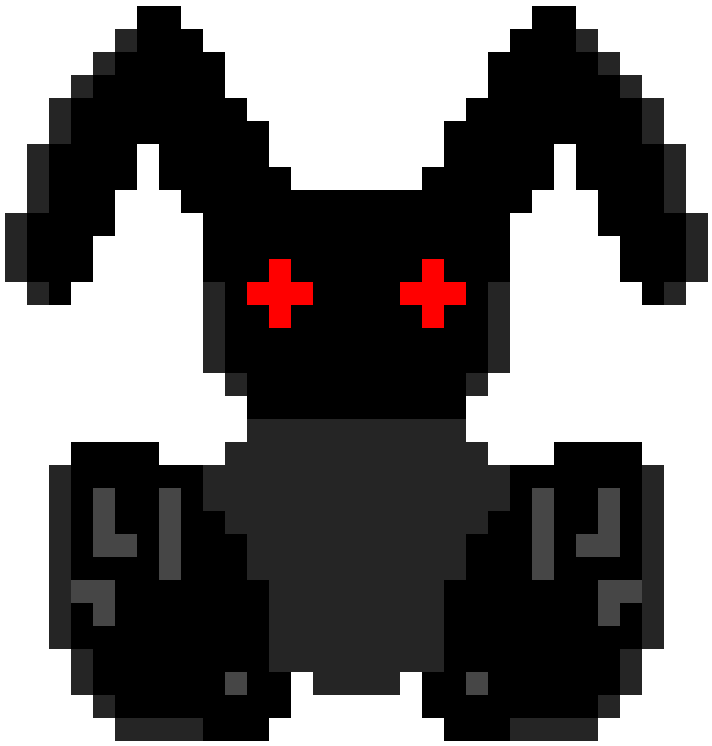
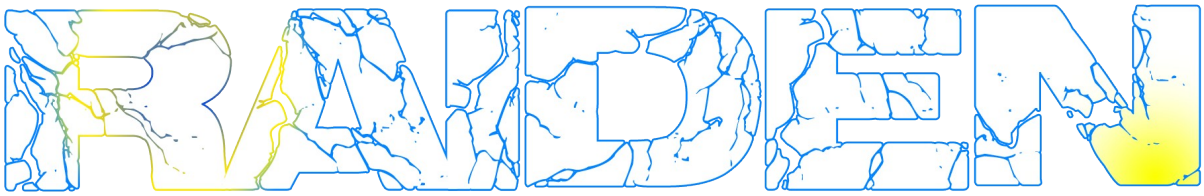


LAPINS NOIRS



Tracé de lignes

- La Caverne aux Lapins Noirs -

INDEX

01 – Avant-propos

Votre travail doit être rendu via le dossier **~/tp/spiral/** dans votre espace personnel.

Si vous faites erreur et que le dossier que vous utilisez pour votre rendu est différent, vous ne serez pas évalué faute d'avoir pu trouver votre travail.

Ce travail est a effectuer seul. Vous pouvez bien sur échanger avec vos camarades, néanmoins vous devez être l'auteur de votre travail. Utiliser le code d'un autre, c'est **tricher**. Et tricher annule **toutes** les médailles que vous avez reçu sur l'activité. La vérification de la triche est réalisée de la même manière que la correction : de manière **automatique**. Prenez garde si vous pensez pouvoir passer au travers.

Votre rendu doit respecter **strictement** l'ensemble des règles suivantes :

- Il ne doit contenir **aucun** fichier objet. (*.o)
- Il ne doit contenir **aucun** fichier tampon. (*~, #*#)
- Il ne doit pas contenir votre production finale (programme ou bibliothèque)

La présence d'un fichier interdit mettra immédiatement fin à votre évaluation.

Votre programme doit respecter les Tables de la Norme dans leur intégralité. Vous êtes invité à les observer depuis **l'Infosphère**. Elles sont disponibles comme ressource de cette activité.

02 – Fonctions autorisées

La bibliothèque logicielle venant avec le C est vaste et disponible. La LibLapin, que vous utilisez dans vos projets multimédia, est également vaste… Cependant nous avons fait le choix de vous interdire leurs utilisation intégrales, afin de vous amener progressivement à reprogrammer vous même ses fonctionnalités les plus utiles.

L'utilisation d'une fonction interdite est assimilée à de la triche. La triche provoque l'arrêt de l'évaluation et la perte des médailles.

Vous n'avez le droit d'utiliser aucune fonction issue de la LibC ou de la LibLapin à l'exception de celles que nous vous autoriserons explicitement.

