

**N.B. :**

Il y a (au moins) deux manières de faire de la 3D avec Processing :

- avec les commandes 3D du noyau Processing,
- avec la librairie OpenGL.

On utilise P3D (et non OpenGL) dans cette feuille d'exercices.

**Exo1 :** Rappelez ce qui différencie les deux méthodes de rendu (par facettage, par lancer de rayons).

Quels sont les éléments communs pour ces deux méthodes ?

**Corrigé :**

| Lancé de rayons (raytracing)              | Projection de facettes                |
|-------------------------------------------|---------------------------------------|
| Qualité                                   | Quantité                              |
| Temps de calcul important                 | Temps réel                            |
| Triangulisation non nécessaire            | Triangulisation obligatoire           |
| Proche des lois physiques de la lumière   | Modèles de lumière non physiques      |
| Calculs d'intersection droites/géométries | Projection de triangles               |
| Opérateurs booléens faciles et rapides    | Op. booléens approximatifs et coûteux |
| Description des objets                    |                                       |
| Composition de la scène                   |                                       |
| Texture mappings                          |                                       |
| Gestion de la caméra                      |                                       |
| ...                                       |                                       |

**Exo2 :** Etude de l'exemple Primitives 3D (en 3D core – P3D)

```
size(200, 200, P3D);
background(0);
lights();           // Lumière par défaut
noStroke();        // Pas de surlignage des frontières des objets
pushMatrix();
translate(47, height/2, 0);
rotateY(0.75);
box(50);
popMatrix();
pushMatrix();
translate(200, height/2, 0);
sphere(100);
popMatrix();
```

A quoi correspondent les différentes lignes de code ? Que fait ce programme ?

Comment éloigner les primitives géométriques à l'aide d'une seule translation ? (Processing utilise un repère main droite)

Comment rajouter une rotation automatique autour de l'axe Y ?

**Corrigé :**

Pour la signification des lignes de code, voir les commentaires dans le programme ci-dessous. Le programme affiche un cube et une sphère à des emplacements distincts de l'espace 3D (translation des objets).

Eloigner les 2 primitives peut se faire à l'aide d'une seule translation qui influera sur les deux. Avant la première sauvegarde de la matrice de transformations, on peut ajouter la translation commune qui sera ainsi récupérée lors du popMatrix qui intervient après le rendu de la première géométrie (le cube). Une telle translation sera donc prise en compte lors du rendu de la sphère.

Pour une rotation automatique, on peut utiliser le fait qu'à chaque nouvelle image, processing passe dans la méthode draw pour effectuer le rendu. A chaque passage, nous incrémentons donc l'angle de rotation pour qu'il augmente au fur et à mesure. Lorsque celui-ci dépasse la valeur  $2\pi$  nous le réinitialisons à 0.

```
float a = 0.0;          // L'angle de rotation

void setup() {          // Initialisation de la scène
  size(200, 200, P3D); // Taille fenêtre
  noStroke();           // Pas de surlignage des frontières des objets
}

void draw() {           // Rendu de la scène
  background(0);        // Efface la fenêtre avec la couleur noire
  lights();              // Création automatique d'une lumière ambiante

  a += 0.01;            // Incrément de l'angle de rotation
  if(a > TWO_PI) {      // Pour créer l'effet d'animation automatique
    a = 0.0;
  }

  pushMatrix();          // Sauvegarde de la matrice identité
  translate(0,0,-100);   // On « pousse les objets vers le fond »
  rotateY(a);            // On les oriente selon l'angle calculé

  pushMatrix();          // Sauvegarde des 2 transf. précédentes
  translate(47, height/2, 0); // Translation du cube
  rotateY(0.75);         // Rotation selon l'axe Y du cube
  box(50);               // Affichage cube
  popMatrix();           // Récupération de la matrice sauvegardée

  pushMatrix();          // Sauvegarde de la matrice (2 transf)
  translate(200, height/2, 0); // Translation de la sphère
  sphere(100);           // Affichage de la sphère
  popMatrix();           // Récup. De la matrice sauvegardée

  popMatrix();           // Récupération de la matrice identité
}
```

### Exo3 : Créer un relief avec Processing en mode P3D.

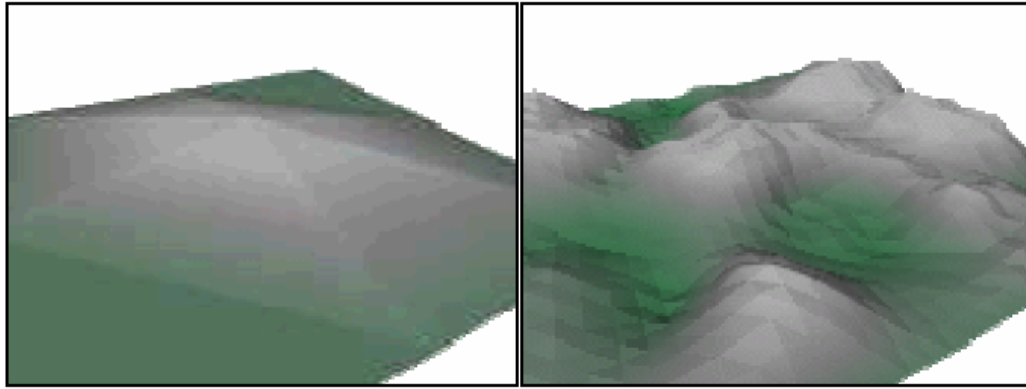
Le point de départ est une grille de 2x2 points se trouvant à la même altitude.

