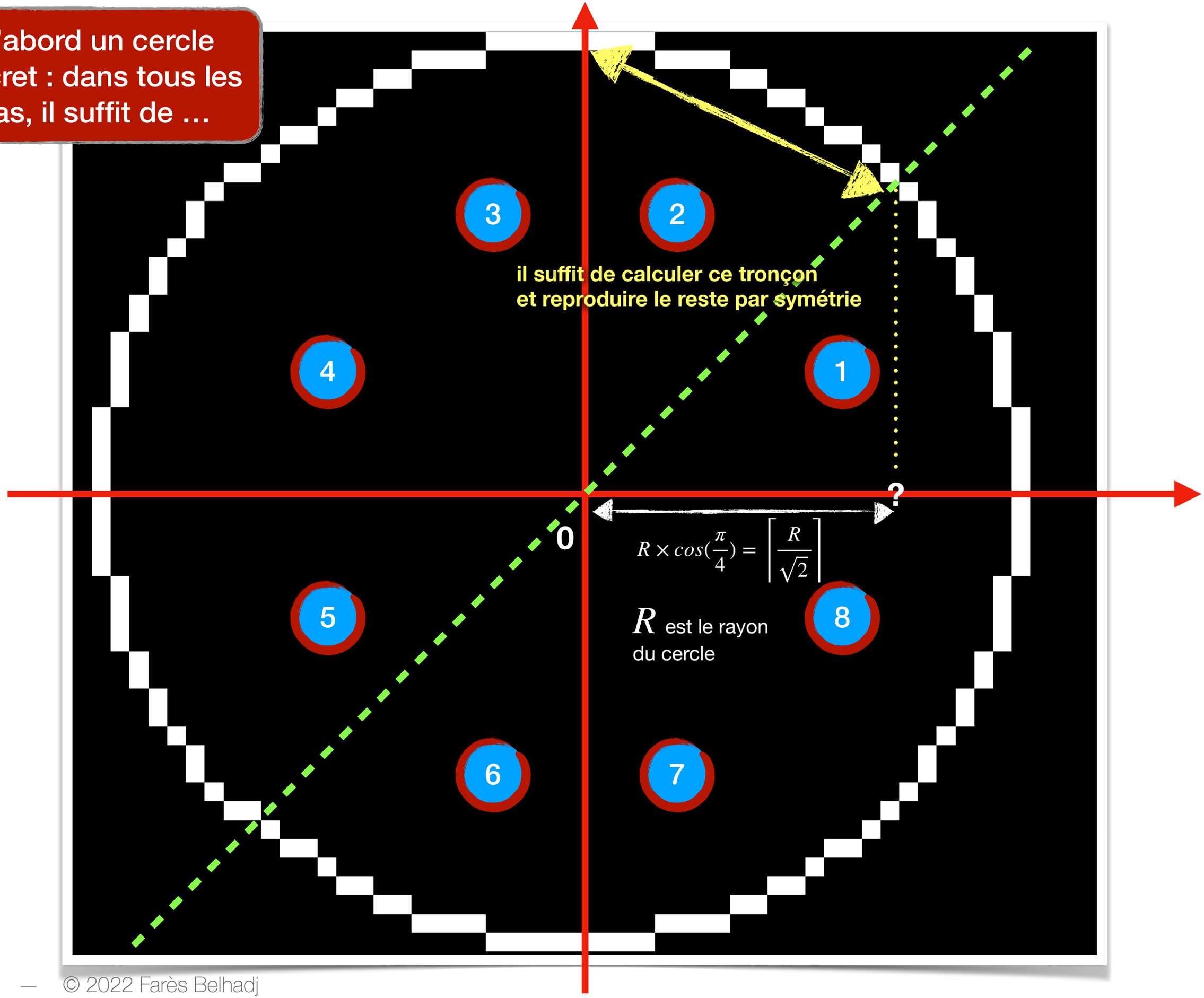
Cercles grossissants

D’abord un cercle discret : dans tous les cas, il suffit de …



D'abord un cercle discret : dans tous les cas, il suffit de …

- Choisir de calculer les coordonnées de l'arc de cercle au 2nd octan ;
- L'abscisse varie de 0 à $\left\lceil \frac{R}{\sqrt{2}} \right\rceil$
- L'ordonnée est calculée à l'aide de l'équation simplifiée du cercle :

