Projections 3d $\rightarrow$ 2d

- Il s'agit de transformer des points et des objets décrits dans l'espace (3 dimensions) en des points et des objets que l'on peut inclure dans un plan (2 dimensions) tout en gardant une impression visuelle de relief.

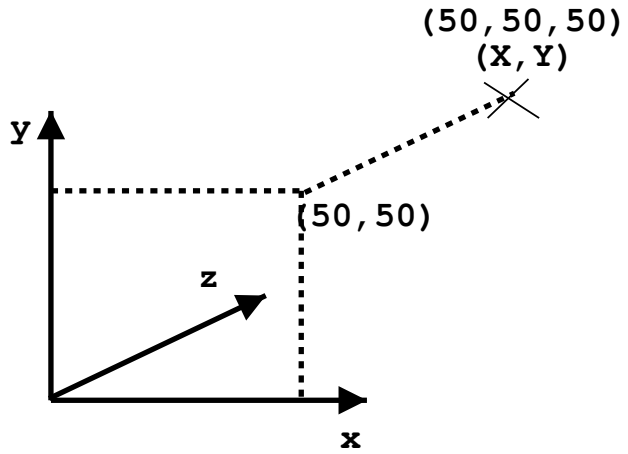
- La projection parallèle

- La projection isométrique

- La projection conique

Projection parallèle

- Un point (x, y, z) dans l'espace sera placé en (x, y) dans le plan, puis déplacé sur un axe proche de la première diagonale, proportionnellement à z. On est alors en (X, Y).



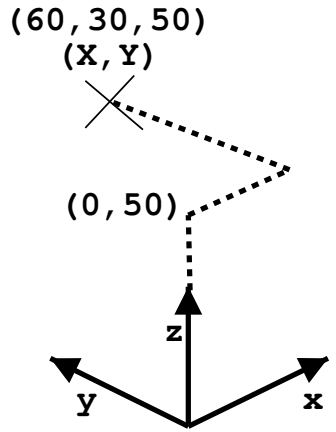
- On utilisera généralement:

$$X = x + cte \times z \text{ et } Y = y + \frac{cte}{2} \times z$$

avec cte une constante entre 0.5 et 1.

Projection isométrique

- Cette fois-ci, le point (x, y, z) est placé en $(0, z)$ sur le plan, puis déplacé sur 2 axes proches des 2 diagonales, et ce proportionnellement à x et à y.



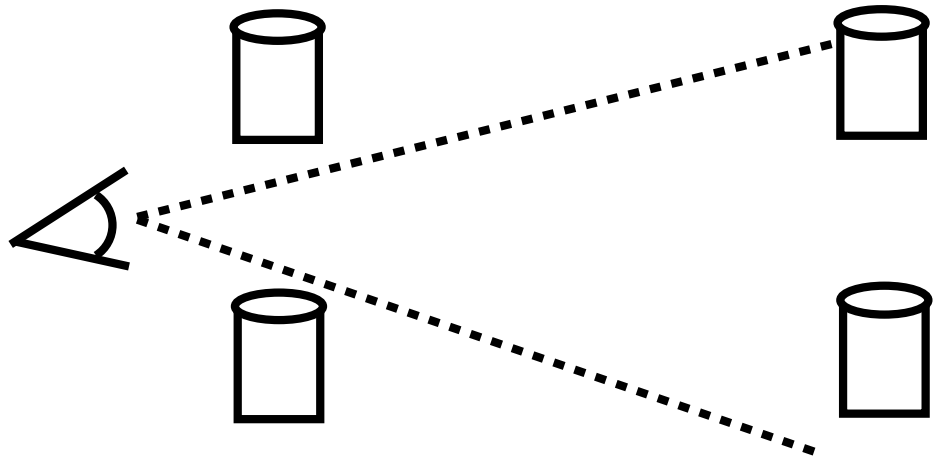
- On utilisera généralement:

$$X = cte1 \times x - cte2 \times y \text{ et } Y = z + \frac{cte1}{2} \times x + \frac{cte2}{2} \times y$$

avec $cte1$ et $cte2$ des constantes entre 0.5 et 1.

