TP Voronoï sur image

Préambule : Lors de la précédente séance*, il était attendu que vous complétiez le code fourni en cours (ou depuis le support) pour avoir l'ensemble des fonctions « cercles », puis que vous implémentiez l'algorithme du Voronoï par cercles grossissants fourni en fin de support. Le résultat attendu devait être similaire à : https://www.youtube.com/watch?v=oWzJvsiSkYE

Le TP d'aujourd'hui exploitera ce travail considéré réalisé afin de vous faire réaliser un diagramme de Voronoï qui se sert des couleurs piochées dans une image comme couleurs des cellules et qui anime l'ensemble.

Le travail attendu est à déposer sur le Moodle du cours, le temps alloué à ce travail est d'**une heure et 15 minutes.**

*Le support de cours utilisé lors de la précédente séance est disponible à l'adresse : https://expreg.org/amsi/C/APG2122S1/supports/03_algo_voronoi_cercles_grossissants.pdf

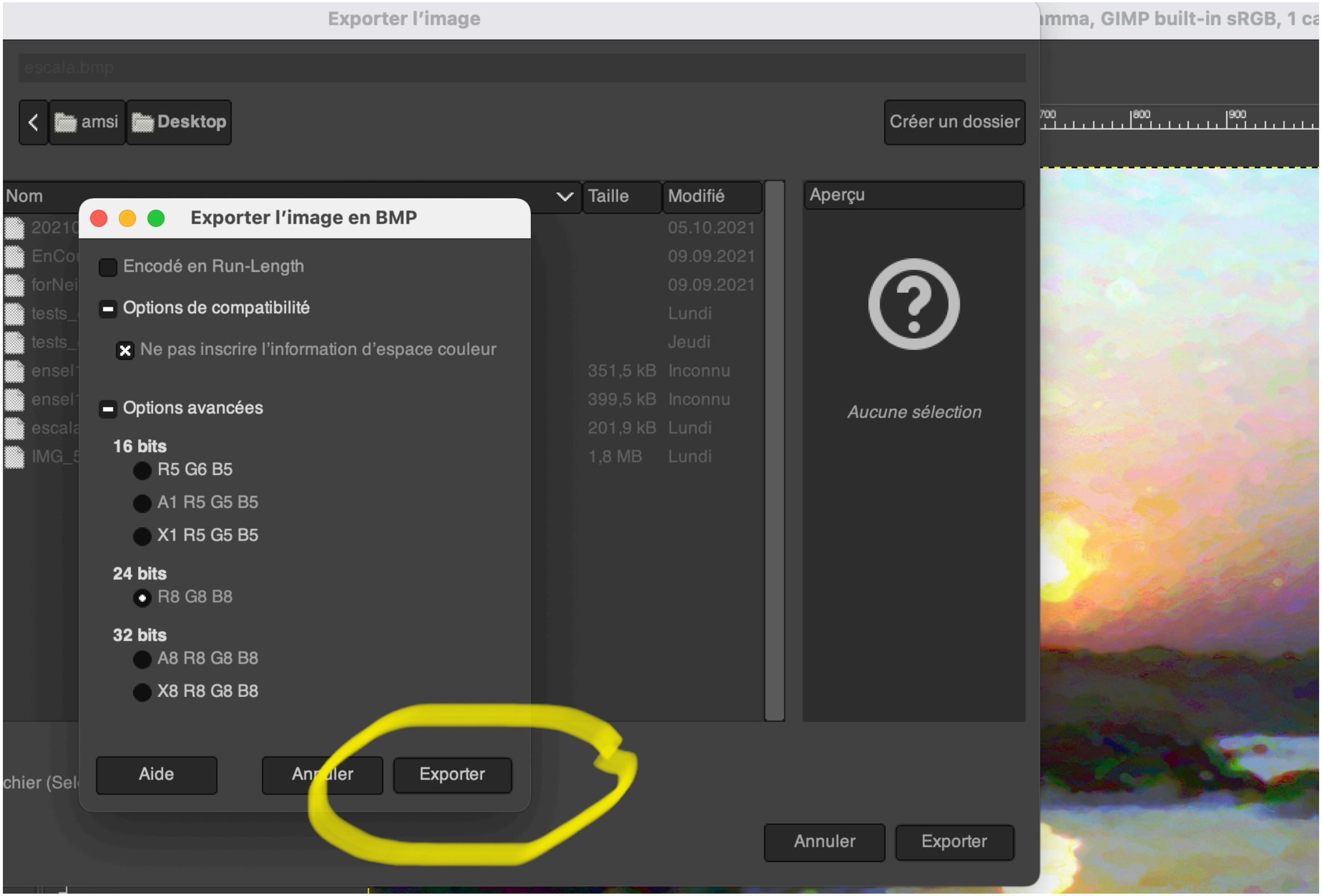
(1) Préparer une image et la charger

* Choisir une image (par exemple une photographie) de résolution moyenne (par exemple 1024×768, redimensionner si nécessaire) puis la convertir au format BMP (par exemple avec GIMP — voir exemple d'export si après).
* Copier l'image dans le répertoire contenant le source du projet, par exemple sous le nom `monImage.bmp` (si vous utilisez le `Makefile` vous pouvez ajouter le nom de ce fichier à la variable `EXTRAFILES` afin de l'inclure automatiquement à l'archive générée par l'exécution make `dist`).
* Modifier le code source afin d'avoir une *Surface SDL* qui pointe vers cette image. Nous utiliserons une globale avec *qualifieur* `static` déclarée juste avant le `main` et chargerons l'image juste avant l'ouverture de la fenêtre. Les dimensions de la fenêtre seront calquées sur celles de l'image (surface).

```
/* ... le début de VOTRE code */

/* pour TP : déclarer la surface */
SDL_Surface * _image = NULL;

int main(int argc, char ** argv) {
  /* pour TP : charger l'image */
  SDL_Surface * src = SDL_LoadBMP("monImage.bmp");
  if(src == NULL) {
    fprintf(stderr, "Impossible de charger monImage.bmp\n");
    return 2;
  }
  /* convertir en 32 bits */
  _image = SDL_CreateRGBSurface(0, src->w, src->h, 32, R_MASK, G_MASK, B_MASK, A_MASK);
  SDL_BlitSurface(src, NULL, _image, NULL);
  /* libérer la source */
  SDL_FreeSurface(src);
  /* tentative de création d'une fenêtre pour GL4Dummies */
  if(!gl4duwCreateWindow(argc, argv, /* args du programme */
                         "GL4Dummies' Voronoi", /* titre */
                         10, 10, _image->w, _image->h, /* x,y, largeur, heuteur */
                         GL4DW_SHOWN) /* état visible */) { /* ... */
  /* ... la suite de VOTRE code */
```



Exemple de paramétrage sous GIMP avant export

(2) Libérer l'image chargée au moment du `atexit`

* **Si vous n'avez pas encore mis en place une fonction à appeler au `atexit`, le faire. Dans la suite, nous l'appellerons `quit`.**
* **Ajouter au code de votre fonction appelée au `atexit`, ces lignes permettant de libérer (la mémoire de) la surface `_image` :**

