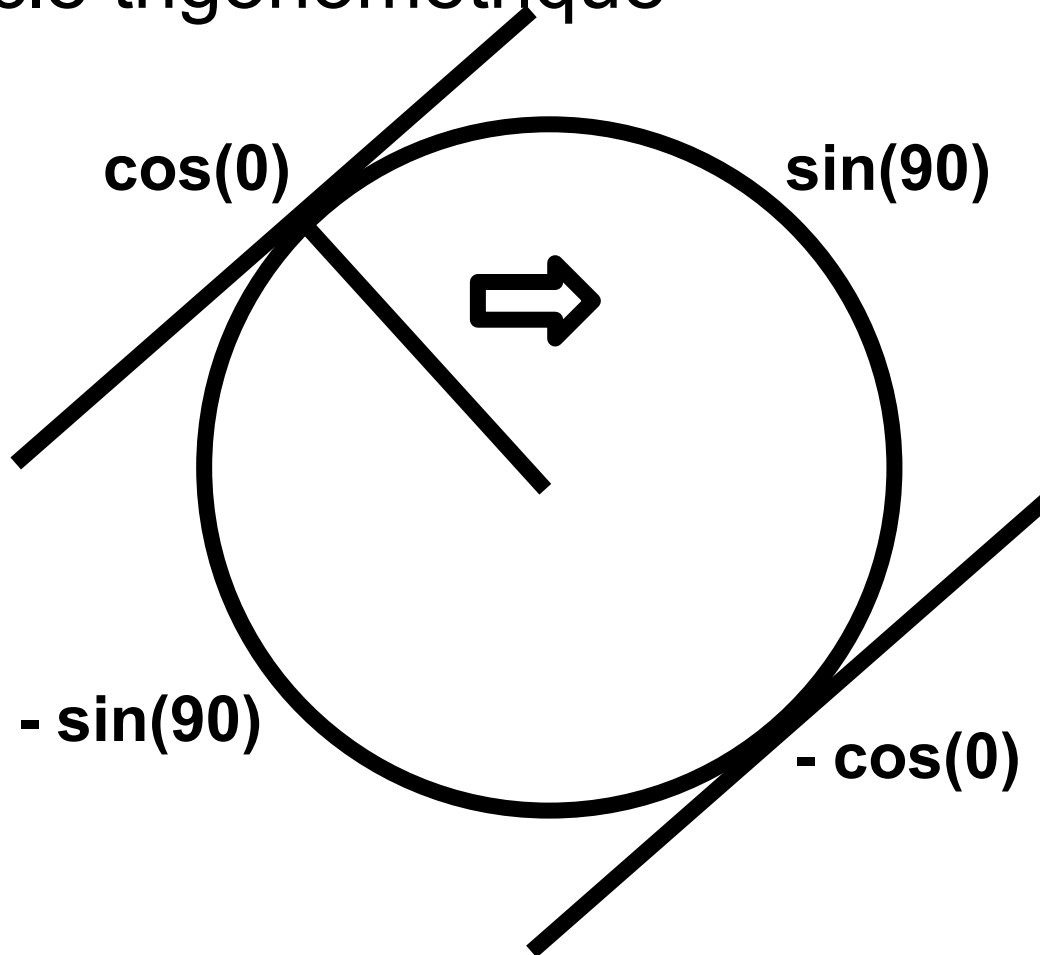
$$\cos(90) = ?$$

$$\sin(90) = ?$$

# OpenGL : Trigonométrie

---

Cercle trigonométrique

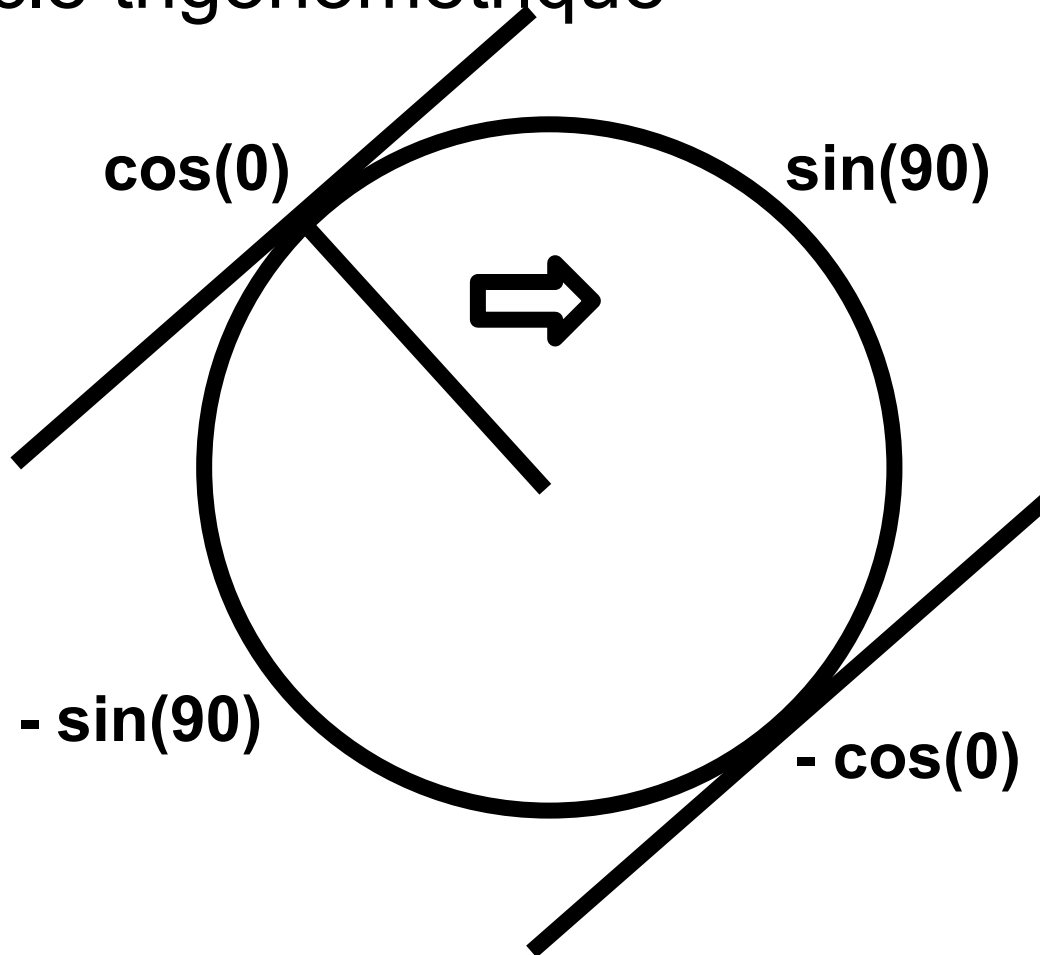


$\cos(0) = ?$   
 $\sin(0) = ?$   
 $\cos(90) = ?$   
 $\sin(90) = ?$

# OpenGL : Trigonométrie

---

Cercle trigonométrique



$$\cos(0) = ?$$

$$\sin(0) = ?$$

$$\cos(90) = ?$$

$$\sin(90) = ?$$

$$\arccos([-1:1]) = [180:0]$$

$$\arcsin([-1:1]) = [-90:90]$$

# OpenGL : Trigonométrie

---

## Fonctions trigonométrique

$\sin(a) = \text{opposé} / \text{hypoténuse}$

$\cos(a) = \text{adjacent} / \text{hypoténuse}$

$\tan(a) = \text{opposé} / \text{adjacent}$

---

# OpenGL : Trigonométrie

---