  $(C) : (X - x_0)^2 + (Y - y_0)^2 = R^2$

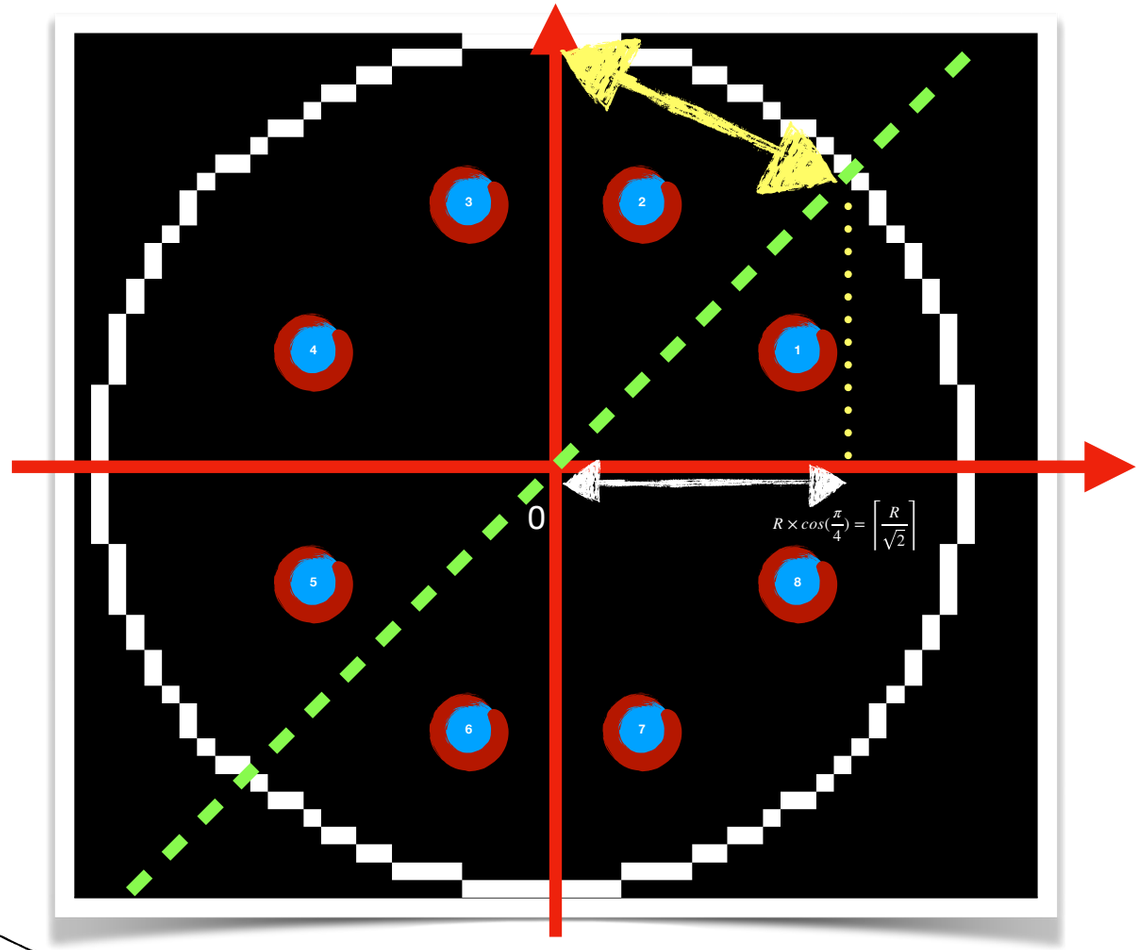
  où (x_0, y_0) est la coordonnée de l'origine du cercle

  et R est le rayon du cercle

  On pose : $\begin{cases} x = X - x_0 \\ y = Y - y_0 \end{cases}$ (on fera **+**(x_0, y_0) au moment du **`plot`**)

  ainsi $x^2 + y^2 = R^2 \Leftrightarrow y^2 = R^2 - x^2$

  donc $y = \sqrt{R^2 - x^2}$ (pas de ± car on est dans le 2nd octan)



Autre possibilité : Bresenham'77
(voir aussi le source de la fonction **`gl4dpCircle`**)

- Pour chaque (x, y) obtenu, colorier (**`plot`**) les 8 coordonnées sur chacun de 8 octans :

2 $(x + x_0, y + y_0)$ ↘symétrie diagonale↘ 1 $(y + x_0, x + y_0)$

← symétrie horizontale ←

3 $(-x + x_0, y + y_0)$ 4 $(-y + x_0, x + y_0)$

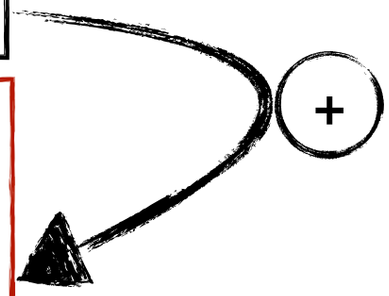
↓ symétrie verticale ↓

7 $(x + x_0, -y + y_0)$ 8 $(y + x_0, -x + y_0)$

6 $(-x + x_0, -y + y_0)$ 5 $(-y + x_0, -x + y_0)$

Code du cercle discret

sample code `sc_00_00_blank-1.0` (github de GL4D)



```c
void circle(int x0, int y0, int r, GLuint color) {
  int maxx = ceil(r / sqrt(2.0f)) + 1, r2 = r * r;
  GLuint * p = gl4dpGetPixels();
  int w = gl4dpGetWidth();
  for(int x = 0, y; x < maxx; ++x) {
    y = ceil(sqrt( r2 - x * x));
    /* attention, pas de test "le point est dans l'ecran ?" */
    p[ ( y + y0) * w + ( x + x0) ] = color; /* octan 2 */
    p[ ( x + y0) * w + ( y + x0) ] = color; /* octan 1 */
    p[ ( y + y0) * w + (-x + x0) ] = color; /* octan 3 */
    p[ ( x + y0) * w + (-y + x0) ] = color; /* octan 4 */
    p[ (-y + y0) * w + ( x + x0) ] = color; /* octan 7 */
    p[ (-x + y0) * w + ( y + x0) ] = color; /* octan 8 */
    p[ (-y + y0) * w + (-x + x0) ] = color; /* octan 6 */
    p[ (-x + y0) * w + (-y + x0) ] = color; /* octan 5 */
  }
  /* dire qu'on a modifié le screen */
  gl4dpScreenHasChanged();
}
```