A chaque pas de subdivision, on utilise la grille précédente pour générer la nouvelle grille.

Pour chaque face de l'ancienne grille, on ajoute un sommet en son milieu. Chaque nouveau sommet est la moyenne des sommets l'encadrant + un aléa (random).

Pour une subdivision de profondeur i, donnez la taille de la matrice (en vertex et en face).

Coder cet algorithme en Processing avec la méthode P3D.



Comment faire ce même exercice sans stocker les informations sur les faces et/ou les sommets ? Donner le code Processing.

### Corrigé :

On utilise un générateur de nombre aléatoire. Si l'on utilise la même racine pour l'initialiser, on obtiendra alors la même série de nombres. Ainsi, on n'est pas obligé de sauvegarder les différentes faces puisqu'on sera les recrées grâce à la série de nombres.

Nous donnons ici la première étape de l'algorithme de subdivision. La deuxième étape consiste à gommer les ruptures visibles.

```
int subdiv = 0;
int racine;

class Sommet {
  float x, y, z;

  Sommet() {
    x = 0;
    y = 0;
    z = 0;
  }

  Sommet(float x, float y, float z) {
    this.x = x;
    this.y = y;
    this.z = z;
  }
}

Sommet [][] sommets = new Sommet[2][2];

void setup() {
  racine = second(); // Racine du générateur pseudo-aléatoire =
seconde système
  size(800, 800, P3D);

  // Init sommets
  sommets[0][0] = new Sommet(-50.0, 0.0, -50.0);
  sommets[0][1] = new Sommet( 50.0, 0.0, -50.0);
  sommets[1][1] = new Sommet( 50.0, 0.0,  50.0);
  sommets[1][0] = new Sommet(-50.0, 0.0,  50.0);
  noLoop();
}
```

```

    stroke(155);
}

void draw() {
    background(0);
    lights();
    translate(400,400,600);
    rotateX(-10*PI/180);
    noStroke();
    fill(255, 255, 255);

    randomSeed(racine); // La même racine pour obtenir la même série
    de valeurs
    dessinRelief(sommets[0][0], sommets[1][0], sommets[1][1],
sommets[0][1], 0, random(0, 10));
}

void dessinRelief(Sommet s1, Sommet s2, Sommet s3, Sommet s4, int
level, float rand) {
    // Dessin du relief
    if (level != subdiv) {
        print(level+": "+rand+ " ");

        // Récursivité = d'un quad on en crée 4

        //
        //      A1
        //      S1 +---*---+ S2
        //          |   |   |
        //      A4 *---G---* A2
        //          |   |   |
        //      S4 +---*---+ S3
        //          A3

        // 5 nouveaux sommets à créer (les 4 milieux d'arêtes + le
        centre de la face)
        Sommet a1 = new Sommet((s1.x+s2.x)/2,
                                (s1.y+s2.y)/2,
                                (s1.z+s2.z)/2);

        Sommet a2 = new Sommet((s2.x+s3.x)/2,
                                (s2.y+s3.y)/2,
                                (s2.z+s3.z)/2);

        Sommet a3 = new Sommet((s3.x+s4.x)/2,
                                (s3.y+s4.y)/2,
                                (s3.z+s4.z)/2);

        Sommet a4 = new Sommet((s4.x+s1.x)/2,
                                (s4.y+s1.y)/2,
                                (s4.z+s1.z)/2);

        Sommet g = new Sommet((s1.x+s2.x+s3.x+s4.x)/4,
                                (s1.y+s2.y+s3.y+s4.y)/4 - rand,
                                (s1.z+s2.z+s3.z+s4.z)/4);

        float rand1 = random(0.1*(s4.x-s1.x));