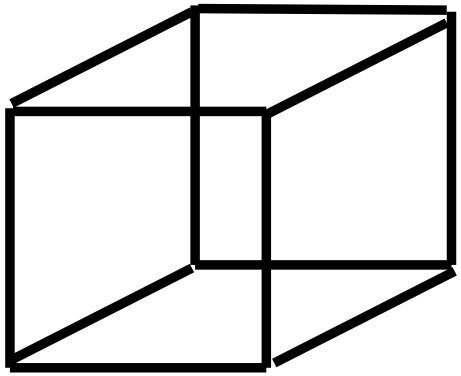
Projection conique

- C'est un type de projection plus proche de la réalité de notre vision. Plus l'on regarde au loin, plus l'on peut voir simultanément deux objets très éloignés.

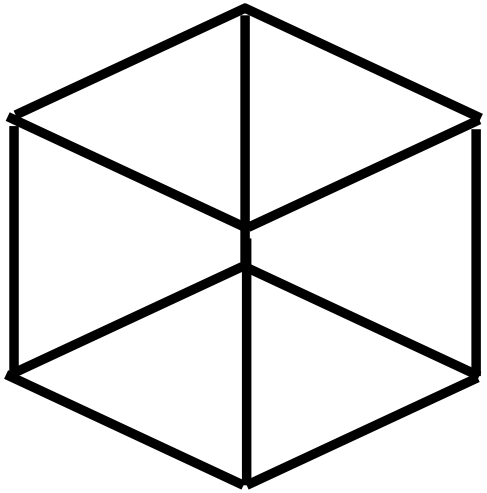


- Pas de formules, mais c'est un système de projection que l'on utilisera pour le RayCasting et le RayTracing. (l'approche ne sera pas la conversion 3d $\rightarrow$ 2d).

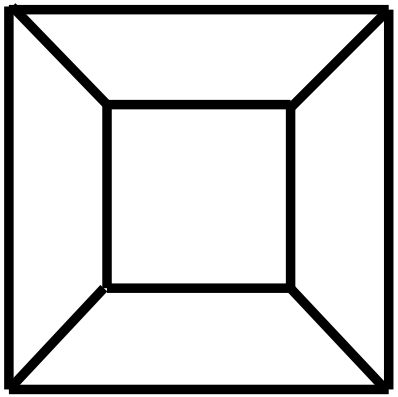
Projections d'un cube



Projection parallele



Projection isometrique



Projection conique

Ensemble de Mandelbrot

- Il s'agit d'un ensemble de points du plan ayant un aspect fractal, découvert par le matheux français Benoit Mandelbrot.

- On définit une fonction qui transforme un point (x, y) du plan en un autre point (x', y') du même plan:
  $x' = x^2 - y^2 + cte_x \qquad y' = 2xy + cte_y$
  Le nouveau point (x', y') est à son tour transformé par la même fonction.

- On part du point $(0, 0)$. Pour un couple (cte_x, cte_y) donné, on regarde si au cours des transformations successives on s'éloigne de l'origine $(0, 0)$.

- L'ensemble des points du plan (cte_x, cte_y) pour lesquels on ne s'éloigne pas de l'origine est l'ensemble de Mandelbrot.

Mandelbrot - suite

- Pour ceux qui se souviennent :
  L'ensemble de Mandelbrot est l'ensemble des cte complexes telles que la suite complexe $z_0 = 0, z_{n+1} = z_n^2 + cte$ ne diverge pas.

- Concrètement, l'ensemble de Mandelbrot est dans la zone du plan $(-2, 2), (-2, 2)$. On effectue un nombre fini d'itérations (100 est suffisant), et on regarde si la distance du point obtenu (x, y) au centre est supérieure à 5 par exemple. On se contentera de calculer $x^2 + y^2$.