Pouvoir utiliser une fonction ne signifie pas nécessairement que celle-ci soit utile à votre cas.

Pour cette activité, issu de la LibC, vous n'avez le droit qu'à la liste suivante :

- open
- close
- read
- srand
- cos
- atan2
- write
- alloca
- atexit
- rand
- sin
- sqrt

Remarquez bien l'absence de **malloc** et de **free**. En effet ils sont **interdits** ! Issu de la LibLapin, vous avez le droit aux fonctions suivantes :

- bunny_start
- bunny_stop
- **bunny_malloc**
- **bunny_free**
- bunny_new_pixelarray
- bunny_delete_clipable
- bunny_blit
- bunny_display
- bunny_set_*_function
- bunny_set_*_response
- bunny_loop
- bunny_create_effect
- bunny_compute_effect
- bunny_sound_play
- bunny_sound_stop
- bunny_delete_sound
- bunny_open_configuration
- bunny_delete_configuration
- bunny_configuration_getf
- bunny_configuration_setf
- bunny_configuration_*
- bunny_set_memory_check
- bunny_release
- bunny_usleep

La suite sur la page d'après.

Pour utiliser **bunny_malloc**, vous pouvez soit programme directement avec, soit en utilisant le modèle de projet qui vous a été transmis, mettre 1 dans la variable de Makefile **BMALLOC**, qui transformera **malloc** en **bunny_malloc**.

Vous appellerez **bunny_set_memory_check** au début de votre fonction **main** de sorte à provoquer une vérification de vos allocations à la fermeture du programme.

bunny_malloc, par défaut, limitera votre consommation de RAM à 20Mo.

Pour information :
Une image en 1920*1080 fait environ 8Mo.

Une musique en 44kHz de 1 minute en stéréo fait environ 10Mo.

Vous devrez donc disposer d'une discipline de fer avec vos allocations... et probablement trouver des compromis.

L'utilisation de **bunny_malloc** parfois **cachera** des erreurs dans votre programme, et parfois en **révélera** : son principe d'allocation était différent de **malloc**, il sera parfois plus « fort » ou plus « faible ». Ne vous mentez pas à vous en disant « l'utilisation de **bunny_malloc** fait planter mon programme », ce n'est pas **bunny_malloc**, c'est *vous*.

N'hésitez pas à l'activer, à le désactiver (Et lorsqu'il est désactivé, à utiliser valgrind). Et n'oubliez pas que désormais, votre demande de RAM a de véritables chances d'échouer. Car exploiter aussi peu de RAM, cela va très vite…

Dans votre rendu, il ne devra pas y avoir la moindre trace de malloc.

03 – Méthode de construction

Il peut vous être demandé d'écrire des programmes ou des fonctions.

Dans le cas des programmes, il vous sera toujours demandé de fournir un **dossier** pour l'exercice le requérant. Un **Makefile** vous sera également demandé. Le **nom du programme** de sortie vous sera précisé à chaque fois. Un Makefile incorrect, un mauvais nom de programme, et votre correction n'aura pas lieu…

Dans le cadre des fonctions, il vous ai demandé de fournir le fichier dans votre **dossier de bibliothèque personnelle**, de sorte à ce que vous puissiez utiliser toutes les fonctions que vous avez déjà réalisé jusqu'ici. Pour rappel, le dossier de votre bibliothèque doit être placé à la racine de votre espace personnel et s'appeler **libstd/.**

N'oubliez pas d'entretenir avec soin votre dossier **libstd/** de sorte à ce qu'il soit toujours propre, respecte la norme et soit en état de compiler… sans quoi elle fera obstacle à la correction.

Votre compilation devra toujours comporter les options **-W**, **-Wall** et **-Werror**.

Dans le cadre de la programmation multimédia, le système de correction établira toujours la variable d'environnement **BMALLOC** à 1. Si vous utilisez le modèle de projet, cela provoquera l'utilisation de **bunny_malloc** dans votre bibliothèque personnelle comme dans votre projet rendu.

04 – Fonction affine

Vous rappelez vous de ce qu'est une fonction affine ? Et l'équation de la courbe qui la représente ?

```
y = ax + b
```

Avec x étant le paramètre de la fonction, a étant le coefficient directeur de la fonction et b étant sa base. Le résultat y étant donc la position sur l'ordonnée du point associé à la position x sur l'abscisse.

Le sujet du jour étant le tracé de ligne, nous allons avoir besoin de savoir quelle est la fonction qui nous permettrait de tracer une ligne entre deux points. Comment constituer une équation à partir de deux points ?