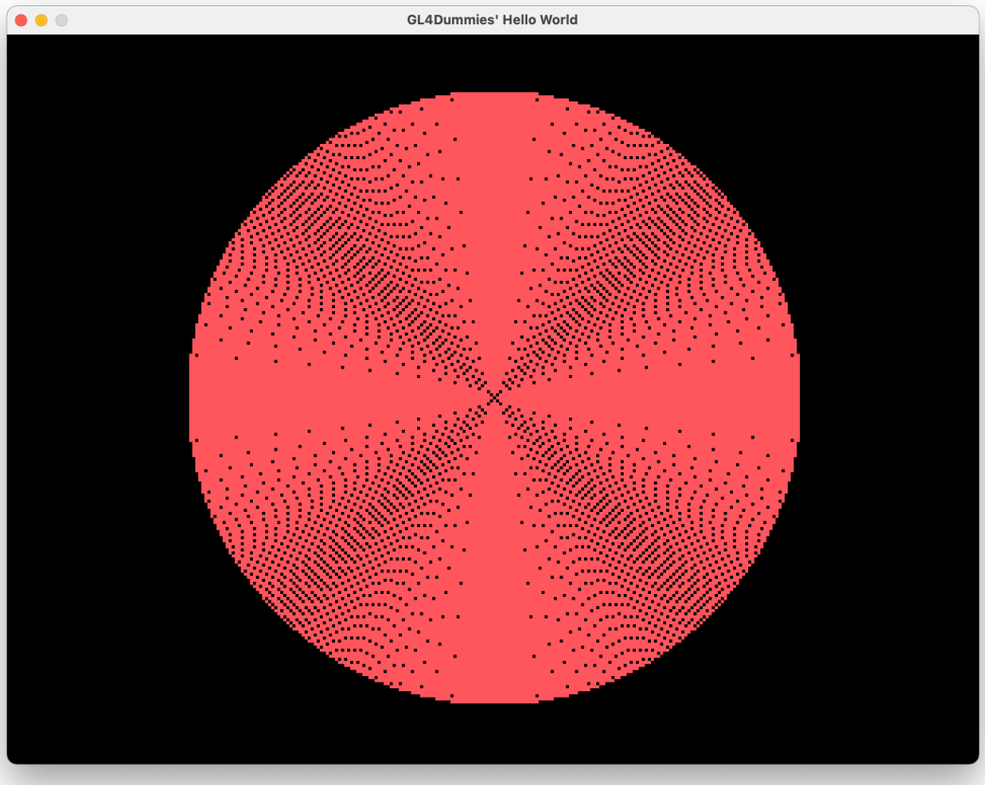
Ne pas oublier de mettre en place la callback display avant la main loop.
Voici un exemple de fonction display :

```c
void dessin(void) {
  /* effacement du screen en cours en utilisant la couleur par défaut,
   * le noir */
  gl4dpClearScreen();
  /* cercle centré sur le screen et de couleur rose */
  circle(160, 120, 100, RGB(255, 100, 100));
  /* fonction permettant de rafraîchir l'ensemble de la fenêtre*/
  gl4dpUpdateScreen(NULL);
}
```

Faire grossir le cercle (V1)

Faire une boucle sur le rayon

```c
void dessin(void) {
  /* effacement du screen en cours en utilisant la couleur par défaut,
   * le noir */
  gl4dpClearScreen();
  /* cercle centré sur le screen et de couleur rose */
  for(int r = 0; r < 101; ++r)
    circle(160, 120, r, RGB(255, 100, 100));
  /* fonction permettant de raffraîchir l'ensemble de la fenêtre*/
  gl4dpUpdateScreen(NULL);
}
```



Problème 1 (??) : on ne voit pas grossir le cercle
Problème 2 : c’est quoi ces points non colorés ?

Faire grossir le cercle (V2)

Pour le 1er problème, pour voir grossir le cercle,
il est nécessaire de laisser l'écran se rafraîchir après
chaque cercle « plus gros de 1 »
La boucle est donc pas la bonne solution
(on est déjà dans une boucle, ***event-simu-draw***)

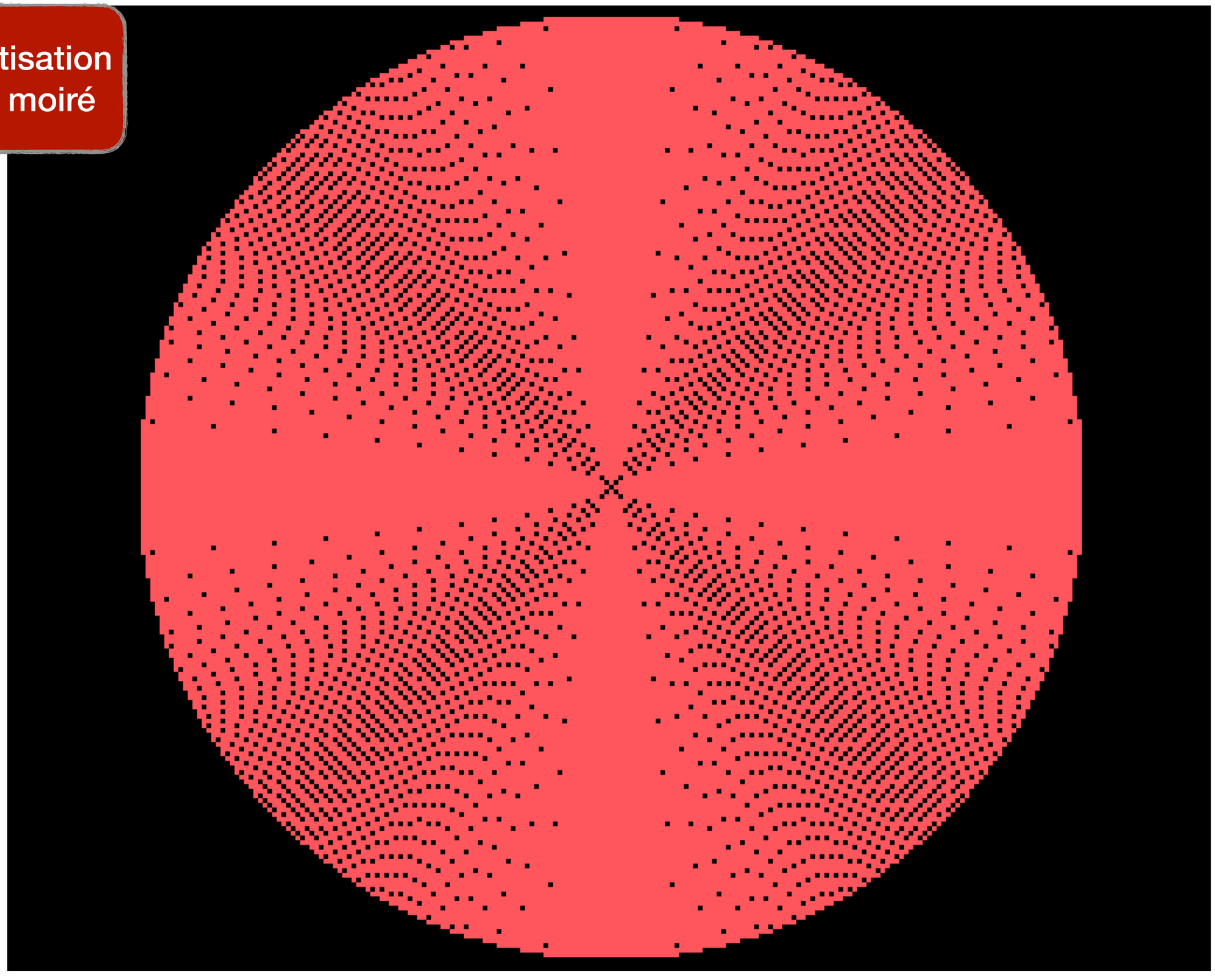
⇒ on *clear* le *screen* avant dessin,
on dessine avec un rayon qui commence à 1
et on l'augmente de 1
IL FAUT QU'IL SE RAPPELLE DE SA PRÉCÉDENTE VALEUR

