Introduction à la Rastérisation

Remplissage de polygone : le triangle

 —

La Rastérisation

Fait aussi bien référence ① au processus de transformations que la scène 3D subit que ② au processus transformant une forme vectorielle 2D en une grille matricielle (l'image). L'étape ② est aussi appelée rastérisation 🤯.

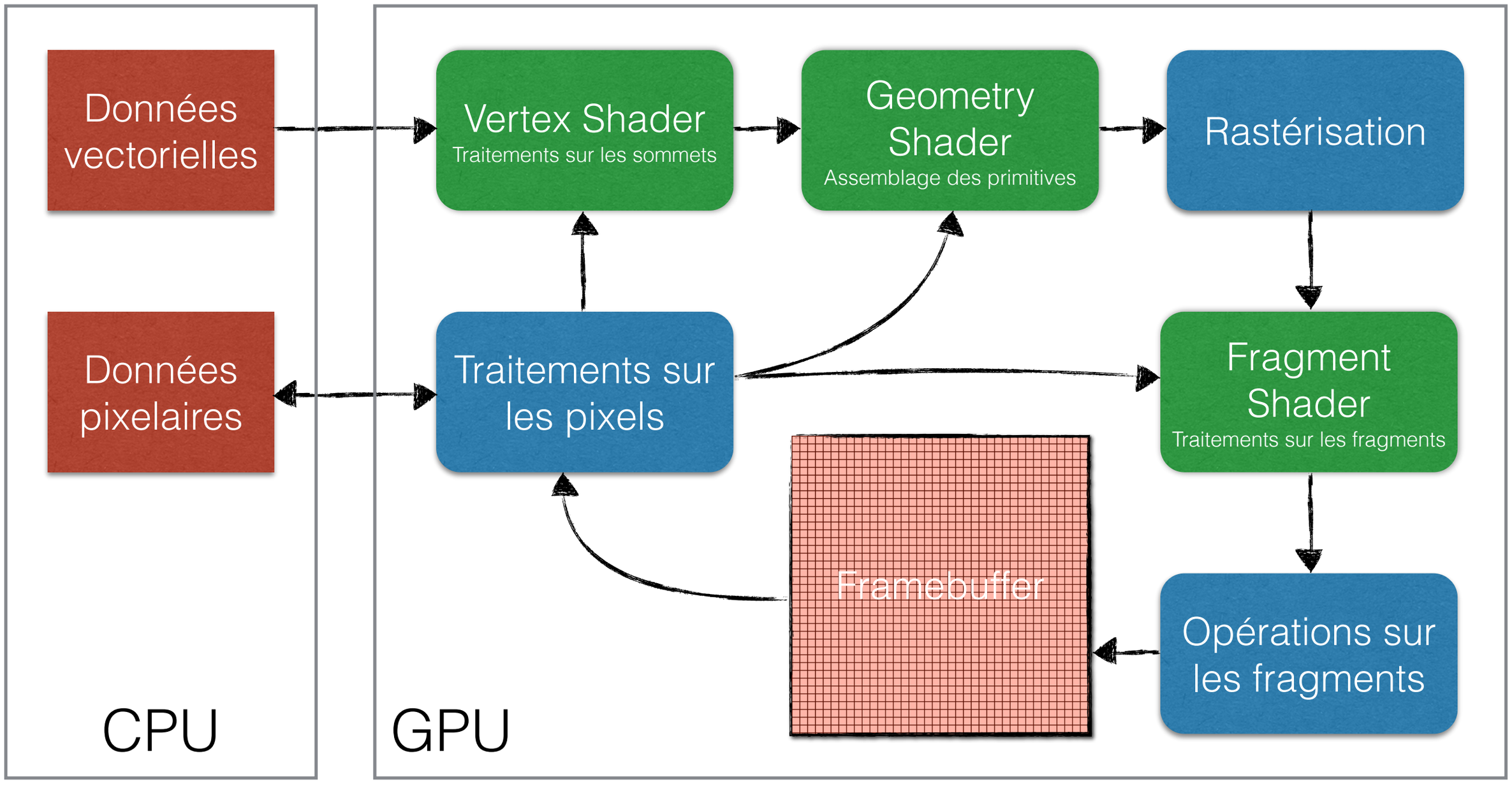
Pour une scène 3D représentant le point de vue de l'observateur, faire :

① projection → clipping (couper/rogner) → passage en coordonnées écran

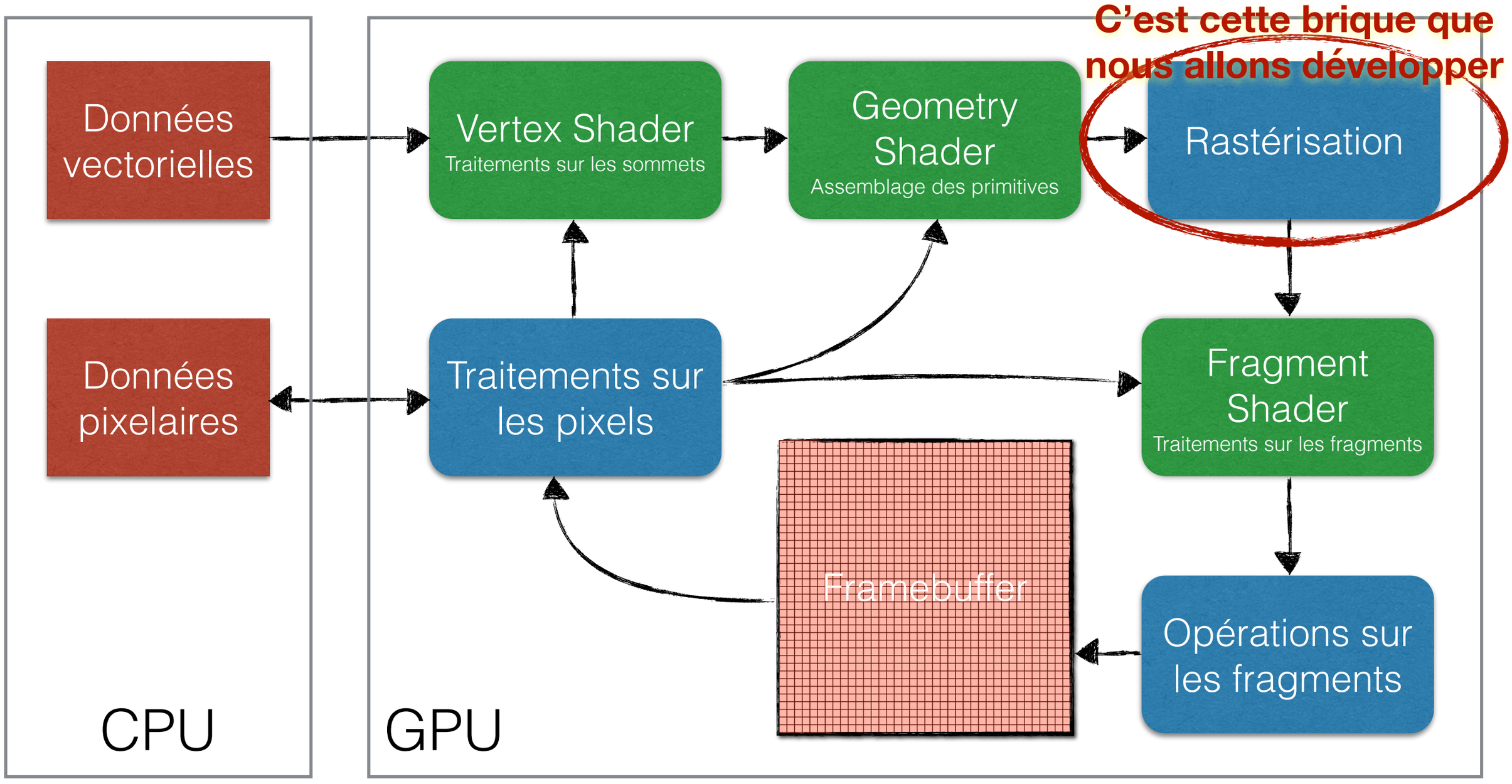
② « Matricialiser » les formes vectorielles 2D

Matricialiser = trouver les pixels qui représentent (approximent) la forme

S’inspirer du modèle (simplifié) OpenGL



S’inspirer du modèle (simplifié) OpenGL



S’inspirer du modèle (simplifié) OpenGL ?