## Fonctions trigonométrique

$\sin(a) = \text{opposé} / \text{hypoténuse}$

$\cos(a) = \text{adjacent} / \text{hypoténuse}$

$\tan(a) = \text{opposé} / \text{adjacent}$

/!\ division par zéro

---



# OpenGL : Trigonométrie

---

## Fonctions trigonométrique

$\sin(a) = \text{opposé} / \text{hypoténuse}$

$\cos(a) = \text{adjacent} / \text{hypoténuse}$

$\tan(a) = \text{opposé} / \text{adjacent}$

/!\ division par zéro

/!\ range [0:90] seulement

---

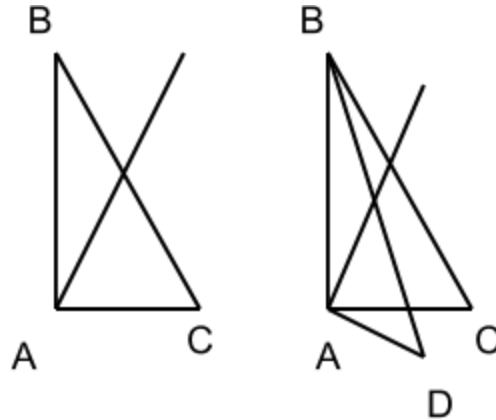
# OpenGL : Trigonométrie

---

Pythagore

$$AB^2 + AC^2 = BC^2$$

$$AB^2 + AC^2 + AD^2 = ?$$



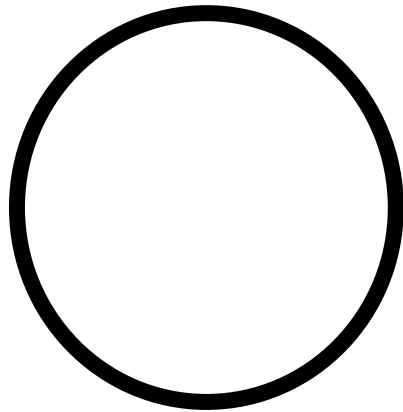
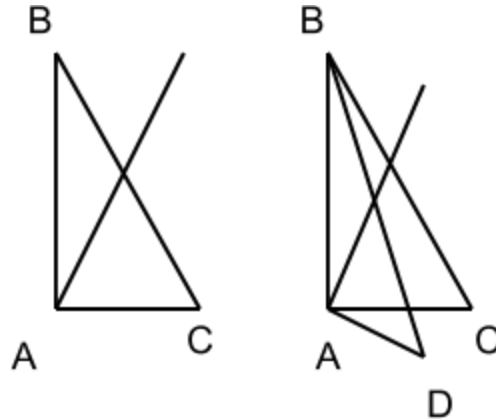
# OpenGL : Trigonométrie