```c
void quit(void) {
  /* pour TP : libérer l'image */
  if(_image)
    SDL_FreeSurface(_image);
  _image = NULL;
}
```

* **Compiler et tester l'ensemble (vous pouvez ajouter un print dans votre fonction appelée au `atexit` afin de vérifier qu'elle est bien appelée**

(3) Piocher la couleur dans l'image

* Lors de la phase initiale permettant de créer le diagramme de Voronoï, faire en sorte de piocher une couleur dans la surface référençant l'image au lieu de récupérer aléatoirement une couleur.
* Remarque : l'axe des ordonnées est souvent représenté vers le bas dans les fichier images, il faut donc l'inverse avec une formule du type : $y \leftarrow H - y - 1$ où H est la hauteur de l'image.
* Ci-après un exemple des lignes de code réalisant ce qui est attendu (attention vous n'avez probablement pas un code identique à celui ci-après, il faut donc juste s'inspirer de ce qui suit) :

```
for(int i = 0; i < _nb_sites; ++i) {
  /* _sites est mon tableau dynamique contenant l'ensemble des cellules (struct) de mon diagramme */
  _sites[i].x = rand() % gl4dpGetWidth();
  _sites[i].y = rand() % gl4dpGetHeight();
  /* Pour TP : récupérer la couleur depuis _image */
  /* AVANT */
  /* GLubyte r = rand()&0xFF, g = rand()&0xFF, b = rand()&0xFF; */
  /* _sites[i].color = RGB(r, g, b); */
  _sites[i].color = ((GLuint *)_image->pixels)[(gl4dpGetHeight() - _sites[i].y - 1) * gl4dpGetWidth() + _sites[i].x];
  _sites[i].growing = 1;
}
```

* Compiler et tester l'ensemble

(4) Ne visualiser que l’étape finale du diagramme

* Dans la callback function liée au display, faire en sorte de ne plus voir la progression de la construction du diagramme de Voronoï par cercles grossissants, mais voir directement le diagramme fini avant d’en lancer un autre — à l’étape d’après.
* POUR MA PART, ayant les fonctions : « `static void voro_init(int);` » ET « `static int voro_grow(void);` », cela donne la transformation suivante :

```
/* AVANT */
void dessin(void) {
  if(!voro_grow())
    voro_init(2000);
  gl4dpUpdateScreen(NULL);
}
/* APRÈS */
void dessin(void) {
  voro_init(2000);
  while(voro_grow());
  gl4dpUpdateScreen(NULL);
}
```

* Compiler et tester l’ensemble

(5) Mouvements libres des cellules

* Dernier étape, modifier et ajouter à la structure (`struct`) représentant chaque site de cellule la capacité de se déplacer (par exemple une donnée (v_x, v_y) représentant une vitesse par *frame*) et initialiser cette nouvelle donnée.
* A chaque *frame*, au lieu de réinitialiser le diagramme (redonner des positions aléatoires) faire en sorte que les positions précédentes bougent avec la nouvelle donnée (*i.e.* la vitesse (v_x, v_y)) ; il faut garder le fait d'effacer l'écran et de remettre le rayon `radius` à zéro.
* Compiler, tester, archiver et envoyer l'ensemble du source.

* La vidéo donnée là https://expreg.org/amsi/C/APG2122S1/videos/APG_20211020_TP_Voronoi.mp4 retrace tout ce qu'il y a à faire pour ce TP ; il faut néanmoins adapter par rapport à votre code de départ. Bon courage.