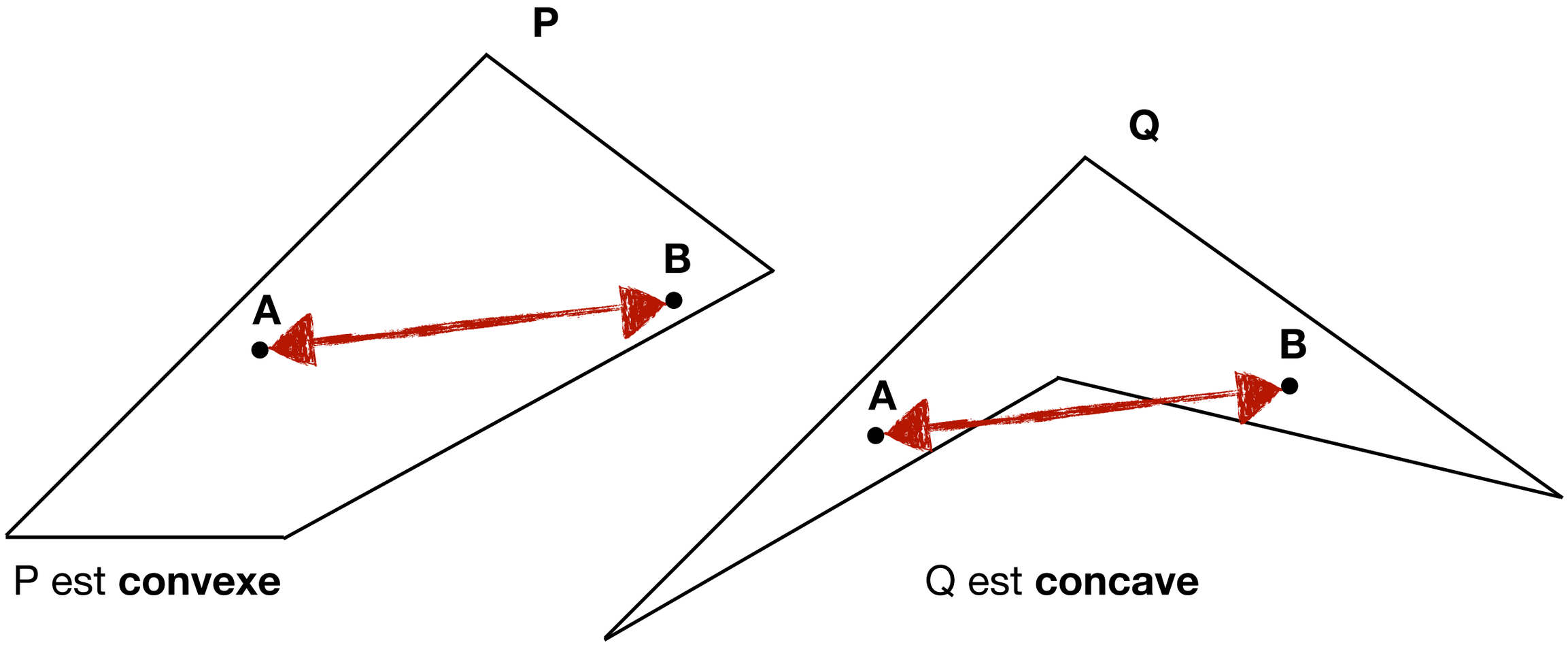
Données vectorielles → Une courbe devient une succession de segments
Une surface devient un assemblage de faces (polygones — forme plane)
(On parle souvent de maillage — cf. *mesh*)

Données pixelaires → Des images (ou textures dans ce cas)

Pour l’instant, on considère que les données vectorielles sont données en coordonnées écran (pas nécessairement dans les limites de l’écran).

Un polygone est constitué de sommets (plus tard … plus qu’une simple coordonnée)