---

Pythagore

$$AB^2 + AC^2 = BC^2$$

$$AB^2 + AC^2 + AD^2 = ?$$

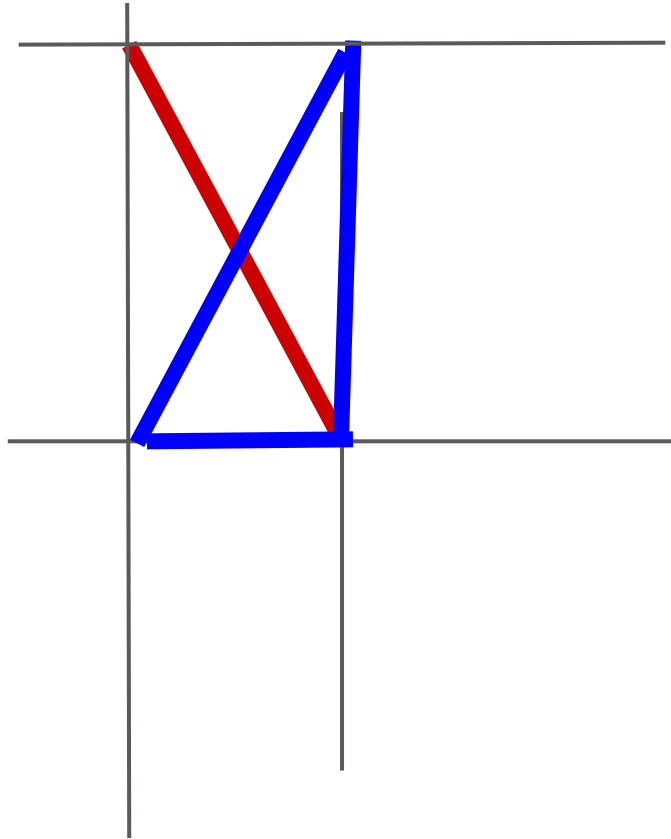


Cercle de centre  $(0,0)$  :  $x^2 + y^2 = r^2$

Sphère de centre  $(0,0,0)$  :  $x^2 + y^2 + z^2 = r^2$

# OpenGL : Trigonométrie

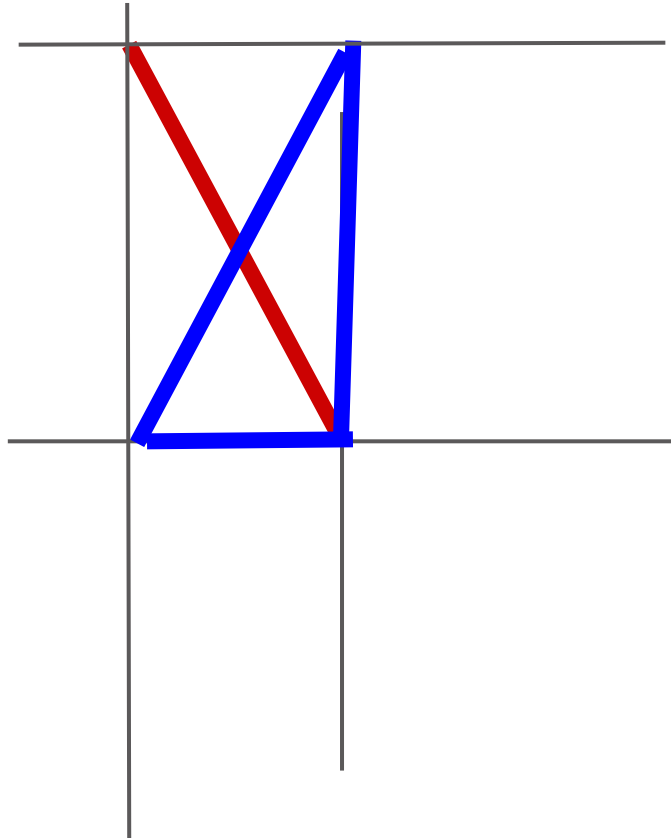
---



J'ai une coordonnée

# OpenGL : Trigonométrie

---

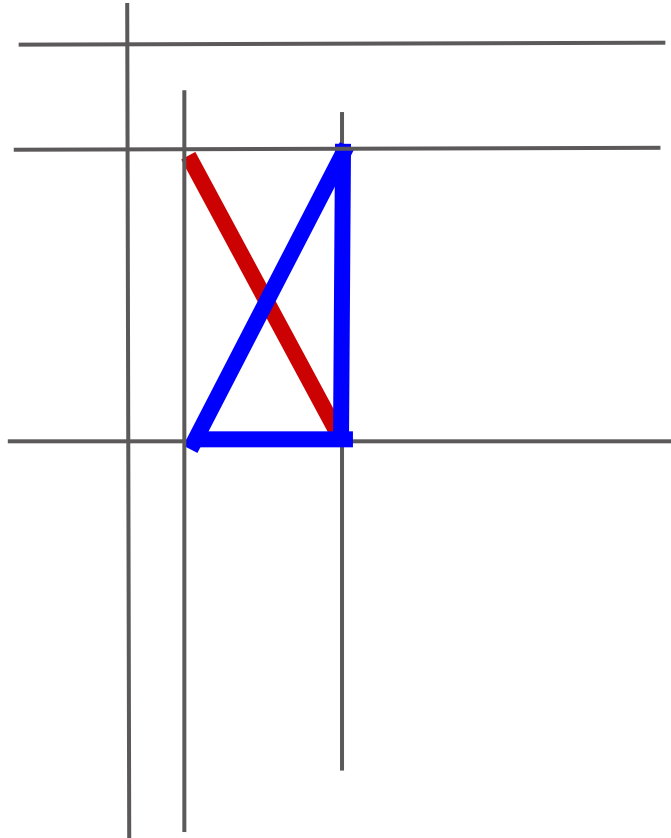


J'ai une coordonnée

- longueur de la normale ?
  - Pythagore