En trouvant le coefficient directeur de la droite liant les deux points, vous pourrez ensuite parcourir les pixels depuis un point vers l'autre. Vous rappelez vous comment calculer le coefficient directeur ? Si ce n'est pas le cas, n'hésitez pas à chercher sur internet !

Ces fonctions vous seront également utile. Programmez les :

```
double std_get_ratio(double value,
                     double min,
                     double max);
double std_get_value(double ratio,
                     double min,
                     double max);
```

La fonction get_ratio calcule le pourcentage auquel est value dans l'échelle [min;max].
La fonction get_value calcule la valeur que représente le pourcentage envoyé sur l'échelle [min;max].

Le terme « pourcentage » représente un coefficient compris entre 0 et 1.

05 – Une ligne imparfaite

Mettre dans le fichier : set_line.c

Programmez la fonction suivante :

```
void std_set_line(t_bunny_pixelarray    *picture,
                  t_bunny_position      *position,
                  unsigned int          *color);
```

Cette fonction trace une ligne depuis position[0] jusqu'à position[1]. La couleur de cette ligne sera color[0]. Un couleur se situera également dans color[1] mais nous allons l'ignorer pour l'instant. Le dessin a lieu dans picture.

Comment faire ?

Vous allez partir de position[0].x jusqu'à position[1].x, ajoutant à position[0].y le coefficient directeur pour chaque pixel.

Ci-dessous, un morceau de programme de test afin de vous aider.

```
void                            linedisc(t_bunny_pixelarray         *px)
{
        unsigned int            color[2];
        t_bunny_position        pos[2];
        double                  i;

        i = 0;
        color[0] = WHITE;
        color[1] = WHITE;
        pos[0].x = px->clipable.buffer.width / 2;
        pos[0].y = px->clipable.buffer.height / 2;
        while (i < 2 * M_PI)
        {
                pos[1].x = px->clipable.buffer.width * (0.5 + cos(i));
                pos[1].y = px->clipable.buffer.height * (0.5 + sin(i));
                std_set_line(px, &pos[0], &color[0]);
                i += M_PI / 16;
        }
}
```

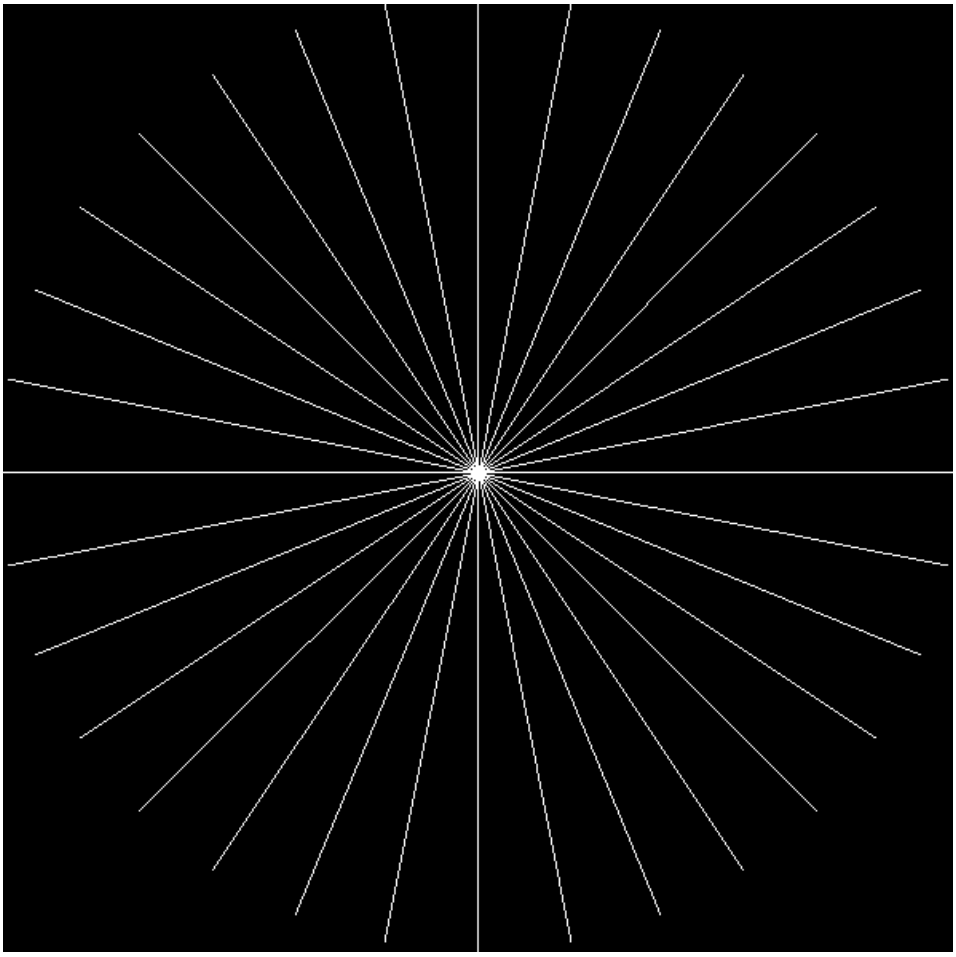
Vous aurez une fois terminé le défaut de la méthode : il n'y a qu'un seul pixel par colonne . Normal : vous itérez sur x. Ce n'est pas un problème quand le coefficient directeur est compris entre certaines valeurs, quand la pente est douce… Mais quand elle est forte, cela ne marche plus.