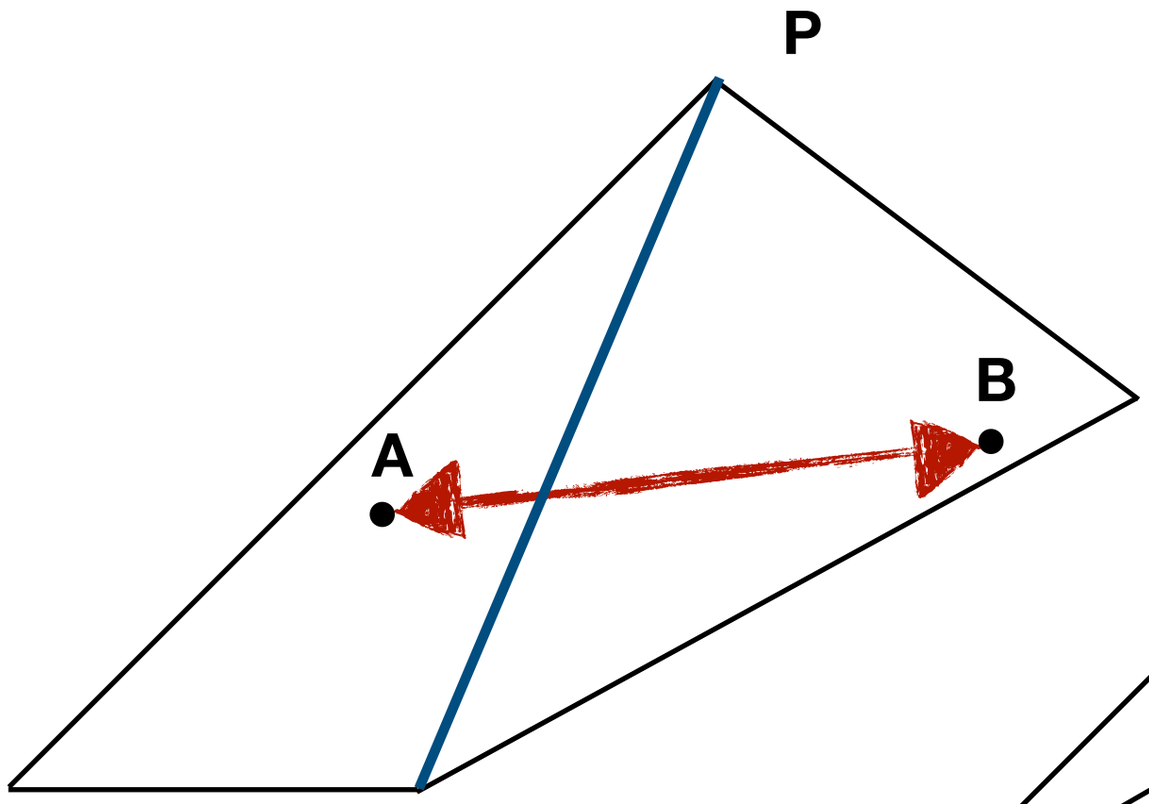
Exemples de polygones



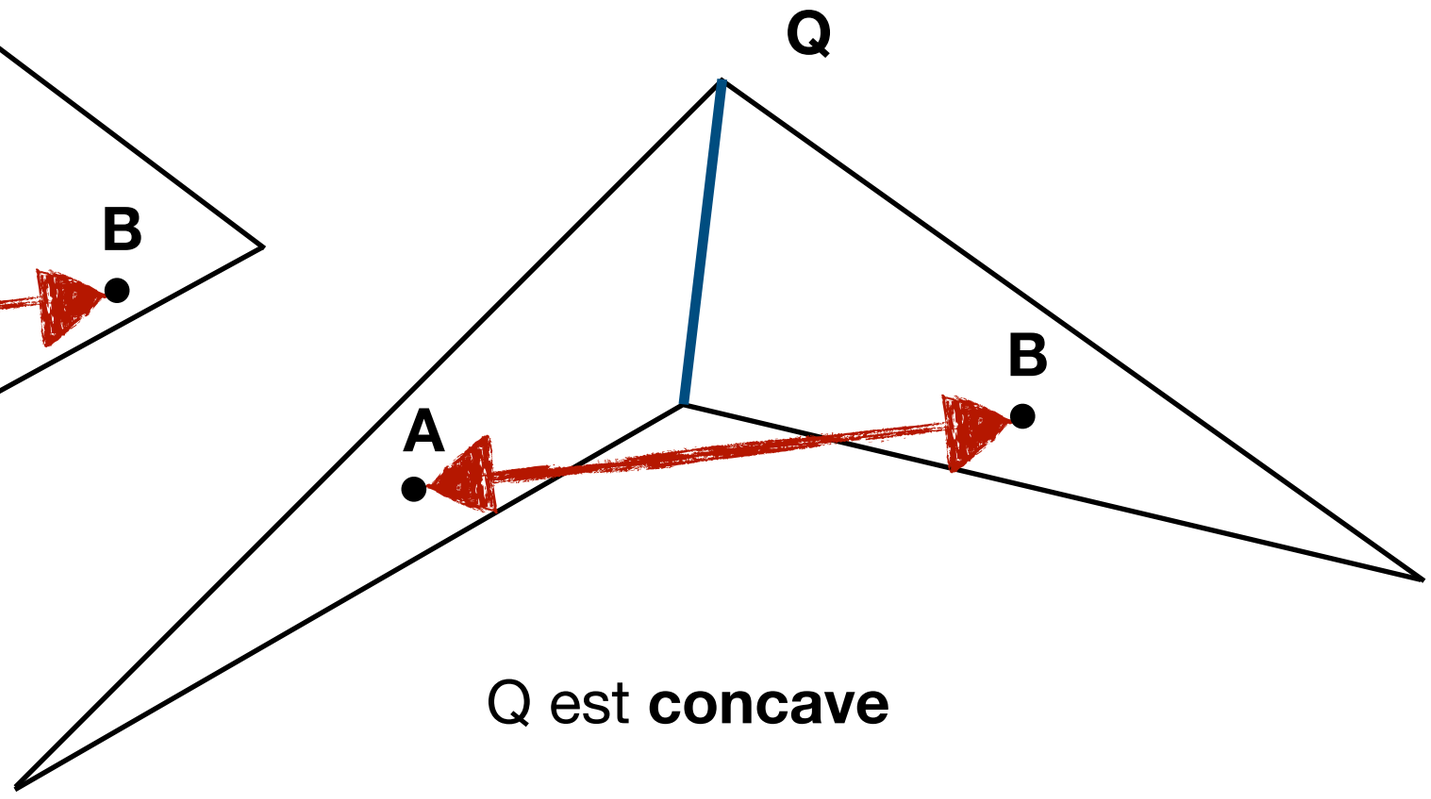
Convexité : Pour un polygone P et deux points A et B, le segment [AB] est dans le polygone P

$$\forall (A, B) \in P, [AB] \in P$$

On peut trianguler : les triangles sont toujours convexes



P est **convexe**



Q est **concave**

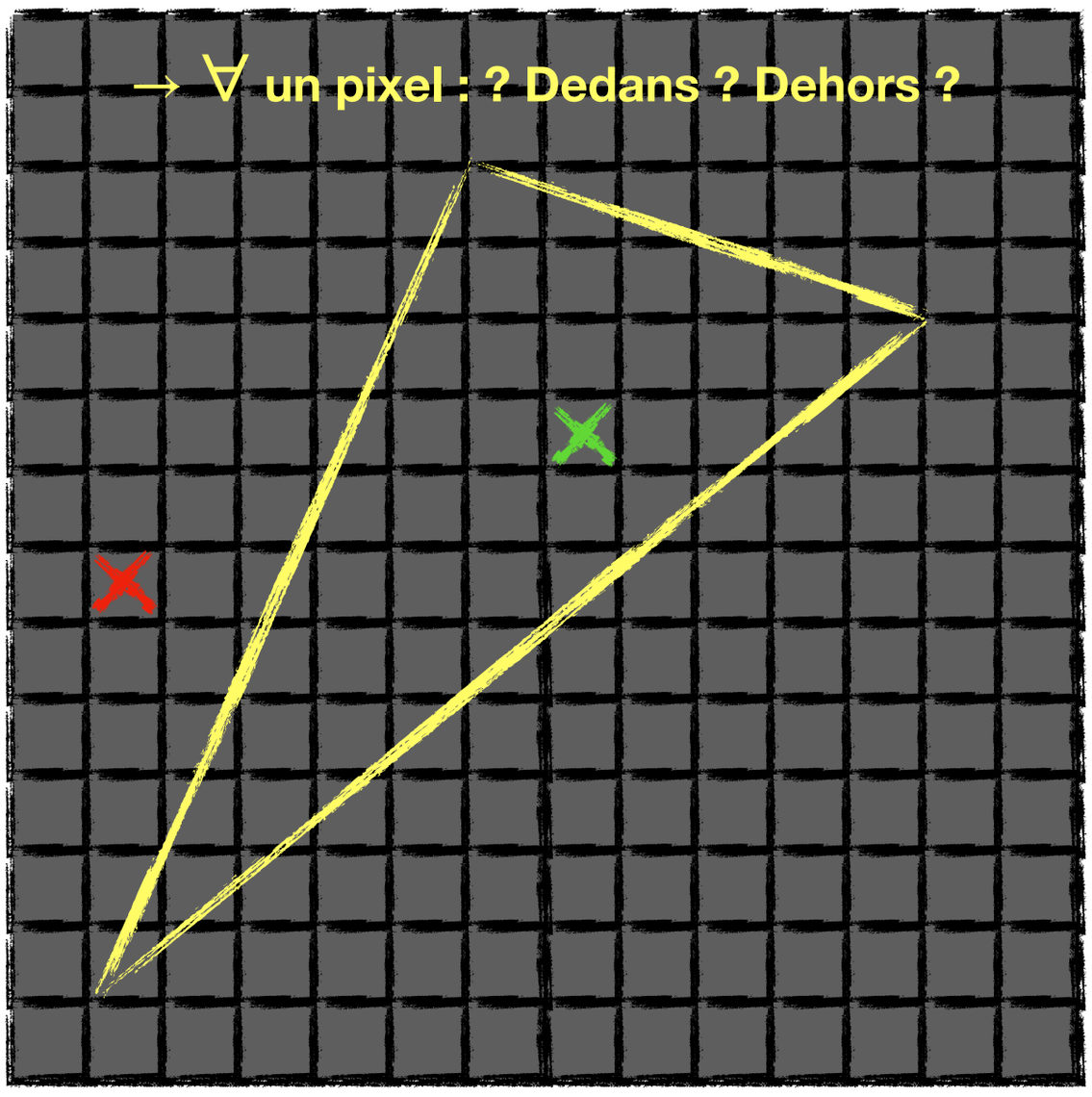
Convexité : Pour un polygone P et deux points A et B, le segment [AB] est dans le polygone P

