Modalités du travail pratique

- L'étudiant dispose du temps d'autonomie supervisé en classe par l'enseignant (pendant lequel celui-ci se tient disponible pour reformuler une incompréhension, donner des pistes de résolution en cas de blocage, guider l'étudiant à la résolution d'éventuelles problématiques).

- S'il le souhaite, l'étudiant a la possibilité d'avancer individuellement et sans assistance sa production sur le temps non dédié au cours.

- Il est conseillé de maîtriser les exercices d'entraînement (à rendre) jusqu'à la section 11 (types structurés) pour réussir les exercices proposés.

- Chaque exercice doit être représenté par un code source `.c` rendu dans une archive `.zip` sous forme de fichiers sources en **Langage C** en extension `.c` (qui compilent). Soyez rigoureux sur votre rendu.

- L'évaluation vérifiera et valorisera des éléments de code, d'efficacité et d'adéquation de la solution proposée (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).

- L'utilisation d'outils ou moyens se substituant à la connaissance ou aux compétences de l'étudiant sera pénalisée en cas de détection par l'enseignant (niveau de l'étudiant dissimulé et donc non évaluable : gain potentiel passé en malus).

- Le rendu d'un travail par groupe est autorisé uniquement selon les modalités exprimées dans le support de cours (préciser collaborateurs en début de fichier).

- Le devoir est à rendre sur **MyGES** au plus tard pour la date spécifiée sur MyGES ou le planning des séances.

Bon courage !

Évaluations papiers (10 points)

Deux évaluations en classe sur papier compléteront ce travail. Ces évaluations auront lieu sur les séances suivantes :

- Première évaluation à la séance 5 (5 points) : consolidation (bases de la première année et allocation dynamique).
- Seconde évaluation à la séance 8 (5 points) : fichiers.

Consulter le planning pour les dates précises des évaluations.

Rendu d'exercices d'assiduité (max 5 points)

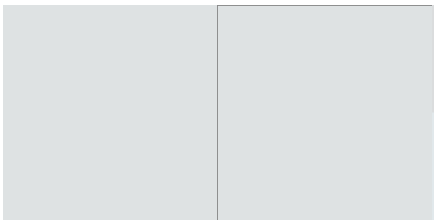
L'étudiant peut choisir de présenter son travail sur un choix de 5 exercices. Il s'agit d'une démonstration d'un travail personnel au format d'un rendu de fichiers source en Langage C. Les exercices peuvent être choisis dans la liste suivante :

- **9. Consolidation** : exercices 83 à 95.
- **10. Fichiers** : exercices 96 à 103.
- **11. Types structurés** : exercices 104 à 113.

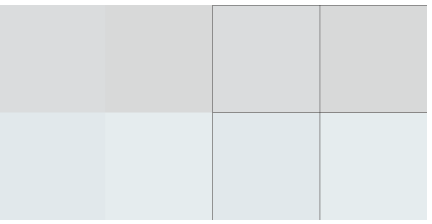
Mini-projet (5 points)

Charlie s'intéresse à la compression d'images. Il a découvert que simplifier une image pourrait la faire perdre en précision mais permettrait aussi potentiellement un stockage réduit de cette image. Oscar l'informe qu'une possibilité pour simplifier une image est les Quadtrees. L'idée c'est de découper une région en 4 régions et l'appliquer récursivement jusqu'à arriver à une région qu'il n'est plus possible de réduire comme un pixel. Chaque région pourrait posséder une couleur.

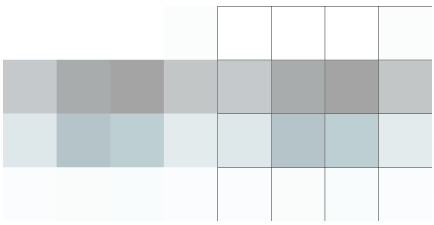
Région pleine :



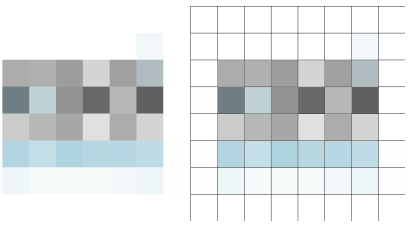
Subdivisée une fois :



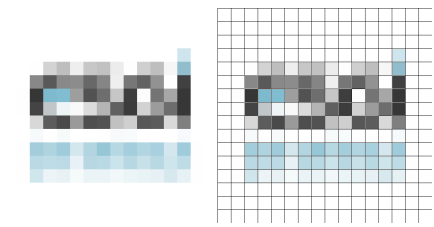
Subdivisée deux fois :