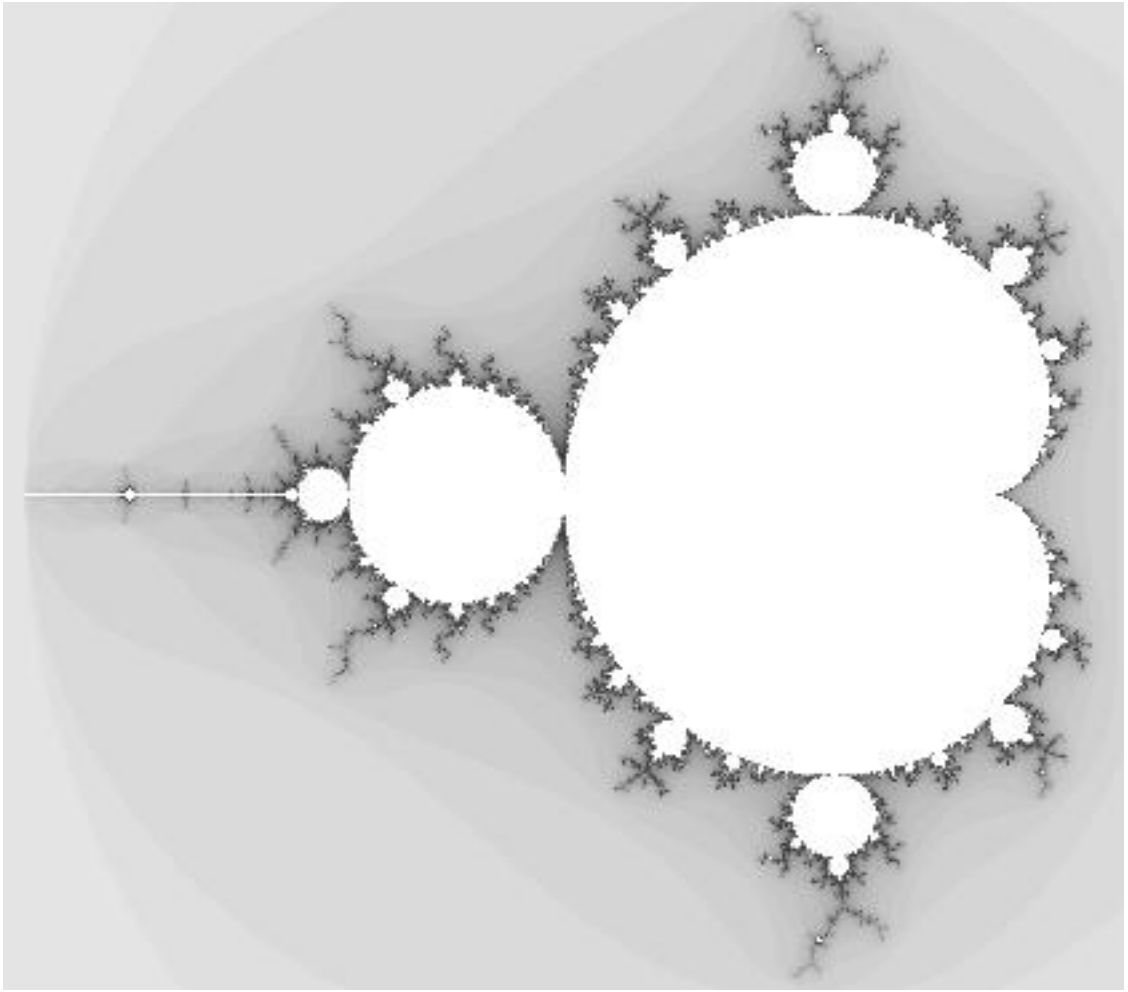
Ensembles de Julia

- Les ensembles de Julia sont basés sur le même principe. Cette fois-ci, on ne part pas de l'origine $(0,0)$ mais du point que l'on teste comme faisant ou non partie de l'emsemble. $((cte_x, cte_y)$ ou $z_0 = cte)$. A chaque transformation, ce n'est plus le point testé qui est ajouté, mais une constante arbitraire j ou (j_x, j_y).