$$\forall (A, B) \in P, [AB] \in P$$

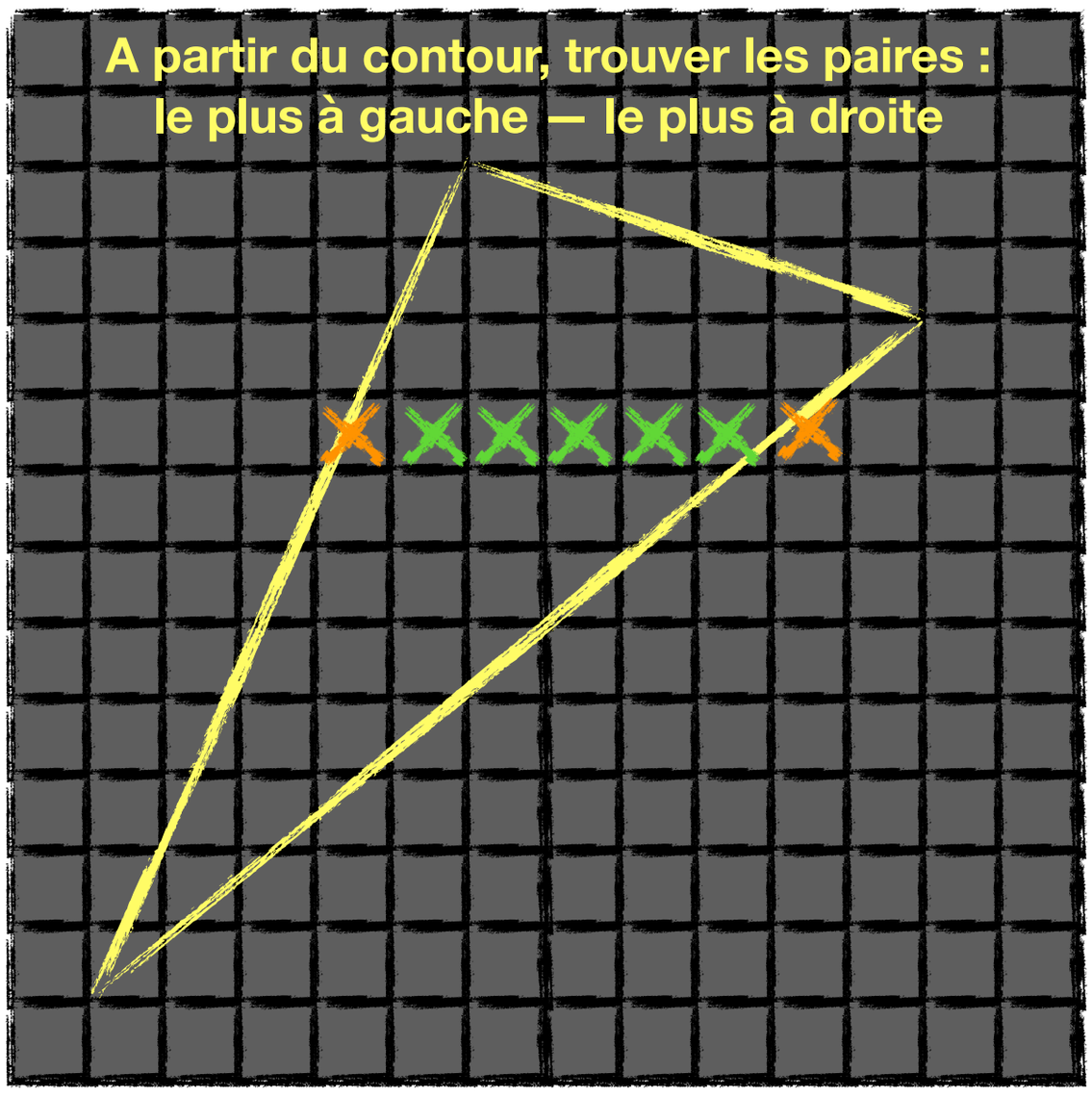
Rastériser un triangle

① Tracer les trois segments qui le constituent (facile / déjà fait précédemment)
② Dessiner un triangle plein