```

```

float rand2 = random(0.1*(s4.x-s1.x));
float rand3 = random(0.1*(s4.x-s1.x));
float rand4 = random(0.1*(s4.x-s1.x));

dessinRelief(s1, a1, g, a4, level+1, rand1);
dessinRelief(a1, s2, a2, g, level+1, rand2);
dessinRelief(g, a2, s3, a3, level+1, rand3);
dessinRelief(a4, g, a3, s4, level+1, rand4);
}
else {
    // On dessine : une face = 2 triangles
    beginShape(TRIANGLES);
    vertex(s1.x, s1.y, s1.z);
    vertex(s2.x, s2.y, s2.z);
    vertex(s3.x, s3.y, s3.z);

    vertex(s1.x, s1.y, s1.z);
    vertex(s3.x, s3.y, s3.z);
    vertex(s4.x, s4.y, s4.z);
    endShape();
}
}

void keyReleased() {
    if(key == '+' && subdiv < 8) {
        subdiv++;
        println("\nsubdiv = " + subdiv);
        redraw();
    }

    if(key == '-' & subdiv > 0) {
        subdiv--;
        println("\nsubdiv = " + subdiv);
        redraw();
    }

    if (key == ' ') {
        racine = second()*millis();
        subdiv=0;
        redraw();
    }
}
}
}

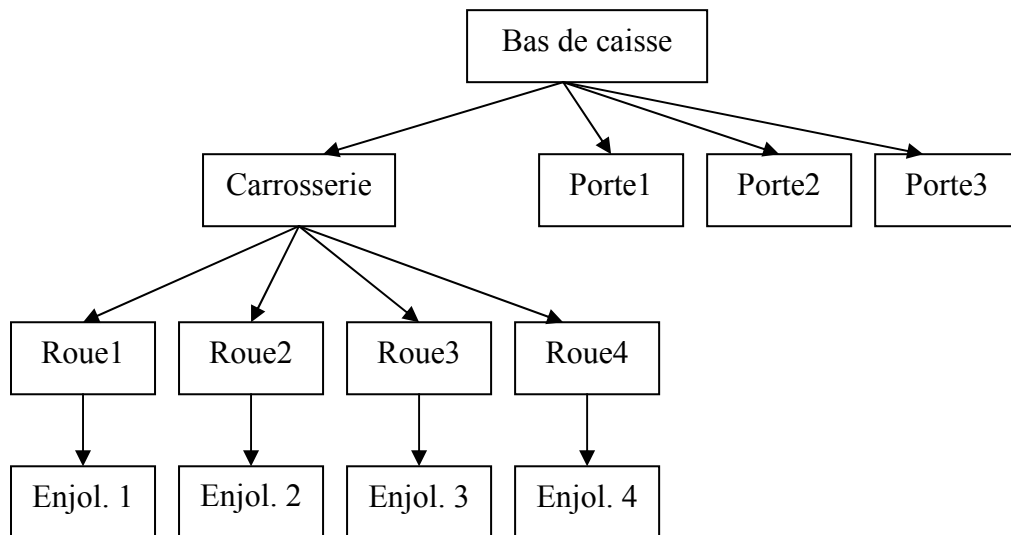
```

**Exo4** : Organiser les différents éléments d'une voiture dans un graphe de scène afin de profiter au maximum de la cascade de comportements. Les éléments à insérer sont le bas de caisse, la carrosserie, les portières, les roues, les enjoliveurs. Donner un pseudo code OpenGL permettant de dessiner cette voiture avec des transformations (utiliser pushMatrix et popMatrix).



### Corrigé :

On obtiendra le graphe suivant pour les éléments de la voiture à afficher. Il peut y avoir différents graphes de scène corrects.



### Signification d'un graphe de scène :

Prenons par exemple l'enjoliveur de la roue avant droite. Sa position et son orientation seront déterminées, si l'on suit le graphe de scène de la racine à la roue considérée, par les transformations appliquées au bas de caisse auxquelles s'ajoutent les transformations de la roue à auxquelles s'ajoutent ses propres transformations.

La traduction de ce graphe de scène en pseudo-code OpenGL sera la suivante (le code pour dessiner les objets n'est pas inclus) :

```

glTranslatef(position du bas de caisse);
dessine(as de caisse);

glPushMatrix(); // sauvegarde de la transformation bas de caisse
glTranslatef(position de la carrosserie);
dessine(carrosserie);

glPushMatrix(); // sauvegarde transformation caisse + carrosserie
glTranslatef(position roue1);
  
```

```

dessine(roue);
glTranslatef(position enjoliveur 1);
dessine(enjoliveur);
glPopMatrix(); // Récup. Transformation caisse + carrosserie

glPushMatrix(); // sauvegarde Transformation caisse + carrosserie
glTranslatef(position roue2);
dessine(roue);
glTranslatef(position enjoliveur 2);
dessine(enjoliveur);
glPopMatrix(); // Récup. Transformation caisse + carrosserie

glPushMatrix(); // sauvegarde Transformation caisse + carrosserie
glTranslatef(position roue3);
dessine(roue);
glTranslatef(position enjoliveur 3);
dessine(enjoliveur);
glPopMatrix(); // Récup. Transformation caisse + carrosserie

glPushMatrix(); // sauvegarde Transformation caisse + carrosserie
glTranslatef(position roue4);
dessine(roue);
glTranslatef(position enjoliveur 4);
dessine(enjoliveur);
glPopMatrix(); // Récup. Transformation caisse + carrosserie

glPopMatrix(); // Récup. Transformation caisse

glPushMatrix(); // sauvegarde transformation caisse
glTranslatef(position porte 1);
dessine(porte);
glPushMatrix(); // Récup. Transformation caisse

glPushMatrix(); // sauvegarde transformation caisse
glTranslatef(position porte 2);
dessine(porte);
glPushMatrix(); // Récup. Transformation caisse

glPushMatrix(); // sauvegarde transformation caisse
glTranslatef(position porte 3);
dessine(porte);
glPushMatrix(); // Récup. Transformation caisse

```