⇒ SOLUTION : variable locale avec *specifier* **`static`**
(plus élégant qu'une variable globale)

```c
void dessin(void) {
  static int r = 1;
  /* cercle centré sur le screen et de couleur rose */
  circle(160, 120, r, RGB(255, 100, 100));
  if(r < 100)
    ++r;
  /* fonction permettant de raffraîchir l'ensemble de la fenêtre*/
  gl4dpUpdateScreen(NULL);
  /* juste histoire de nous laisser le temps de le voir grossir */
  SDL_Delay(100);
}
```

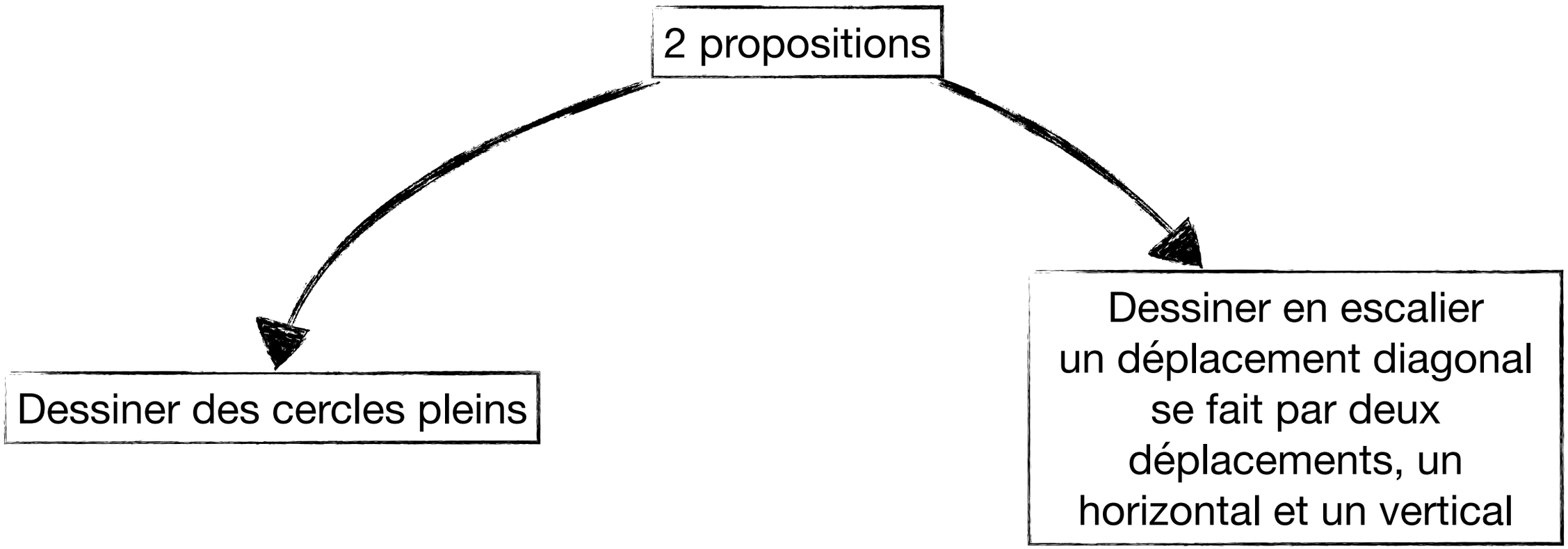
~~Problème 1 (??) : on ne voit pas grossir le cercle~~
Problème 2 : c'est quoi ces points non colorés ?

La discrétisation et l'effet moiré



Problème 2 : faire une recherche **images** sur « effet moiré »
ou aller sur le wiki du Moiré physique