② (a) première option



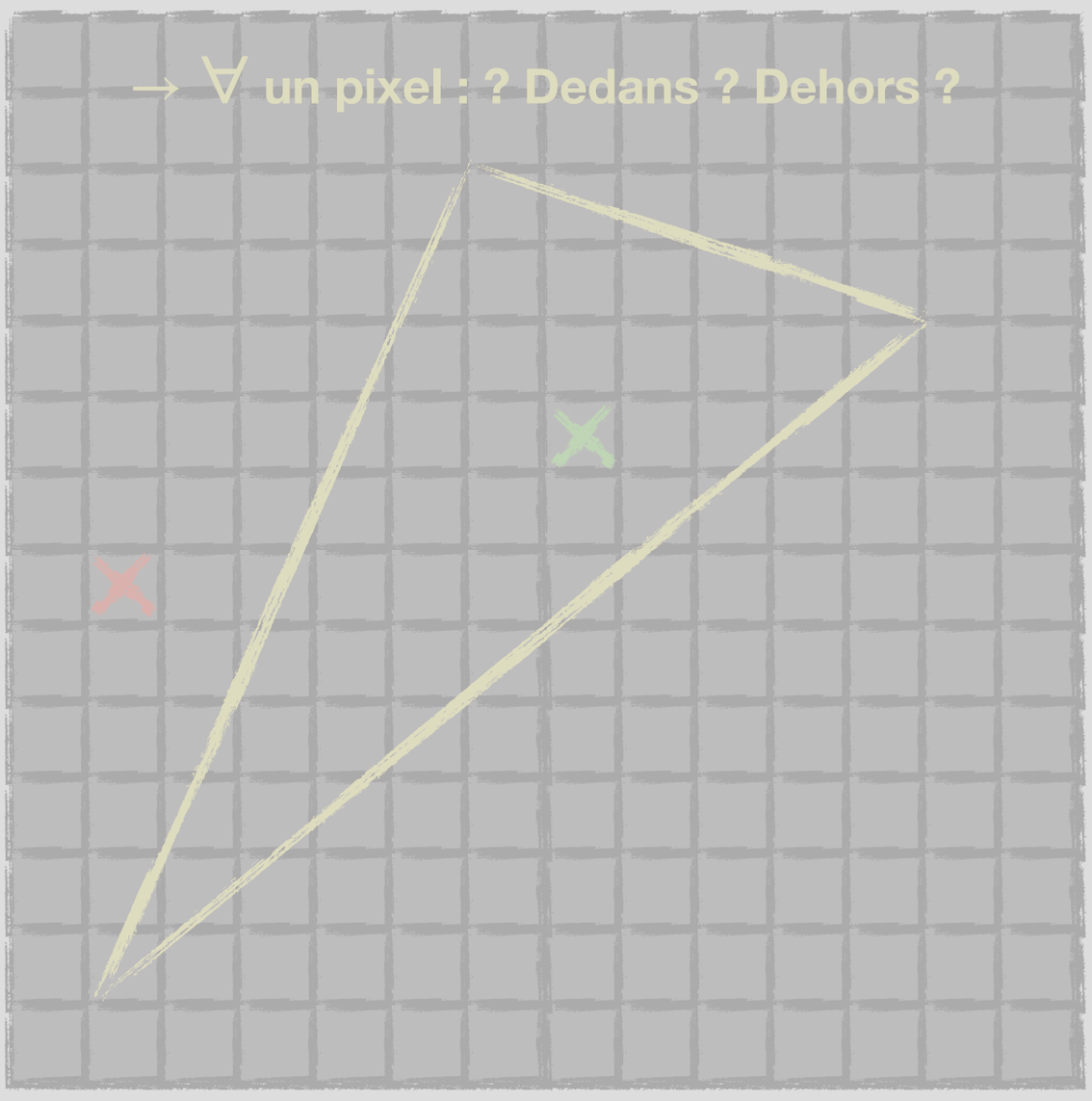
② (b) seconde option



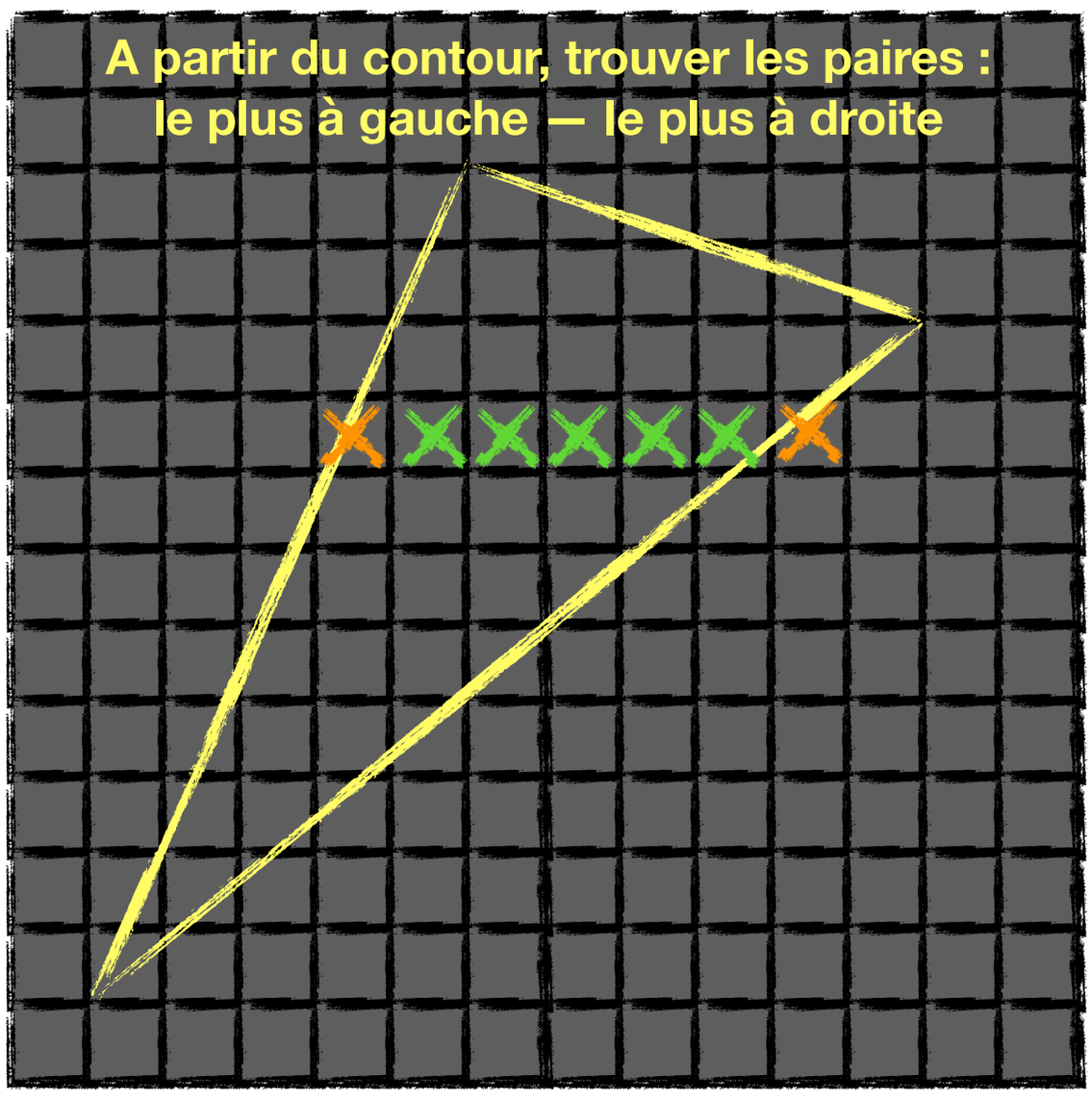
Rastériser un triangle, on choisit ② (b)

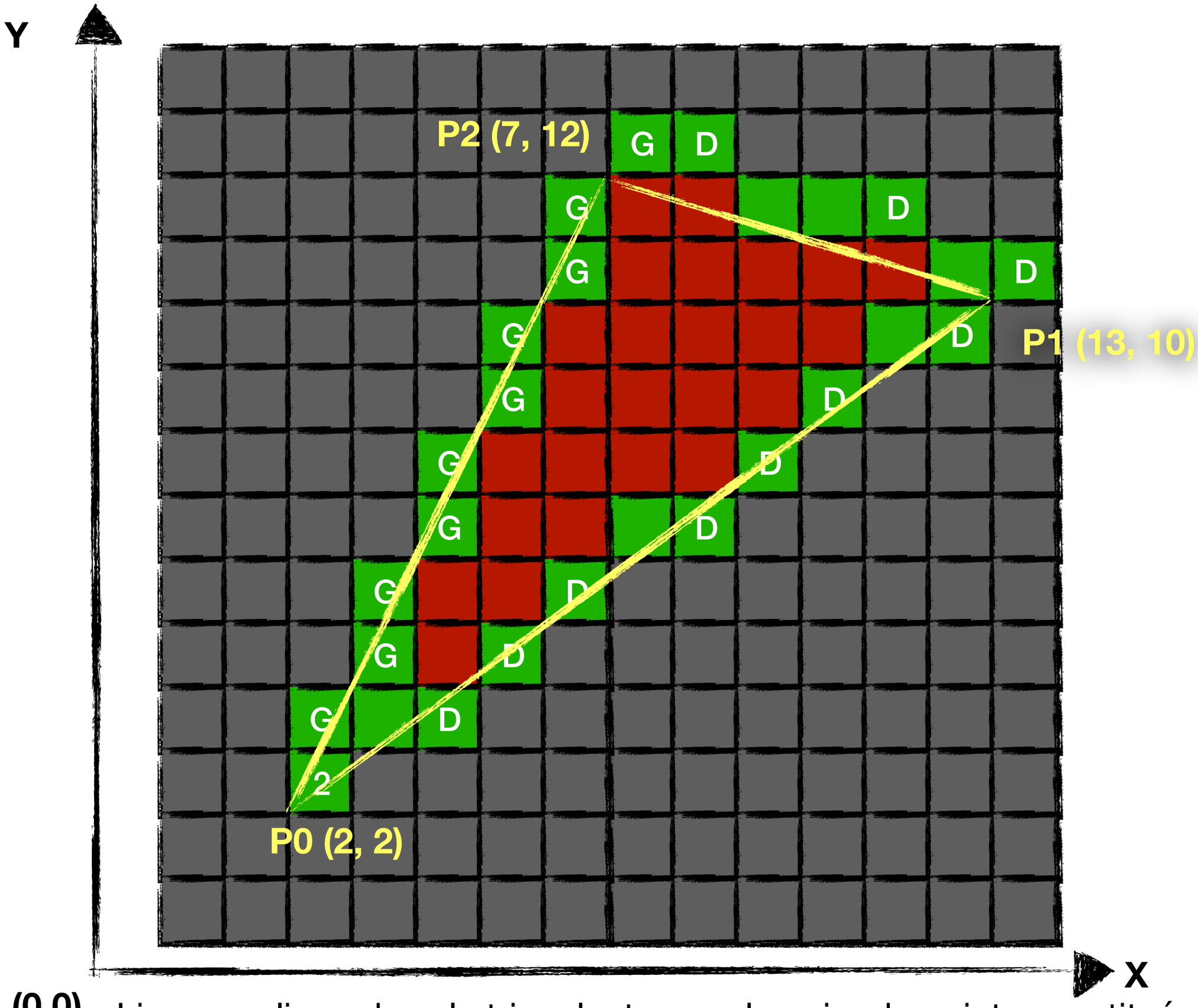
① Tracer les trois segments qui le constituent (facile / déjà fait précédemment)
② Dessiner un triangle plein