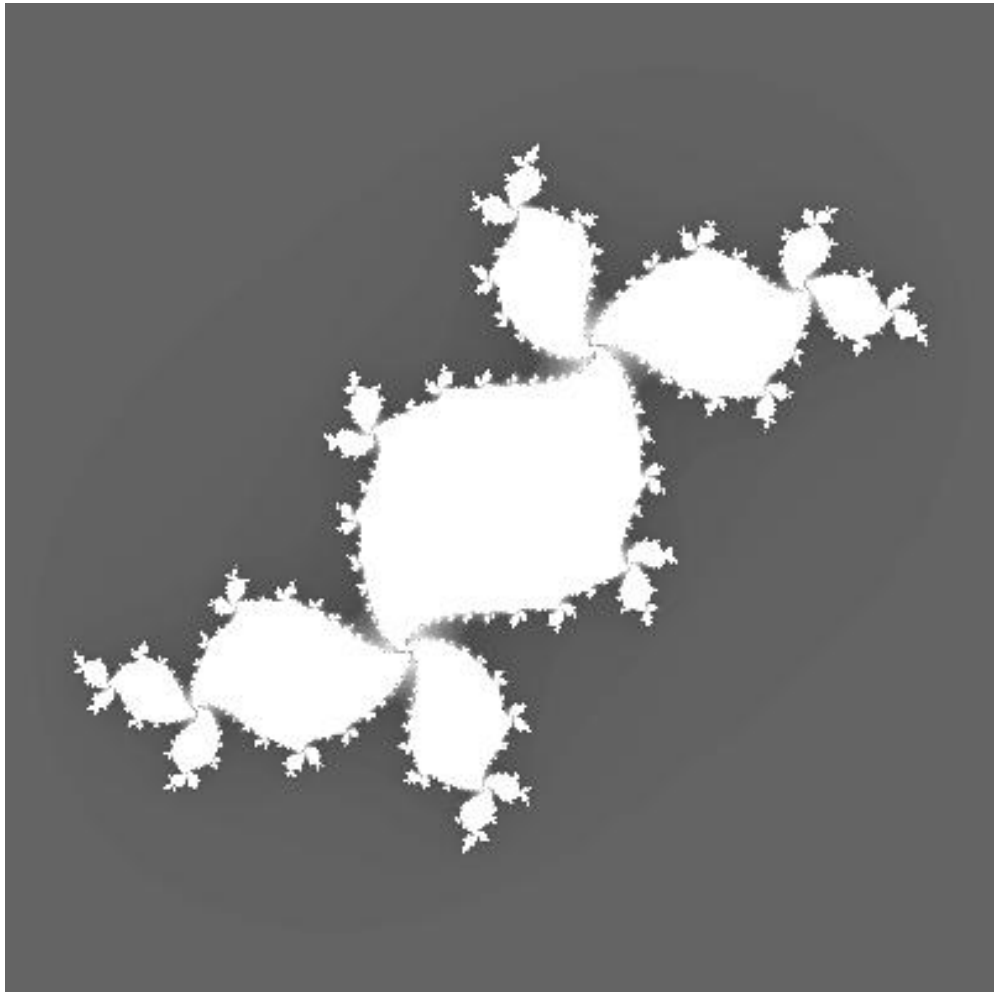
- Pour chaque couple (j_x, j_y) ou chaque constante j, on a un ensemble de Julia différent.

- En résumé :
  Pour un constante j complexe, l'ensemble de Julia associé est l'ensemble des cte complexes telles que la suite complexe $z_0 = cte, z_{n+1} = z_n^2 + j$ ne diverge pas.

Exemples



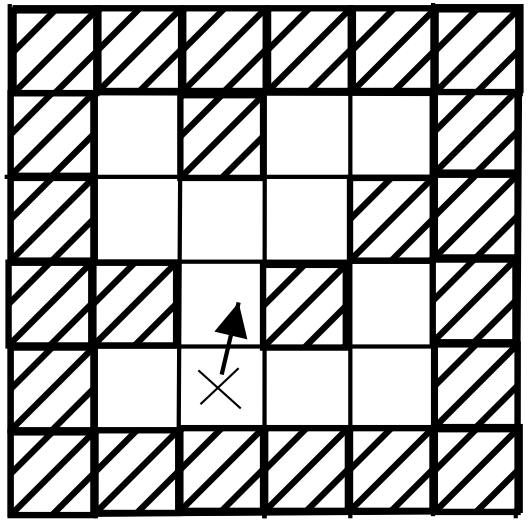
Mandelbrot



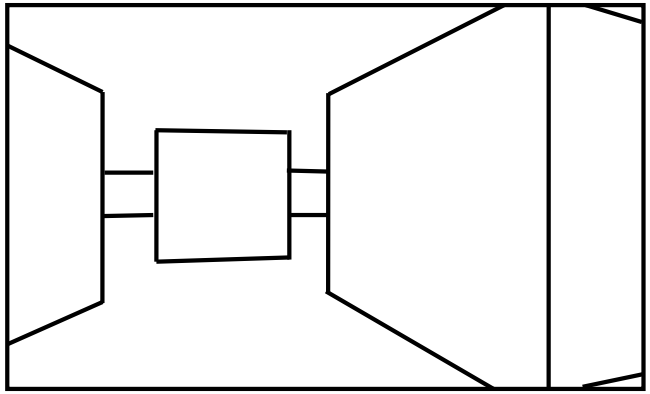
Julia

RayCasting - Principe

- Il s'agit de simuler en "temps réel" un environnement de type labyrinthe, avec une vue réaliste (telle qu'on pourrait la voir).