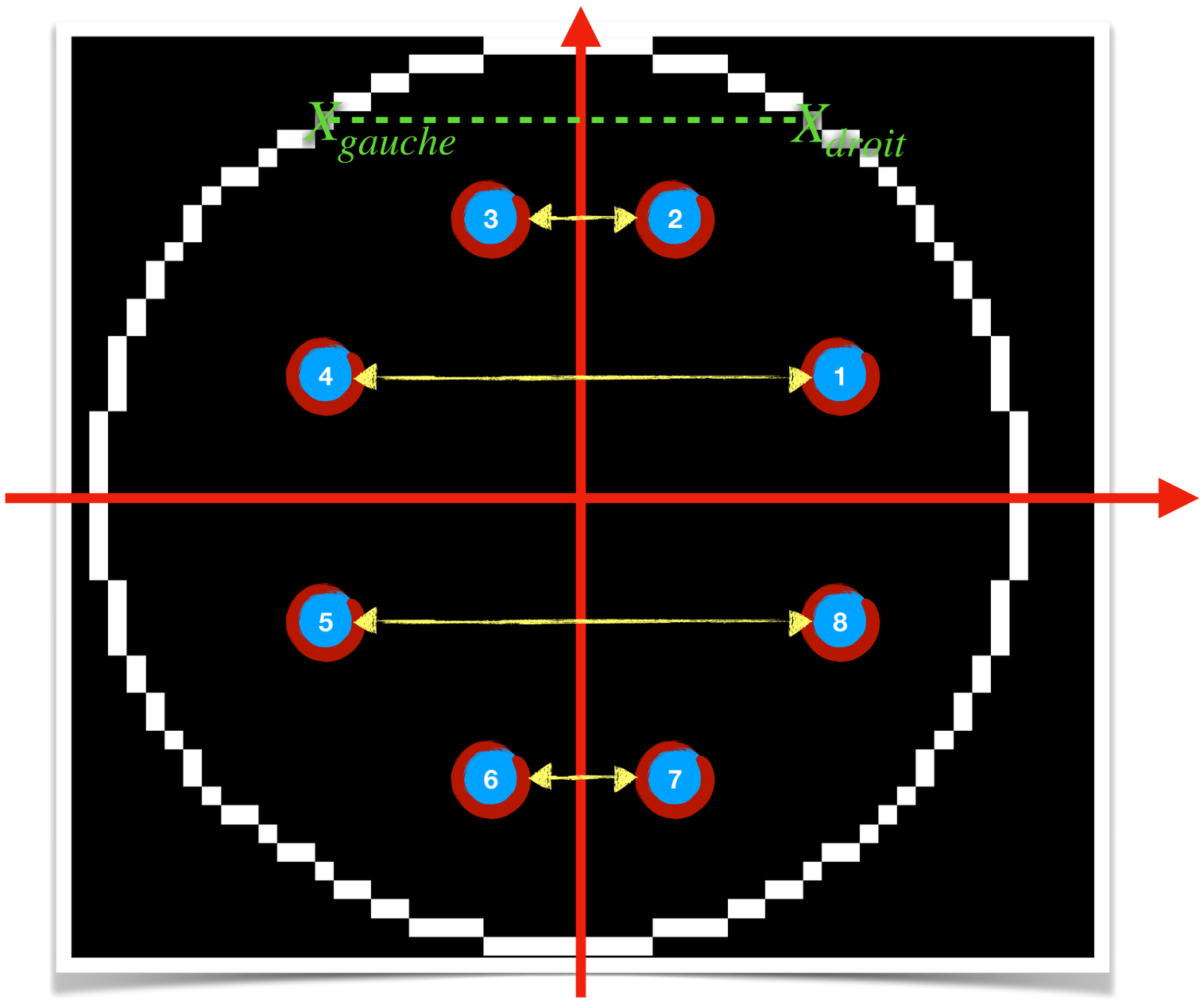
Éviter l’effet moiré



Éviter l'effet moiré 1/2

Dessiner des cercles pleins **(sans optimisation)**

Relier deux à deux certains points parmi les 8 coordonnées obtenues pour les 8 octans



Éviter l'effet moiré 1/2

Dessiner des cercles pleins

Avoir une fonction pour dessiner des lignes horizontales

Code de départ :
https://expreg.org/amsi/C/APG2223S1/code/01_algos/tests_circle-1.0.tgz

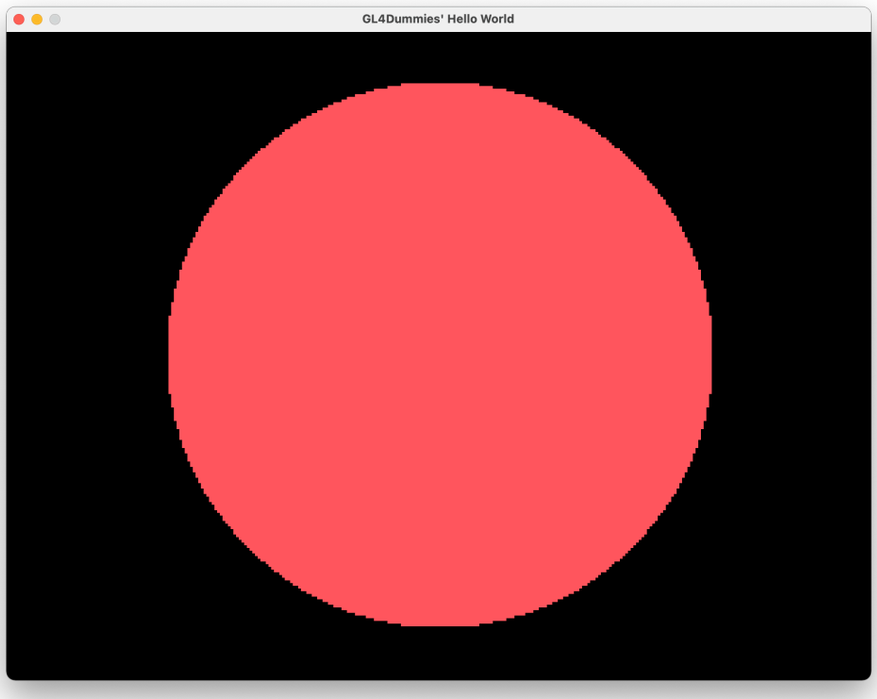
```c
/* droite horizontale */
static inline void hline(int x0, int x1, int y, GLuint color) {
  int w = gl4dpGetWidth(), h = gl4dpGetHeight();
  if(y < 0 || y >= h) /* pas besoin de dessiner */
    return;
  /* x le plus à gauche, x le plus à droite */
  int xg, xd;
  if(x0 < x1) {
    xg = x0 < 0 ? 0 : x0;
    xd = x1 >= w ? w - 1 : x1;
  } else {
    xg = x1 < 0 ? 0 : x1;
    xd = x0 >= w ? w - 1 : x0;
  }
  if(xg >= w || xd < 0) /* pas besoin de dessiner */
    return;
  GLuint * p = gl4dpGetPixels();
  for(int x = xg, yw = y * w; x <= xd; ++x)
    p[yw + x] = color;
}
```