② (a) première option



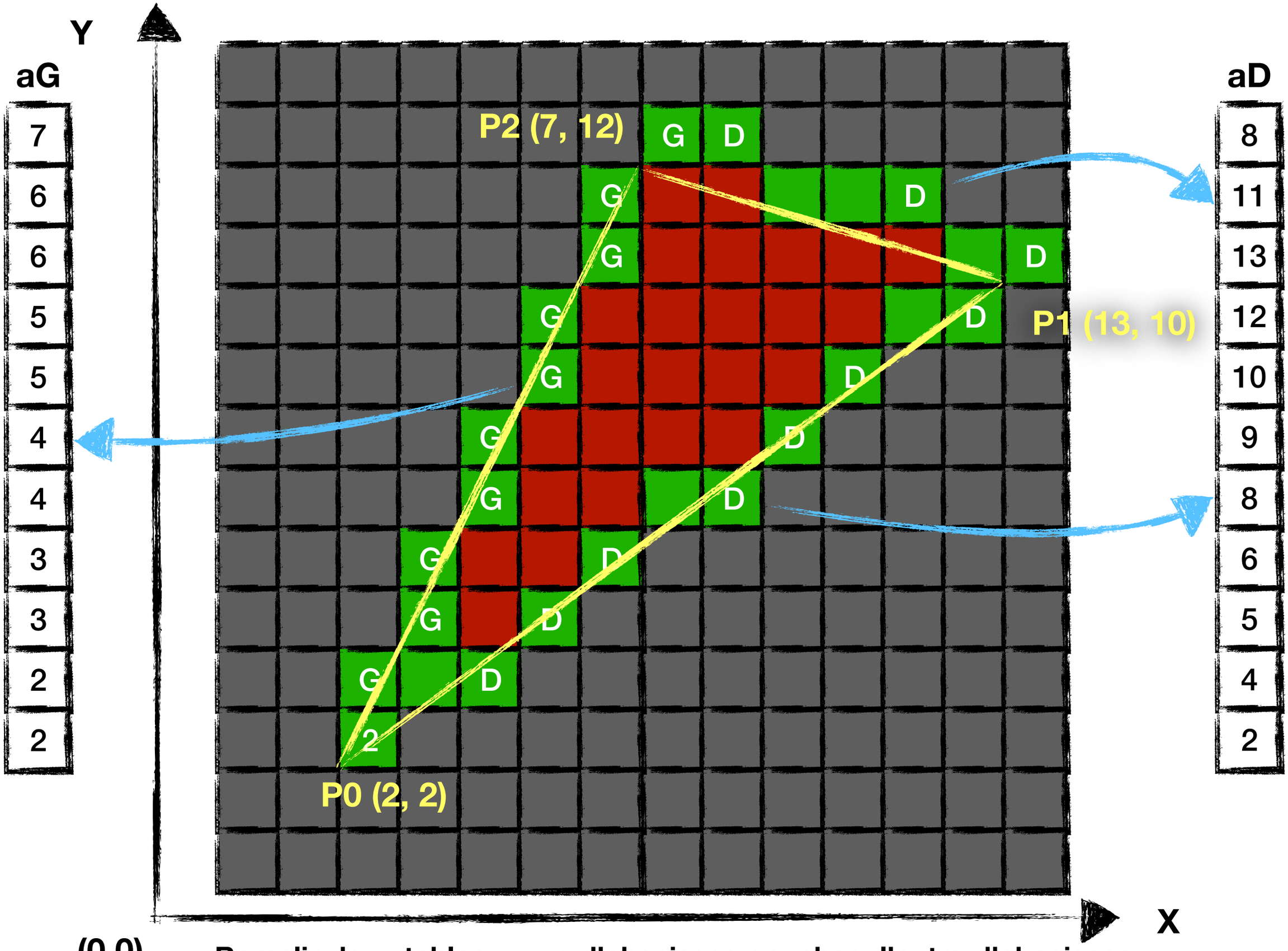
② (b) seconde option





(0,0)

Ligne par ligne dans le triangle, trouver la paire de points constituée du point le plus à gauche et du plus à droite, et les relier



Remplir deux tableaux, un d’abscisses gauches, l’autre d’abscisses droites et les relier entre-eux à l’aide de segments horizontaux.

Implémentation : les types

```c
typedef struct vertex_t vertex_t;
typedef struct triangle_t triangle_t;

/*!\brief le sommet et l'ensemble de ses attributs */
struct vertex_t {
   int x, y; /* coordonnée dans l’espace écran */
};

/*!\brief le triangle */
struct triangle_t {
    vertex_t v[3];
};
```

Implémentation : `fill_triangle` (1/3)

Déterminer le haut, bas et médian

```
void fill_triangle(triangle_t * t) {
 int bas, median, haut;
  if(t->v[0].y < t->v[1].y) {
    if(t->v[0].y < t->v[2].y) {
      bas = 0;
      if(t->v[1].y < t->v[2].y) {
        median = 1;
        haut = 2;
      } else {
        median = 2;
        haut = 1;
      }
    } else {
      bas = 2;
      median = 0;
      haut = 1;
    }
  } else { /* p0 au dessus de p1 */
    if(t->v[1].y < t->v[2].y) {
      bas = 1;
      if(t->v[0].y < t->v[2].y) {
        median = 0;
        haut = 2;
      } else {
        median = 2;
        haut = 0;
      }
    } else {
      bas = 2;
      median = 1;
      haut = 0;
    }
  }
//...
```

Implémentation : `fill_triangle` (2/3)

Trouver si la position (gauche/droite) du médian et remplir les tableaux d'abscisses gauches et d'abscisses droites

```
//...
  int signe, n = t->v[haut].y - t->v[bas].y + 1;
  vertex_t * aG = malloc(n * sizeof *aG);
  assert(aG);
  vertex_t * aD = malloc(n * sizeof *aD);
  assert(aD);
  /* est-ce que Pm est à gauche (+) ou à droite (-) de la droite (Pb->Ph) ? */
  /* idée TODO?, un produit vectoriel pourrait s'avérer mieux */
  if(t->v[haut].x == t->v[bas].x || t->v[haut].y == t->v[bas].y) {
    /* eq de la droite x = t->v[haut].x; ou y = t->v[haut].y; */
    signe = (t->v[median].x > t->v[haut].x) ? -1 : 1;
  } else {
    /* eq ax + y + c = 0 */
    float a, c, x;
    a = (t->v[haut].y - t->v[bas].y) / (float)(t->v[bas].x - t->v[haut].x);
    c = -a * t->v[haut].x - t->v[haut].y;
    /* on cherche le x sur la DROITE au même y que le median et on compare */
    x = -(c + t->v[median].y) / a;
    signe = (t->v[median].x >= x) ? -1 : 1;
  }
  if(signe < 0) { /* aG reçoit Ph->Pb, et aD reçoit Ph->Pm puis Pm vers Pb */
    abscisses(&(t->v[haut]), &(t->v[bas]), aG, 1);
    abscisses(&(t->v[haut]), &(t->v[median]), aD, 1);
    abscisses(&(t->v[median]), &(t->v[bas]), &aD[t->v[haut].y - t->v[median].y], 0);
  } else { /* aG reçoit Ph->Pm puis Pm vers Pb, et aD reçoit Ph->Pb */
    abscisses(&(t->v[haut]), &(t->v[bas]), aD, 1);
    abscisses(&(t->v[haut]), &(t->v[median]), aG, 1);
    abscisses(&(t->v[median]), &(t->v[bas]), &aG[t->v[haut].y - t->v[median].y], 0);
  }
//...
```