Vue de dessus



Vue "realiste"

- Lorsqu'on se déplace dans le labyrinthe, la vue "réaliste" évolue.

- La méthode expliquée n'est pas l'unique méthode possible mais c'est celle utilisée pour le jeu original.

Premières réflexions

- La vue est symétrique haut/bas. On a pour chaque moitié d'image un demi bout de mur et un bout de sol ou plafond.

- A x constant, plus le mur est loin, plus la demi-ligne "mur" est petite (et donc la ligne sol ou plafond est plus grande).

- D'où le principe de calculer le distance entre nous et le mur pour chaque x de notre image. On trace alors une ligne verticale pour le plafond suivie d'une ligne verticale pour le mur et enfin une pour le sol.

Préliminaires

- Il va nous falloir conserver plusieurs données:
  - la description du labyrinthe : un tableau de `int` remplis de 0 et 1
  - notre position dans le labyrinthe : $(x0, y0)$
  - notre angle de vision : a

```
int  laby[]=
{
```

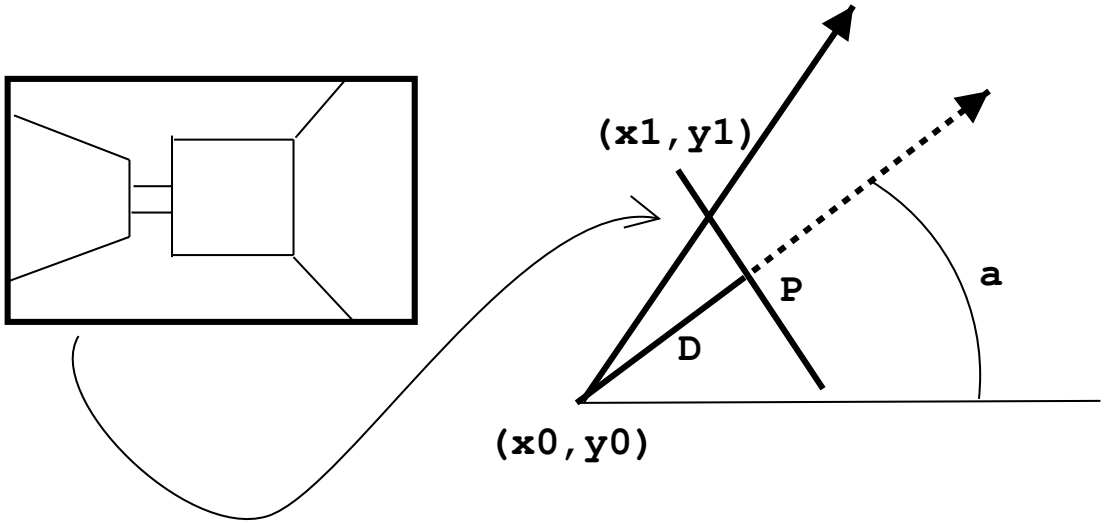
1	1	1	1	1	1
1	0	1	0	0	1
1	0	0	0	1	1
1	1	0	1	0	1
1	0	0	0	0	1
1	1	1	1	1	1


```
};
```

Ensemble de droites

- Pour chaque x de notre image, on crée une droite passant par notre position, et légerement decalée par rapport à la direction de vision. On obtient tout un ensemble de droites.

- On utilise un plan de projection simulant l'écran vu de dessus, représenté par un segment de droite. Chaque x de notre image sera un point différent sur ce segment de droite. Les deux points $(x0, y0)$ et $(x1, y1)$ permettent de déterminer une unique droite définie par un point $(x0, y0)$ et un vecteur directeur (Vx, Vy).



Calcul de $(x1, y1)$

- Pour la taille du segment P, la taille d'une case convient (on prendra 1.0 comme taille de case). Pour D, la distance entre notre position et le segment de projection P, une moitié de case convient.