06 – Une ligne parfaite

Mettre dans le fichier : set_line.c

Pour régler le problème, lorsque la pente est forte, vous devez donc, à la place d'itérer sur x… itérer sur y.

Modifiez la fonction afin que la ligne devienne parfaite.



07 – Un éclair !

Mettre dans le fichier : raiden.c

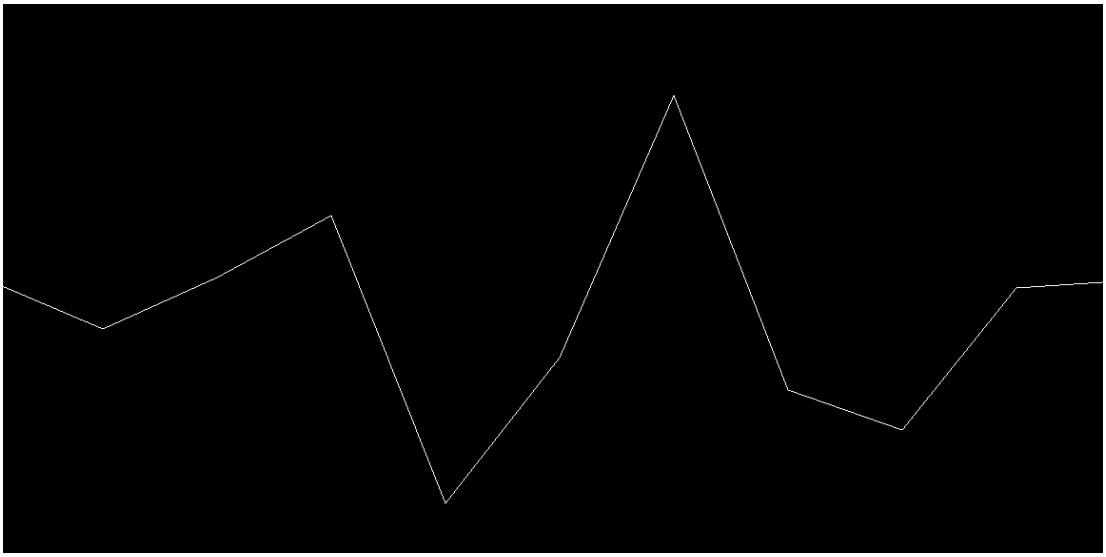
Écrivez la fonction suivante :

```
void std_raiden(t_bunny_pixelarray      *picture,
                size_t                  angles);
```

Le paramètre **angles** indique le nombre de cassures dans une ligne brisée à dessiner à l'écran. La position en x de chaque angle est déterminé directement par la largeur de l'image divisé par **angles**. La hauteur en y est déterminé par un appel à la fonction **rand**.

Un **unique** appel à **rand** par position de x est permis, cela afin de permettre au logiciel de correction de reproduire le comportement de votre fonction. Vous devrez dessiner votre ligne brisé de gauche à droite.

Les positions tout à gauche et tout à droite de l'image doivent être situé à mi-hauteur et ne sont pas déterminées par l'aléatoire.



10 – Pour finir

SI vous êtes arrivé jusqu'ici, déjà, félicitations !

Maintenant que votre fonction **std_raiden** fonctionne, voici une proposition. Ce n'est pas un exercice, c'est simplement parce que c'est chouette. Écrivez un programme qui appelle en boucle la fonction **td_raiden** avec des valeurs aléatoires non nulle et affichez l'image à chaque fois. Ça bouge, on dirait presque un éclair.

Si le cœur vous en dit, écrivez des variations de **std_raiden** pour améliorer l'esthétique de votre programme.