Implémentation : `fill_triangle` (3/3)

Tracer les segments horizontaux puis libérer la mémoire allouée

```
//...
  int h = gl4dpGetHeight();
  for(i = 0; i < n; ++i) {
    if( aG[i].y >= 0 && aG[i].y < h )
      hline(&aG[i], &aD[i]); /* modifier celle vue la semaine dernière */
  }
  free(aG);
  free(aD);
}
```

Implémentation : `abscisses` (1/2)

Premier octan, le plus complexe (revoir schéma)

```
void abscisses(vertex_t * p0, vertex_t * p1, vertex_t * absc, int replace) {
  int u = p1->x - p0->x, v = p1->y - p0->y, pasX = u < 0 ? -1 : 1, pasY = v < 0 ? -1 : 1;
  u = abs(u); v = abs(v);
  if(u > v) { // 1er octan
    if(replace) {
      int objX = (u + 1) * pasX;
      int delta = u - 2 * v, incH = -2 * v, incO = 2 * u - 2 * v;
      for (int x = 0, y = 0, k = 0; x != objX; x += pasX) {
        absc[k].x = x + p0->x;
        absc[k].y = y + p0->y;
        if(delta < 0) {
          ++k;
          y += pasY;
          delta += incO;
        } else
          delta += incH;
      }
    } else {
      int objX = (u + 1) * pasX;
      int delta = u - 2 * v, incH = -2 * v, incO = 2 * u - 2 * v;
      for (int x = 0, y = 0, k = 0, done = 0; x != objX; x += pasX) {
        if(!done) {
          absc[k].x = x + p0->x;
          absc[k].y = y + p0->y;
          done = 1;
        }
        if(delta < 0) {
          ++k;
          done = 0;
          y += pasY;
          delta += incO;
        } else
          delta += incH;
      }
    }
  }
//...
```

Implémentation : `abscisses` (2/2)

Second octan

```
//...
else { // 2nd octan
    int objY = (v + 1) * pasY;
    int delta = v - 2 * u, incH = -2 * u, incO = 2 * v - 2 * u;
    for (int x = 0, y = 0, k = 0; y != objY; y += pasY) {
      absc[k].x = x + p0->x;
      absc[k].y = y + p0->y;
      ++k;
      if(delta < 0) {
        x += pasX;
        delta += incO;
      } else
        delta += incH;
    }
  }
}
```

Et maintenant ?

Dégradé de couleurs, comment ajouter une texture ? …

Interpolation de couleurs => dégradé (ici une intensité pour une des composantes R, G ou B)

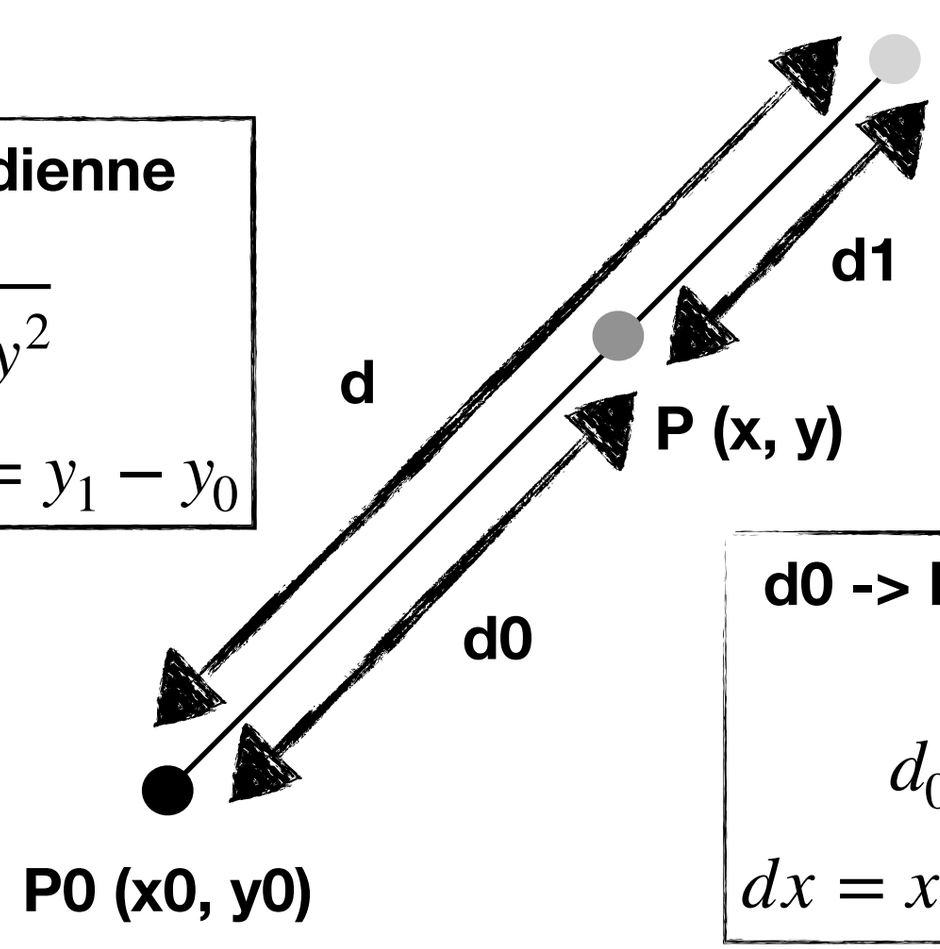
en affectant une couleur à chaque sommet, nous pouvons réaliser un dégradé en appliquant une interpolation bi-linéaire

Une au niveau des abscisses gauches et droite, puis une autre entre les deux lors du dessin de la ligne horizontale

d -> Distance euclidienne

$$d = \sqrt{dx^2 + dy^2}$$

$dx = x_1 - x_0$ **et** $dy = y_1 - y_0$



d = d0 + d1

d = d0 + (d - d0)

d0 -> Distance euclidienne

$$d_0 = \sqrt{dx^2 + dy^2}$$

$dx = x - x_0$ **et** $dy = y - y_0$

```
Intensité(P) = ?
Intensité(P) = (d1 / d) x Intensité(P0) + (d0 / d) x Intensité(P1)
Intensité(P) = ((d - d0) / d) x Intensité(P0) + (d0 / d) x Intensité(P1)
```