# OpenGL : Trigonométrie

---

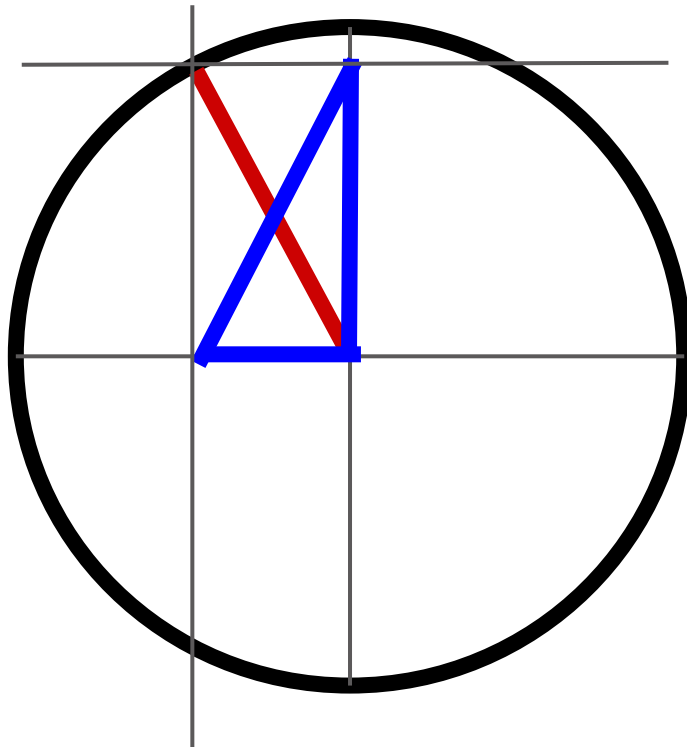


J'ai une coordonnée

- longueur de la normale ?
  - Pythagore
- normalize coordonnée
  - règle de trois

# OpenGL : Trigonométrie

---

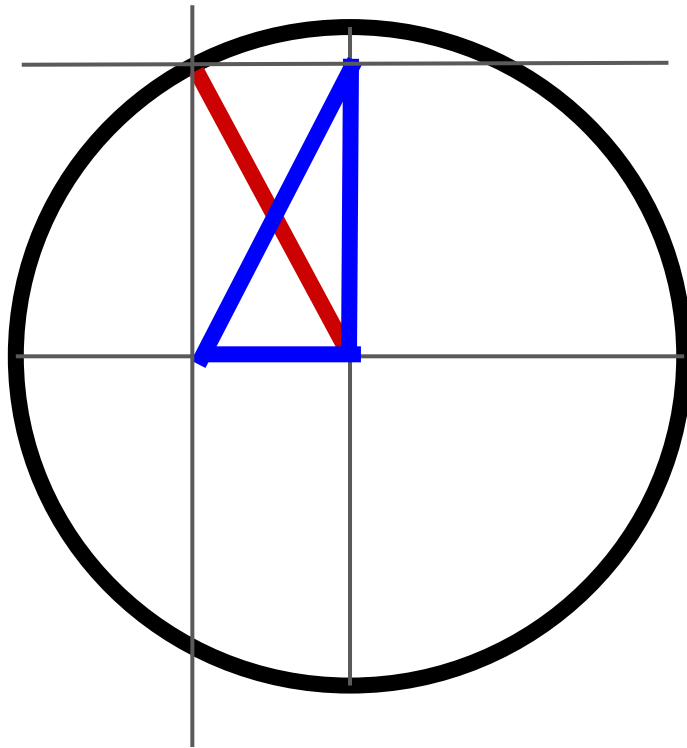


J'ai une coordonnée

- longueur de la normale ?
  - Pythagore
- normalize coordonnée
  - règle de trois
- je déduit l'angle
  - cercle trigo

# OpenGL : Trigonométrie

---



J'ai une coordonnée

- longueur de la normale ?
  - Pythagore
- normalize coordonnée
  - règle de trois
- je déduit l'angle
  - cercle trigo
- j'ai un doute ?
  - test fct trigo



# OpenGL : Trigonométrie

---

Exemple :

- Rotation d'une figure à la souris

# OpenGL : Trigonométrie

---

Solution 1 :

- Deux rotations suffisent

```
glRotatef(360.0f*(rotX+rotXDynamic), 0.0f,1.0f,0.0f);  
glRotatef(360.0f*(rotY+rotYDynamic), 1.0f,0.0f,0.0f);
```