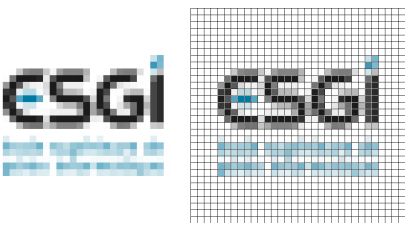
Subdivisée trois fois :



Subdivisée quatre fois :



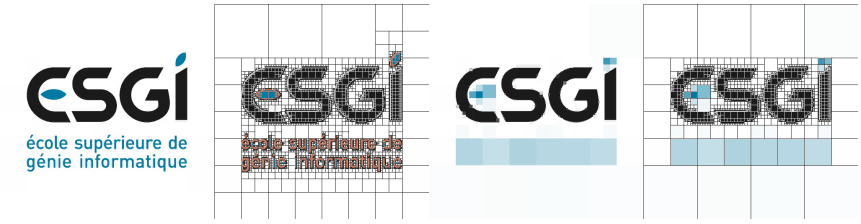
Subdivisée cinq fois :



Une fois que l'image aurait été subdivisée au maximum, une simplification pourrait être réalisée en fusionnant les nœuds qui englobent des nœuds de couleurs proches à un degré donné. Ceci pourrait se calculer comme un écart type de la valeur moyenne d'un nœud à ses nœuds enfants. Ce qui pourrait donner pour des pourcentages de similarité de couleur les résultats suivants :

Simplification précise (similarité à 0.5%) : Simplification standard (similarité à 10%) :

Simplification grossière (similarité à 28%) : Simplification absurde (similarité à 50%) :



