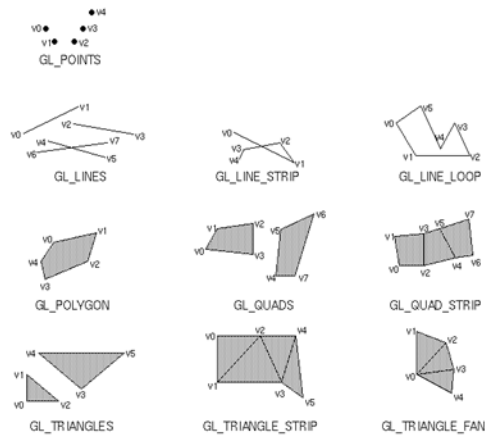


I. Formes de base

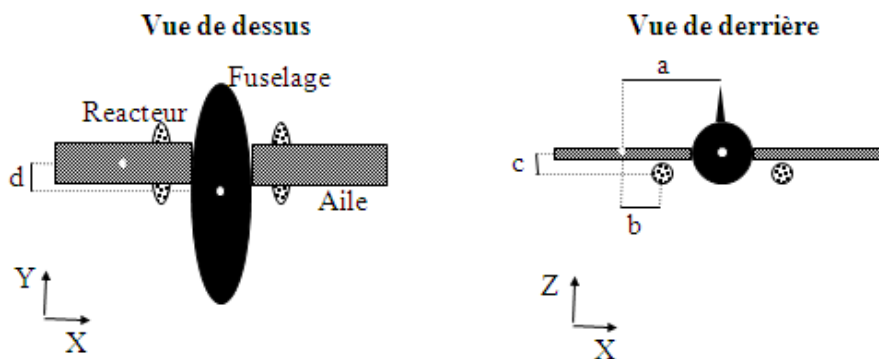
OpenGL

- `glBegin(GL_QUADS); //GL_POLYGON,`
etc.
`glVertex3f(1.0f, 1.0f, 1.0f);`
`glVertex3f(1.0f,-1.0f, 1.0f);`
`glVertex3f(1.0f,-1.0f,-1.0f);`
`glVertex3f(1.0f, 1.0f,-1.0f);`
- `glEnd();`



- 1) Afficher un cube en OpenGL
 - a. Avec 6 quadrilatères
 - b. Avec des TRIANGLE_STRIP
 - c. Avec une structure indexée (Indexed Face Set)
- 2) Afficher un cylindre et un cône
- 3) Afficher une sphère
- 4) Ajouter les normales et les coordonnées textures à ces 4 formes de base

II. Affichage à l'aide de la pile de matrices de transformation



On dispose des 2 fonctions :

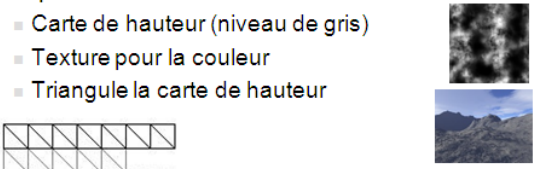
- dessineCube : dessine un cube centré en 0
- dessineSphere : dessine une sphère centrée en 0

Ecrire la procédure `dessineAvion` qui combine les formes de base en utilisant la pile de matrices de transformation de GL pour afficher un avion

III. Terrain

Terrain

- Représentation d'un terrain
 - Carte de hauteur (niveau de gris)
 - Texture pour la couleur
 - Triangle la carte de hauteur



Taille image	Nb triangles
64x64	8,192
128x128	32,768
256x256	131,072
512x512	524,288
1024x1024	2,097,152

Attention :
assez rapidement
beaucoup de triangles

Plaquage

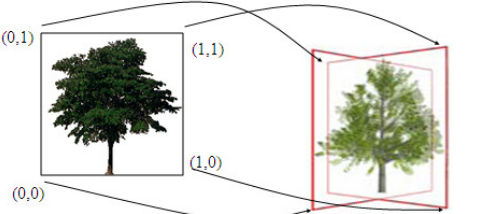
Image=texture

- A chaque sommet
 - Coordonnées textures (u,v)
 - (u,v) correspond à une position dans l'image (texture)

- 1) A partir d'une image interprétée comme une carte de hauteur, afficher les sommets du terrain correspondant
- 2) Transformer votre procédure précédente pour afficher les triangles formant le terrain
- 3) Ajouter le calcul de normal pour chaque sommet du maillage
- 4) Ajouter les coordonnées textures