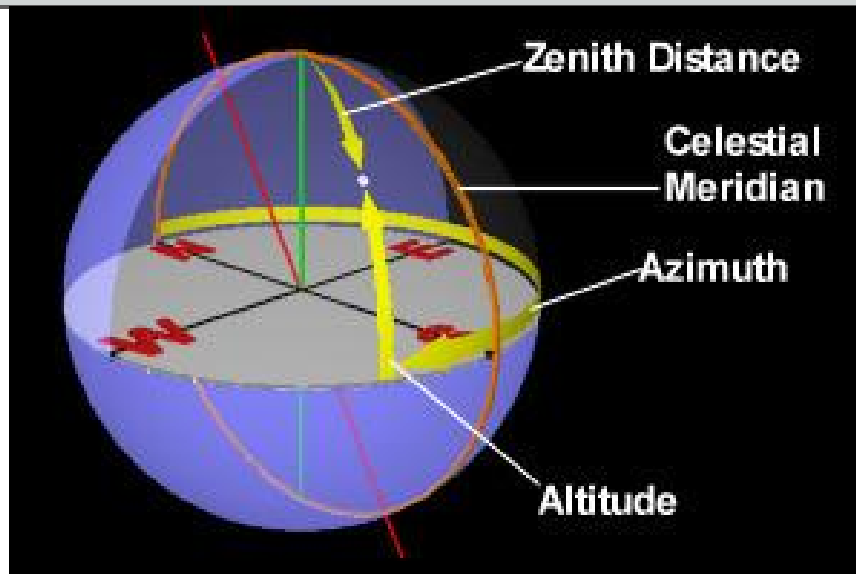
# OpenGL : Trigonométrie

---

Solution 1 :

- Deux rotations suffisent

```
glRotatef(360.0f*(rotX+rotXDynamic), 0.0f,1.0f,0.0f);  
glRotatef(360.0f*(rotY+rotYDynamic), 1.0f,0.0f,0.0f);
```



# OpenGL : Trigonométrie

---

Solution 2 :

- Deux rotations suffisent
- Reset du repère après chaque déplacement

```
void recDisplayUsingMouse(int r) {  
    if (r<=-1) {  
        exempleMesh();  
    } else {  
        glPushMatrix();  
        rotate(rotx[r], roty[r]);  
        recDisplayUsingMouse(r-1);  
        glPopMatrix();  
    }  
}
```

# OpenGL : Trigonométrie

---

Solution 2 :

- Deux rotations suffisent
- **Reset du repère** après chaque déplacement



Buffer pour chaque déplacement

Équivaut à un produit de chaque matrice de rotation

# OpenGL : Trigonométrie

---

Solution 3 :

- Rotation vers XY à la place de X and then Y
- Reset du repère après chaque déplacement

```
glRotatef(360.0f*(sqrt(x2+y2)),y/(|x|+|y|),x/(|x|+|y|),0.0f);
```

# OpenGL : Trigonométrie

---

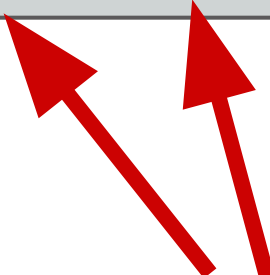
## Solution 3 :

- Rotation vers XY à la place de X and then Y
- Reset du repère après chaque déplacement

```
glRotatef(360.0f*sqrt(x2+y2),y/(|x|+|y|),x/(|x|+|y|),0.0f);
```



Pythagore pour la longueur de la normale  
**Nombre de degrés**



équilibre tel que la somme soit égale à 1  
**Axe de rotation**  
Remarque : y,x,0 fonctionne aussi

# A vous de jouer

---

- Dessiner un levier en 2D
    - Tourne sur un seul axe
    - Détecter la position via la souris
    - Effectuer la rotation via la souris
  - Dessiner un levier en 3D
    - Faire une projection orthogonal d'un plan à l'autre
      - Transformer un point d'un plan à l'autre
      - Transformer un vecteur d'un plan à l'autre
      - $\text{point} + \text{vecteur} = \text{droite}$
      - $\text{droite} + \text{plan} = \text{point projeté}$
-