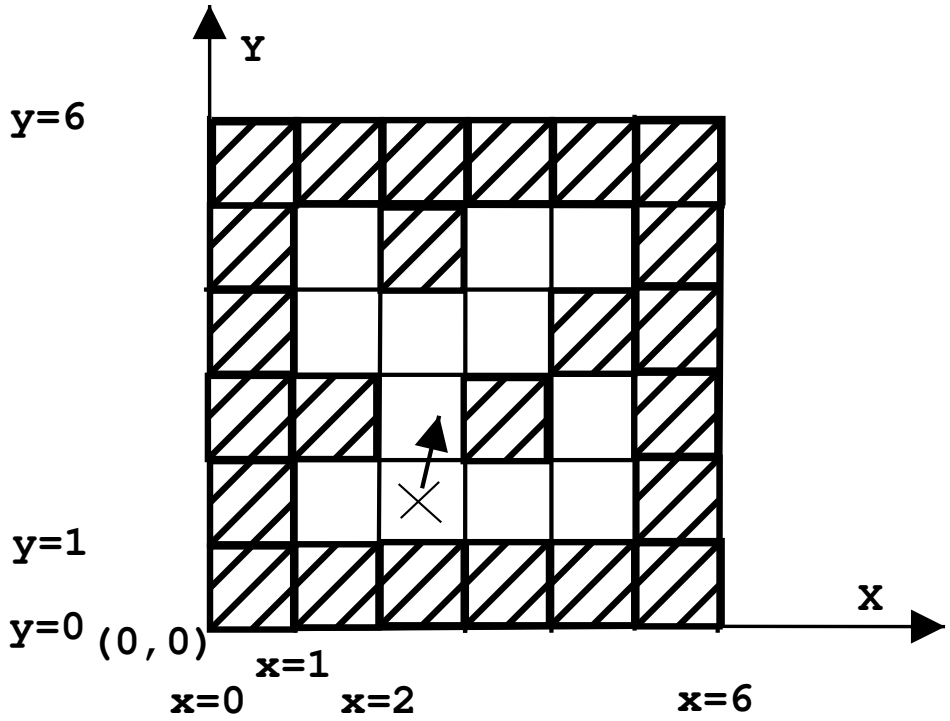
- Partons du cas où l'angle de vision a est nul et que notre position est $(0, 0)$. Les coordonnées$(x1, y1)$ sont alors égales à $(D, \frac{P(\frac{win\_x}{2} - x)}{win\_x})$

- On transforme ensuite ces coordonées grâce aux formules de rotation suivant l'angle a :
$X' = Xcos(a) - Ysin(a)$ et $Y' = Xsin(a) + Ycos(a)$

- Enfin on effectue une translation jusqu'à notre position réelle dans le labyrinthe :
$X'' = X' + x0$ et $Y'' = Y' + y0$

Les droites du labyrinthe

- Chaque séparation de case est une droite à x ou à y constant.



Intersections

- Pour chaque x, on va chercher les intersections entre la droite $(x0, y0)/(Vx, Vy)$ et toutes les droites du labyrinthe.

- En paramétriques :
$\begin{cases} x = x0 + kVx \\ y = y0 + kVy \end{cases}$ et $x = cte_x$ ou $y = cte_y$

- k représente la distance entre nous et le changement de case (proportionellement à la norme du vecteur (Vx, Vy)).

- Il faut alors déterminer si la case dans laquelle on vient de rentrer est un mur ou non. En remettant k dans l'équation paramétrique de la droite, on obtient par exemple $(5.0, 3.4)$ pour notre point d'intersection. Il faut alors aller voir la case [5] [3] de notre tableau.

A l'écran

- Enfin, de toutes ces intersections, il faut garder la plus petite non négative, qui croise un mur. On a donc la distance avec le mur le plus proche qui doit être affiché.

- $\frac{win\_y}{2k}$ convient comme demi-hauteur du mur en nombre de pixel.

Optimisations

- Utilisation des `int` à la place des `float` avec la méthode de la virgule fixe (on manipule nos valeurs multipliées par 1000).

- Pré-calcul de 360 sinus et cosinus, et utilisation d'un entier pour stocker l'angle de vision.

- Réaliser une image avec seulement le sol et le plafond. A chaque nouvelle image, on copie celle du sol+plafond. Seuls les murs sont à tracer.

- Ne pas faire 2 fois le même calcul.