À noter qu'Oscar vous a réservé une image au format quadtree dont les spécifications sont précisées dans la fonction de charger à coder.

```c
/* Commande de compilation via MSYS2 MinGW 64 : */
/* gcc start.c -lmingw32 -lSDLmain -lSDL -lSDL_image -lSDL_gfx */

#define SDL_MAIN_HANDLED
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 1024;
const int hauteur = 512;
const char * titre = "ESGI image simplification";
const char * target_image = "media/esgi.png";
const char * quadtree_save = "image.quadtree";
SDL_Surface * ecran = NULL;
```

```c
int max_depth = -1;
int visualize = 0;
float simplify_coeff = 0.1;

#if SDL_BYTEORDER == SDL_BIG_ENDIAN
#define RMASK 0xff000000
#define GMASK 0x00ff0000
#define BMASK 0x0000ff00
#define AMASK 0x000000ff
#define RSHIFT 24
#define GSHIFT 16
#define BSHIFT 8
#define ASHIFT 0
#else
#define RMASK 0x000000ff
#define GMASK 0x0000ff00
#define BMASK 0x00ff0000
#define AMASK 0xff000000
#define RSHIFT 0
#define GSHIFT 8
#define BSHIFT 16
#define ASHIFT 24
#endif

typedef struct Color Color;
struct Color {
  unsigned char r;
  unsigned char g;
  unsigned char b;
  unsigned char a;
};

Color get_pixel(SDL_Surface * surface, int x, int y) {
  Uint8 *p = (Uint8*)surface->pixels + (y * surface->pitch + x *
  ↪  surface->format->BytesPerPixel);
  Uint32 px = *(Uint32*)p;
  return (Color) {
    .r = (px >> RSHIFT) & 0xff,
    .g = (px >> GSHIFT) & 0xff,
    .b = (px >> BSHIFT) & 0xff,
    .a = (px >> ASHIFT) & 0xff
  };
}
```

```c
typedef struct Node Node;
struct Node; /* TODO : définir la structure */

typedef struct Image Image;
struct Image {
  SDL_Surface * surface;
  Node * quadtree;
};

void compute_quadtree(Image * image) {
  /* TODO : calculer le quadtree depuis l'image. */
  /* Découper récursivement l'image en noeuds jusqu'à atteindre un
  ↪  pixel. */
  /* Faire remonter la couleur moyenne des noeuds enfants dans le
  ↪  noeud parent. */
}

void simplify_quadtree(Image * image) {
  /* TODO : simplifier le quadtree. */
  /* Dans le cas où la distance de couleur (écart type) entre les
  ↪  couleurs */
  /* des noeuds enfants et du noeud parent est inférieure à 255 *
  ↪  simplify_coeff */
  /* alors fusionner les noeuds enfants en le noeud parent. */
  /* (les noeuds peuvent pourraient être conservés pour
  ↪  désimplifier) */
}

void reset_quadtree(Image * image) {
  /* TODO : réinitialiser le quadtree à sa forme d'origine. */
  /* Efface la simplification appliquée. */
}

Image Image_load(const char * path) {
  /* TODO : charger une image depuis un fichier (png, jpg, bmp) et
  ↪  calculer le quadtree associé. */
}

void Image_free(Image * image) {
  /* TODO : libérer l'image et le quadtree associé. */
}
```

```c
void display_surface(Image * image, int offset_x, int offset_y, int
↪  width, int height) {
  /* TODO : afficher l'image à la position (offset_x, offset_y) */
  /* en tronquant si elle dépasse les dimensions (width, height).
  ↪  */
}

void display_quadtree(Image * image, int offset_x, int offset_y, int
↪  width, int height, int max_depth) {
  /* TODO : afficher le quadtree sans la zone de la position
  ↪  (offset_x, offset_y) */
  /* avec les dimensions (width, height). */
}

void affichage(Image * image) {
  display_surface(image, 0, 0, largeur / 2, hauteur);
  display_quadtree(image, largeur / 2, 0, largeur / 2, hauteur,
  ↪  max_depth);
}

void save_quadtree(Image * image, const char * path) {
  /* TODO : sauvegarder le quadtree au chemin spécifié. */
  /* le format est le suivant pour un noeud : */
  /* écriture d'un octet (is_leaf) */
  /* si is_leaf est 1 : */
  /* - écrire Color sur sizeof(Color) octets. */
  /* sinon : */
  /* - écrire le noeud en haut à gauche */
  /* - écrire le noeud en haut à droite */
  /* - écrire le noeud en bas à gauche */
  /* - écrire le noeud en bas à droite */
}

void load_quadtree(Image * image, const char * path) {
  /* TODO : charger un quadtree depuis un chemin spécifié. */
  /* le format est le suivant pour un noeud : */
  /* lecture d'un octet (is_leaf) */
  /* si is_leaf est 1 : */
  /* - lire Color sur sizeof(Color) octets. */
  /* sinon : */
  /* - lire le noeud en haut à gauche */
  /* - lire le noeud en haut à droite */
  /* - lire le noeud en bas à gauche */
```

```
  /* - lire le noeud en bas à droite */
}

int main(int argc, char * argv[]) {

  /* Création d'une fenêtre SDL : */
  if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
    fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
    exit(EXIT_FAILURE);
  }
  if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
  ↪  SDL_DOUBLEBUF)) == NULL) {
    fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
    ↪  SDL_GetError());
    SDL_Quit();
    exit(EXIT_FAILURE);
  }
  char caption[256];
  sprintf(caption, "%s (simplify at %.2f %%)", titre, simplify_coeff
  ↪  * 100.);
  SDL_WM_SetCaption(caption, NULL);

  Image image = Image_load(argc == 1 ? target_image : argv[1]);

  int active = 1;
  SDL_Event event;

  while(active) {

    affichage(&image);
    SDL_Flip(ecran);

    SDL_WaitEvent(&event);

    switch(event.type) {
      /* Utilisateur clique sur la croix de la fenêtre : */
      case SDL_QUIT : {
        active = 0;
      } break;

      /* Utilisateur enfonce une touche du clavier : */
      case SDL_KEYDOWN : {
        switch(event.key.keysym.sym) {
```

```
          /* Touche Echap : */
          case SDLK_ESCAPE : {
            active = 0;
          } break;

          default : break;
        }
      } break;

      case SDL_KEYUP : {
        switch(event.key.keysym.sym) {
          case SDLK_LEFT : {
            if(max_depth > -1) --max_depth;
          } break;

          case SDLK_RIGHT : {
            ++max_depth;
          } break;

          case SDLK_SPACE : {
            reset_quadtree(&image);
          };

          case SDLK_UP : {
            simplify_coeff *= 1.1;
            sprintf(caption, "%s (simplify at %.2f %%)", titre,
            ↪  simplify_coeff * 100.);
            SDL_WM_SetCaption(caption, NULL);
            reset_quadtree(&image);
            simplify_quadtree(&image);
          } break;

          case SDLK_DOWN : {
            simplify_coeff /= 1.1;
            sprintf(caption, "%s (simplify at %.2f %%)", titre,
            ↪  simplify_coeff * 100.);
            SDL_WM_SetCaption(caption, NULL);
            reset_quadtree(&image);
            simplify_quadtree(&image);
          } break;

          case SDLK_v : {
            visualize = ! visualize;
```

```
          } break;

          case SDLK_x : {
            reset_quadtree(&image);
            simplify_quadtree(&image);
          } break;

          case SDLK_s : {
            save_quadtree(&image, quadtree_save);
          } break;

          case SDLK_l : {
            load_quadtree(&image, quadtree_save);
          } break;

          default : break;
        }
      } break;

      default : break;
    }
  }

  Image_free(&image);

  SDL_FreeSurface(ecran);
  SDL_Quit();
  exit(EXIT_SUCCESS);
}
```

Attendus :

- Construction d'un quadtree pour une image donnée.
- Simplification par écart type à la couleur moyenne.
- Simplification par valeur moyenne.
- Affichage de l'image et sa simplification.
- Chargement et sauvegarder dans un fichier binaire.