Éviter l'effet moiré 1/2

Dessiner des cercles pleins

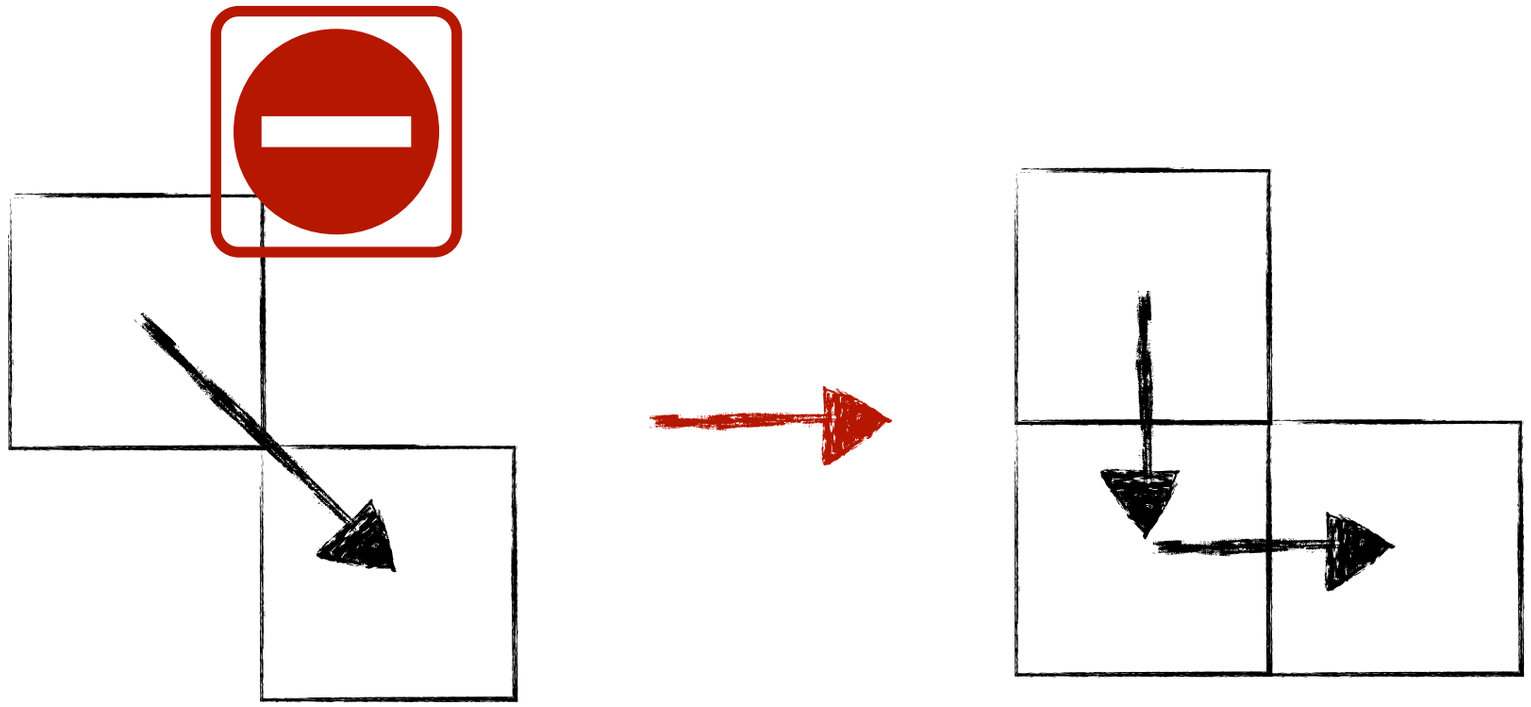
Relier deux à deux, et par une ligne horizontale certains des points du cercle

```c
/* filled circle */
void fcircle(int x0, int y0, int r, GLuint color) {
  int maxx = ceil(r / sqrt(2.0f)) + 1, r2 = r * r;
  for(int x = 0, y; x < maxx; ++x) {
    y = ceil(sqrt( r2 - x * x));
    hline((-x + x0), ( x + x0), ( y + y0), color);
    hline((-y + x0), ( y + x0), ( x + y0), color);
    hline((-x + x0), ( x + x0), (-y + y0), color);
    hline((-y + x0), ( y + x0), (-x + y0), color);
  }
  /* dire qu'on a modifié le screen */
  gl4dpScreenHasChanged();
}
```