Nous avons vu

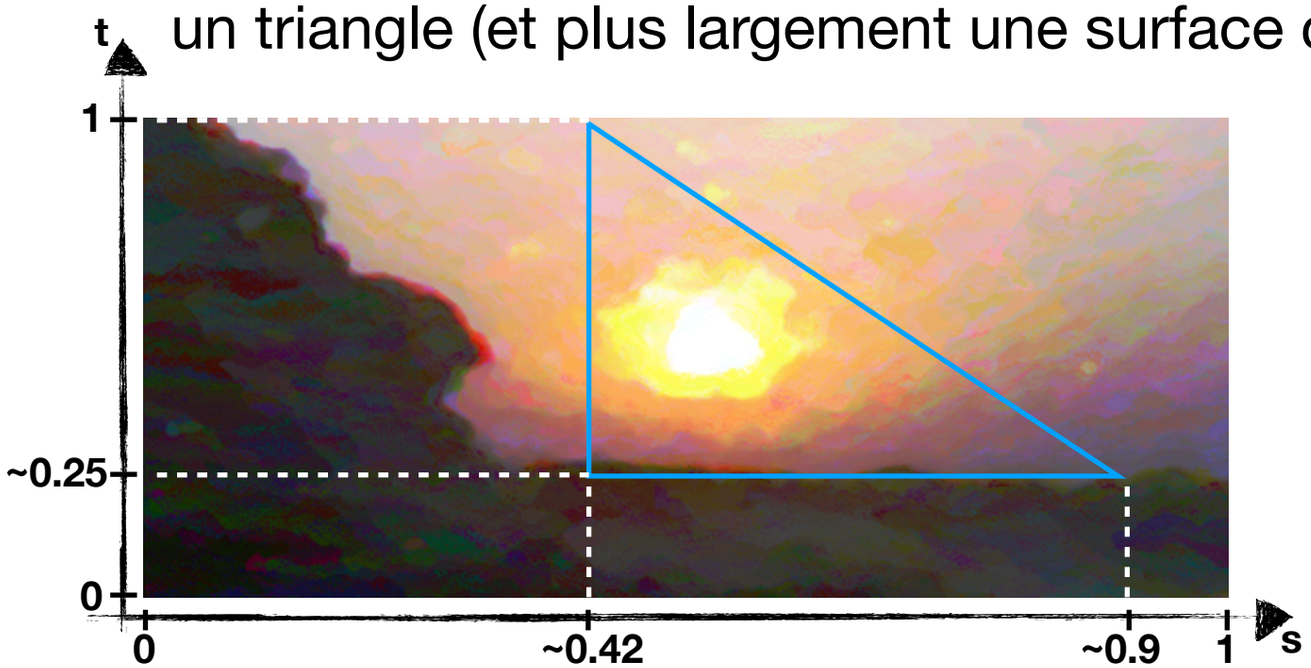
1. Comment remplir un triangle
2. Comment faire en sorte de réaliser un dégradé par interpolation bilinéaire (*slide* précédent). Ci-après une vidéo de 45min (enregistrée l'an dernier) illustrant la mise en pratique de la méthode proposée (merci de visionner, des questions ?) : https://youtu.be/2nagq2-bDvo

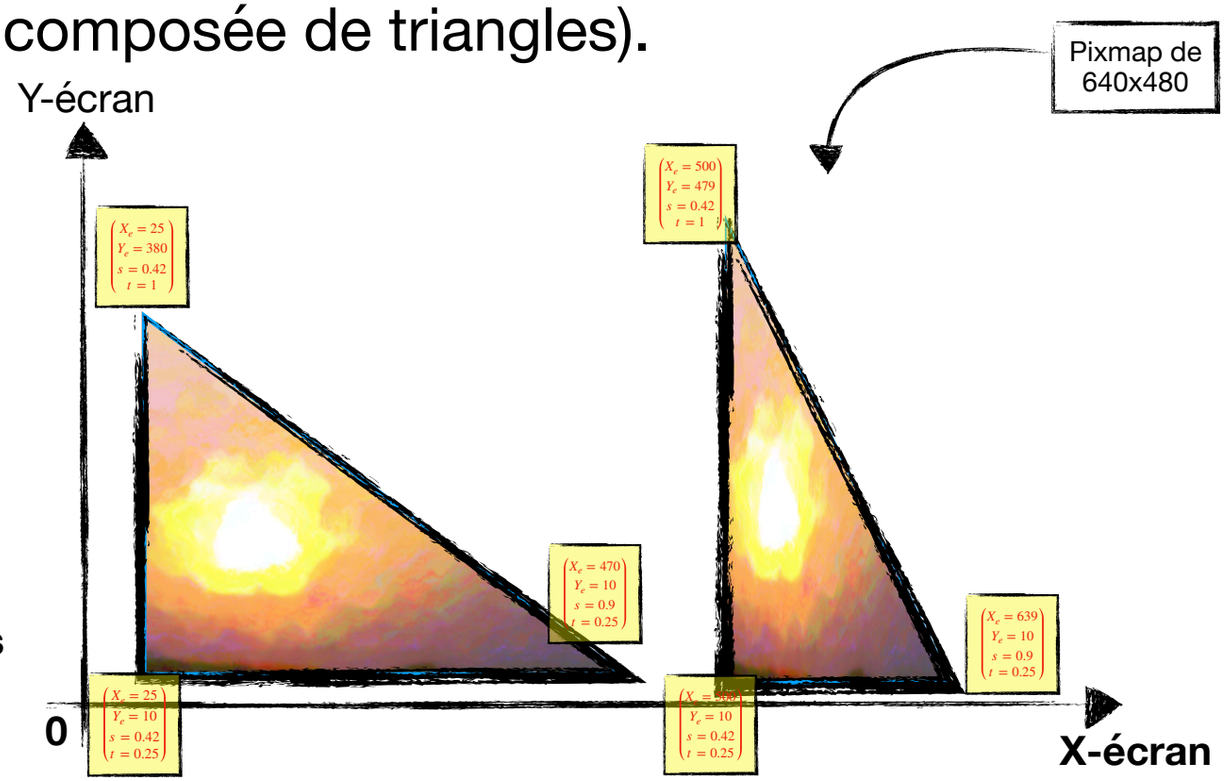
Ainsi, à l'issue de la séance

3. Nous aboutissons au code : https://expreg.org/amsi/C/APG2324S1/code/sc_00_04_triangle-0.2.tgz

Maintenant, passons à la texture

4. **À FAIRE** : ajoutez une coordonnée de texture (couple de flottants s et t) à la structure du vertex dans le code donné et appliquez-la au plaquage d'une texture associée à un triangle (et plus largement une surface composée de triangles).





Aller plus loin, quelques notions pour compléter le moteur de rendu par rastérisation

1. Un support à lire (D. Sobczyk & F. Belhadj) : https://expreg.org/amsi/C/APG2324S1/supports/notions_3D.pdf
2. Du sommet au pixel :
   1. Qu'est-ce qu'un sommet ? (Pas simplement une coordonnée dans l'espace objet)
   2. A quoi sert un vecteur normal ? Le produit scalaire ? Le produit vectoriel ?
   3. De l'espace objet à l'espace écran — toutes les notions
      1. La matrice Model
      2. La matrice Vue
      3. La matrice Projection
      4. Le *back face culling*
      5. Le *Clipping*
      6. Le *buffer* de profondeur (*z-buffer, depth-buffer, depth-map*)
      7. La correction de perspective

Aller plus loin, quelques notions pour compléter le moteur de rendu par rastérisation

- En 2023, avec chaque groupe de L2, nous avions implémenté la majeure partie des éléments d'un pipeline graphique CPU (pour la progression du développement, voir les README à partir des premières versions)

  1. Avec les L2-A, la Lib Claude : https://github.com/noalien/GL4Dummies/tree/master/sandbox/APG23_GRP_A

  2. Avec les L2-B, la Lib Ellule (libellule :)) : https://github.com/noalien/GL4Dummies/tree/master/sandbox/APG23_GRP_B