Coordonnées textures U,V


- Exemple les billboards



- 5) Afficher un arbre représenté par un billboard
- 6) Afficher un ensemble d'arbres sur le terrain en utilisant la carte de hauteur pour les positionner (et glTranslatef)

Textures d'environnement : cubemap

- Un cube texturé par 6 images enveloppe la scène
 - Impression d'espace
 - Reflet de l'environnement sur l'objet



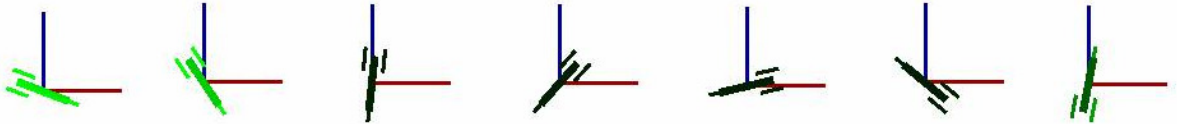
Terminator II, James Cameron, 1991

- 7) Afficher un cube texturé autour de votre scène

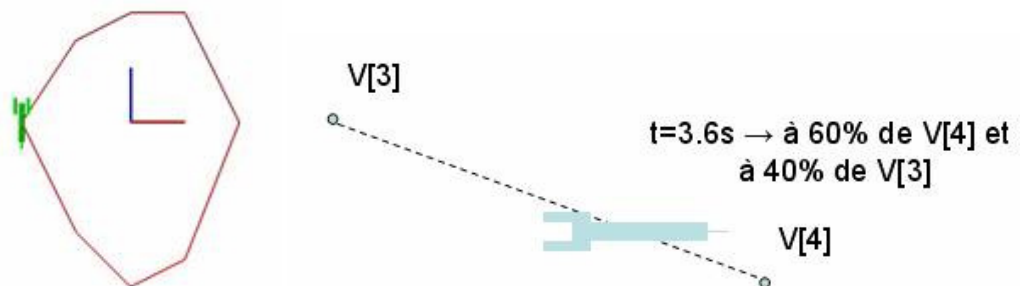
IV. Animation

La fonction système `clock()` renvoie le temps écoulé en nombre de click depuis le lancement du programme. Nous pouvons utiliser ceci pour définir la fonction temps en seconde :

```
float temps()
{
    return ((float)(clock())) / CLOCKS_PER_SEC;
}
```

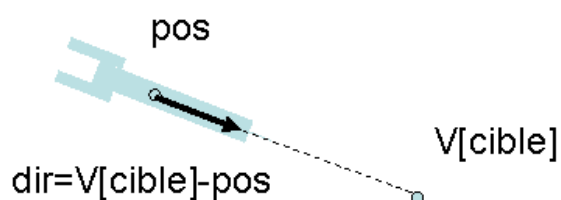


- 1) Ecrivez la procédure qui fait tourner la fusée sur elle-même en fonction du temps
- 2) Nous pouvons définir un tableau de points ($Vec V[NB]$) qui correspond à la trajectoire que devrait suivre la fusée. Pour simplifier, notre animation se déroulera dans le plan X,Z mais il est possible de généraliser à la 3D.



Ecrivez la fonction qui place la fusée sur la trajectoire en fonction du temps (ne pas considérer l'orientation de la fusée pour l'instant). Il faudrait qu'au $temps=3.6$, le jet soit entre le point $V[3]$ et le point $V[4]$ (plus exactement à 60% du point $V[4]$ et à $100-60=40\%$ du point $V[3]$). Oui, c'est une interpolation linéaire !

- 3) Nous connaissons la position du jet (pos), ainsi que sa direction de déplacement ($dir=V[cible]-pos$). Au repos, le jet est aligné avec $X=(1,0,0)$. Il faut donc trouver la matrice de passage faisant tourner le jet vers sa direction. Nous avons donc que $X \rightarrow dir$. Ici, Y ne change pas car les points V sont dans le plan X,Z. Pour trouver Z, nous pouvons faire $dir \times (0,1,0)$.



- 4) La direction du jet n'est pas continue. Pour rendre ses virages plus doux, nous pouvons également interpoler sa direction, ce qui revient en fait à calculer la tangente de la courbe.