Éviter l’effet moiré 2/2

Dessiner en escalier chaque déplacement diagonal



Il faut donc savoir **quand** il y un déplacement diagonal

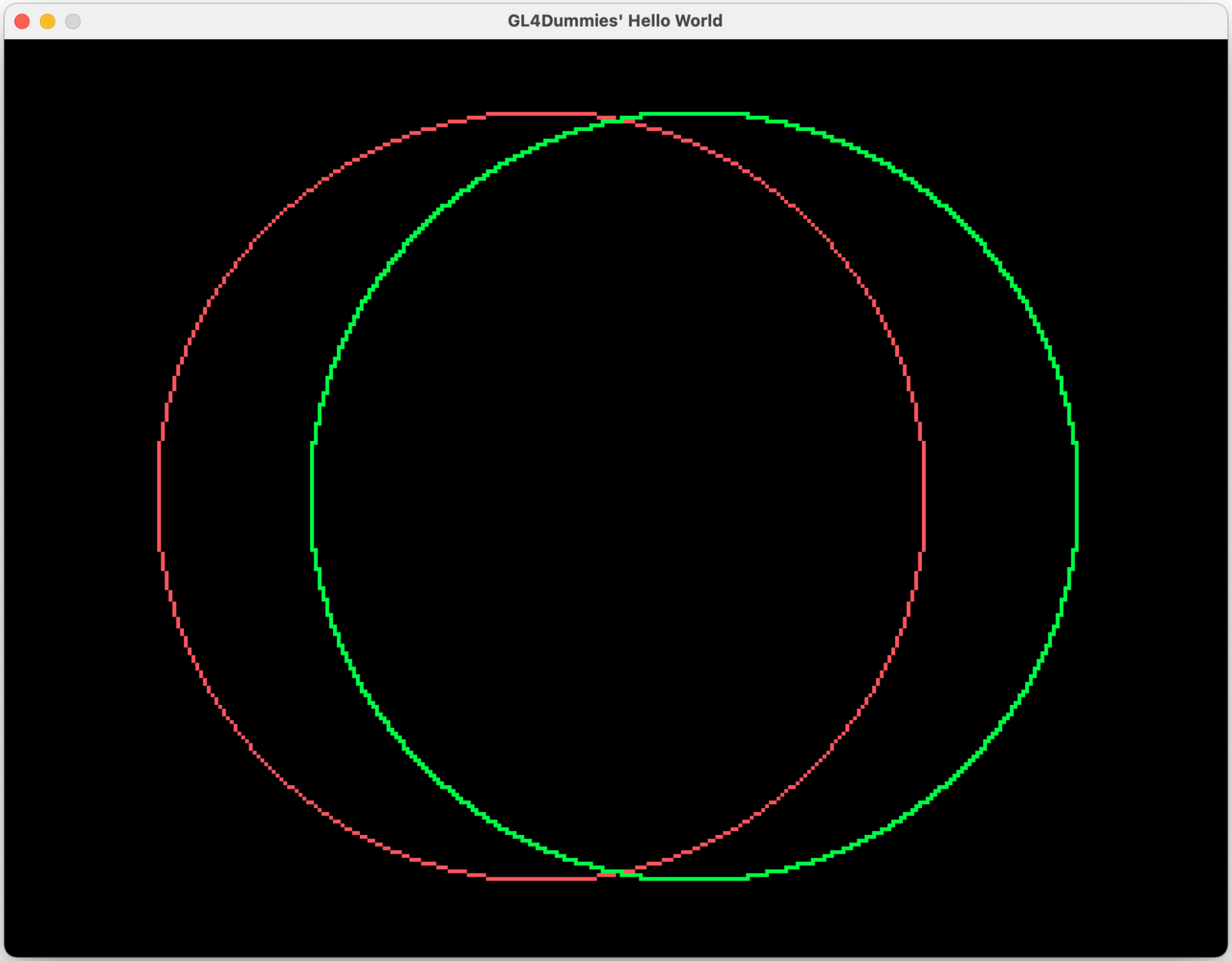
Éviter l'effet moiré 2/2

Dessiner en escalier chaque déplacement diagonal

```
void circleNOM(int x0, int y0, int r, GLuint color) {
  int maxx = ceil(r / sqrt(2.0f)) + 1, r2 = r * r;
  GLuint * p = gl4dpGetPixels();
  int w = gl4dpGetWidth();
  /* cy : current y ; py : previous y */
  for(int x = 0, cy, py = r; x < maxx; ++x) {
    cy = ceil(sqrt( r2 - x * x));
    /* attention, pas de test "le point est dans l'ecran ?" */
    p[ ( cy + y0) * w + ( x  + x0) ] = color; /* octan 2 */
    p[ ( x  + y0) * w + ( cy + x0) ] = color; /* octan 1 */
    p[ ( cy + y0) * w + (-x  + x0) ] = color; /* octan 3 */
    p[ ( x  + y0) * w + (-cy + x0) ] = color; /* octan 4 */
    p[ (-cy + y0) * w + ( x  + x0) ] = color; /* octan 7 */
    p[ (-x  + y0) * w + ( cy + x0) ] = color; /* octan 8 */
    p[ (-cy + y0) * w + (-x  + x0) ] = color; /* octan 6 */
    p[ (-x  + y0) * w + (-cy + x0) ] = color; /* octan 5 */
    if(py > cy) {
      /* previous x est utilisé avec current y */
      int px = x - 1;
      p[ ( cy + y0) * w + ( px + x0) ] = color; /* octan 2 */
      p[ ( px + y0) * w + ( cy + x0) ] = color; /* octan 1 */
      p[ ( cy + y0) * w + (-px + x0) ] = color; /* octan 3 */
      p[ ( px + y0) * w + (-cy + x0) ] = color; /* octan 4 */
      p[ (-cy + y0) * w + ( px + x0) ] = color; /* octan 7 */
      p[ (-px + y0) * w + ( cy + x0) ] = color; /* octan 8 */
      p[ (-cy + y0) * w + (-px + x0) ] = color; /* octan 6 */
      p[ (-px + y0) * w + (-cy + x0) ] = color; /* octan 5 */
    }
  }
  /* dire qu'on a modifié le screen */
  gl4dpScreenHasChanged();
}
```

Éviter l’effet moiré 2/2

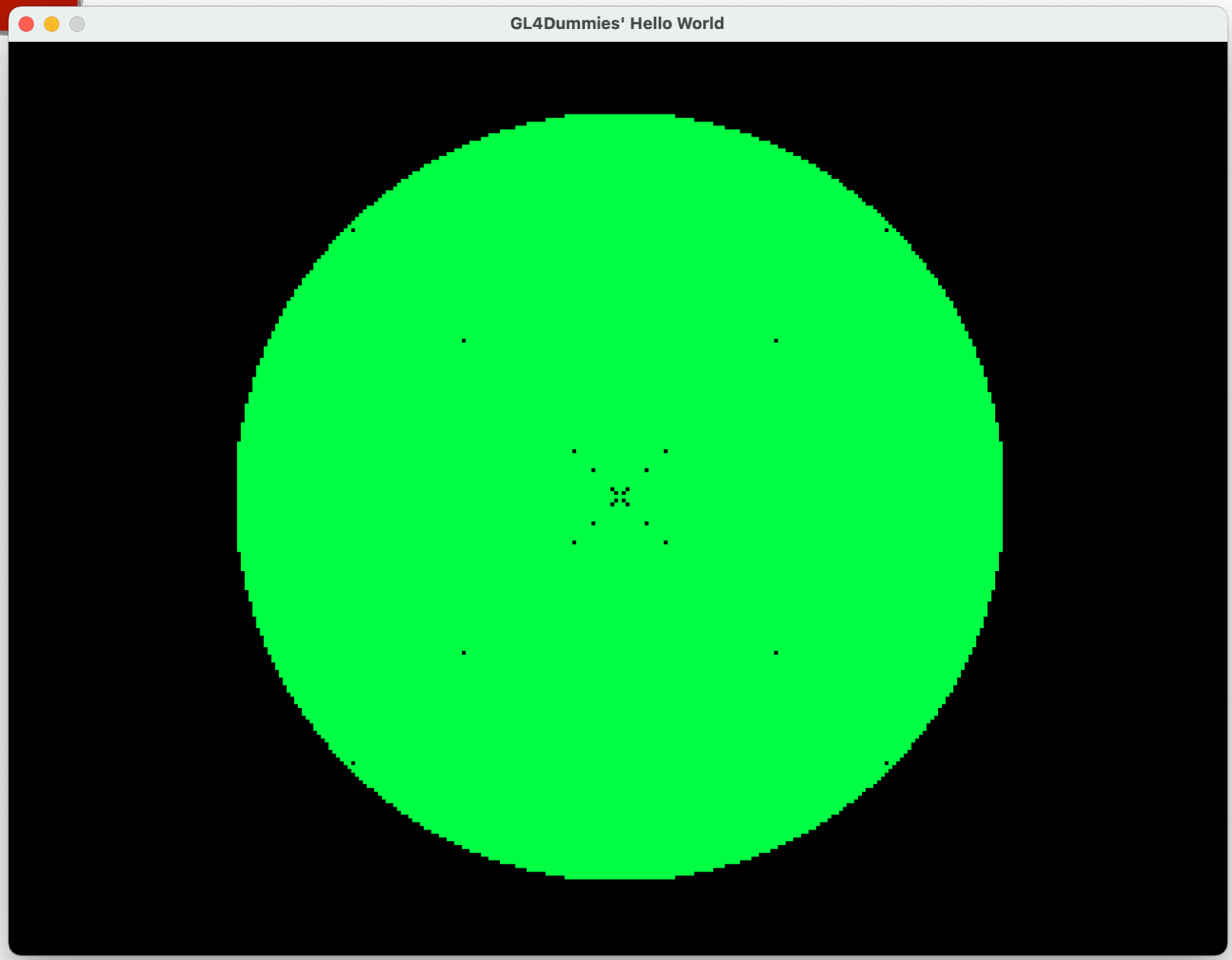
Dessiner en escalier chaque déplacement diagonal



Les deux (déplacement diagonal/escalier) côte à côte

Éviter l’effet moiré 2/2

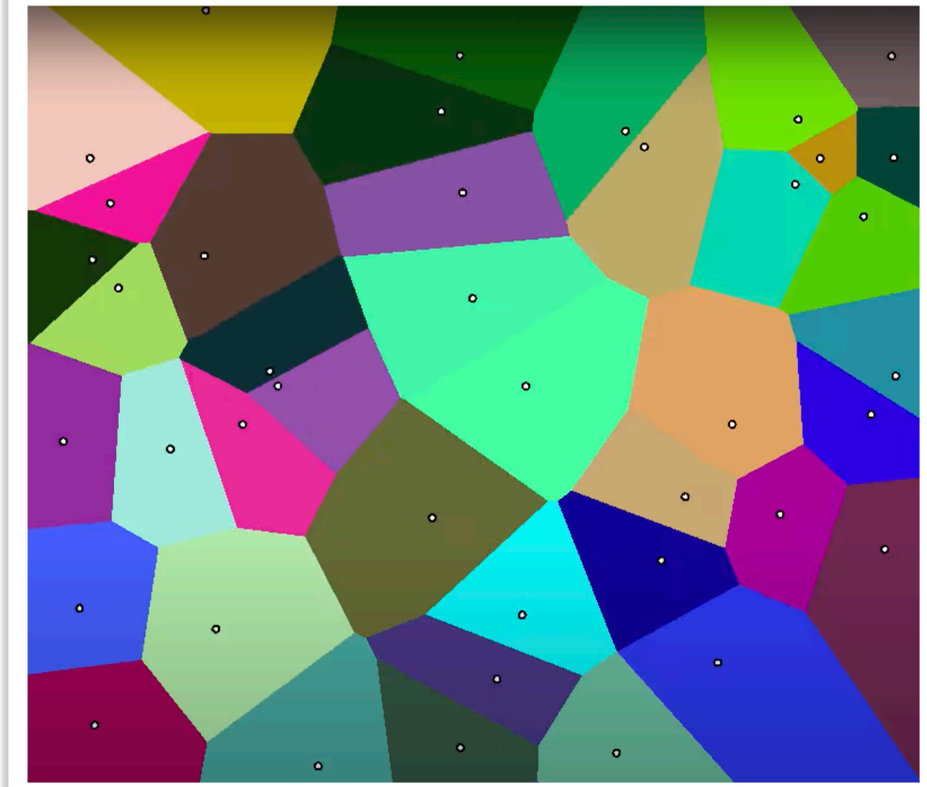
Dessiner en escalier chaque déplacement diagonal



Résultat en dessinant des cercles grossissants (déplacement en escalier)
(mieux mais pas parfait)

Diagramme de Voronoï

Diagramme de Voronoï



Déf. (FB) : c'est un pavage, ou un découpage, ou un partitionnement de l'espace en plusieurs zones déterminées par des points au centre de ses zones, ces points sont appelés « sites ».

Autre (FB) : le dual d'un diagramme de Voronoï est la triangulation de Delaunay.

Plus loin : Wikipédia

Voronoï par cercles grossissants

D’abord une vidéo

https://www.youtube.com/watch?v=oWzJvsiSkYE

Si on a pas fini, préparez, d’ici à la semaine prochaine, un code qui réalise la même chose, et qui utilise les éléments donnés aujourd’hui (`fcircle` ou `circleNOM`)

Voronoï par cercles grossissants — Proposition par Farès Belhadj

- **Diagramme de Voronoï est caractérisé par ses sites (NB_SITES)**
  - Un site est caractérisé par :
    - une coordonnée `(x, y)`, (choisie aléatoirement ?)
    - une couleur `color` (choisie aléatoirement ? tout sauf celle indiquant `libre`)
    - un état `state` prenant deux valeurs possibles `growing` (initial) / `finished` (fin)
  - Un rayon commun `radius` représentant la taille des cercles (à tenter de dessiner) en cours, débute à 1
  - Une couleur `libre` (par exemple `0x0`) représentant l'état libre ou occupé de chaque pixel
- **Comment réaliser le Voronoï par cercles grossissants ?**
  1. Initialiser le diagramme (cf. au dessus)
  2. Effacer SCREEN à l'aide de `libre`
  3. `Faire :`
     1. Pour chaque site dans ce diagramme (boucle `for`)
        1. Si le site est à l'état `finished`, le site passe son tour (passer au site suivant)
        2. Essayer de dessiner à partir de la coordonnée du site un cercle de rayon `radius` : par "essayer" on entend colorier si possible les pixels du cercle à l'aide de `color` ; ça ne sera possible que si le pixel est dans SCREEN et possède une couleur `libre` => Si cette tentative est réussie, `state` du site reste à `growing`. Sinon `state` du site passe à `finished`.
     2. Rafraîchir la fenêtre => UPDATE SCREEN
     3. Incrémenter `radius` de 1
  4. `Tant qu`'il y a au moins un site à l'état `growing`