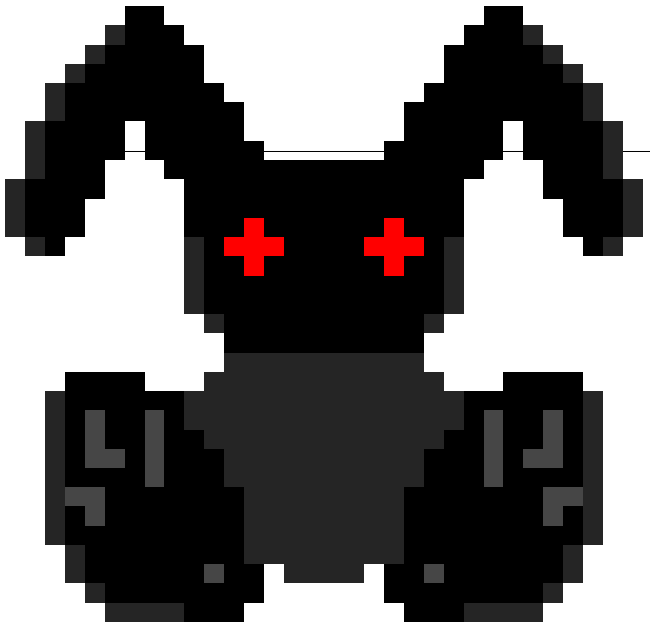
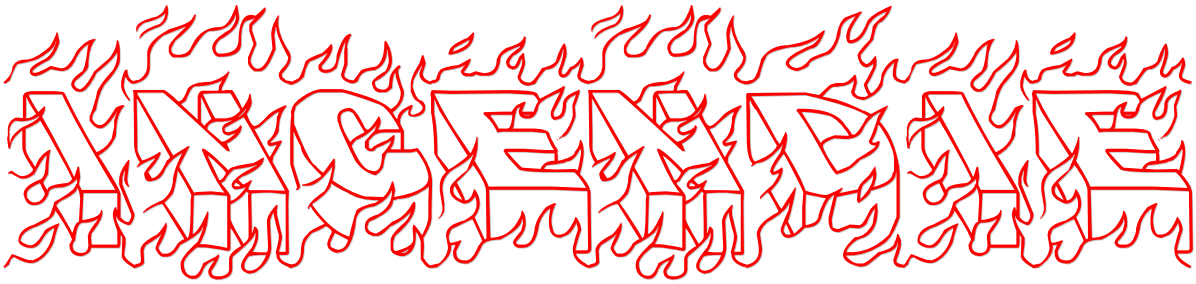


LAPINS NOIRS

- La Caverne Aux Lapins Noirs -



Lors de cette ruée, vous allez programmer de petits effets graphiques.

Auteur : Jason Brillante

INDEX

01 – Avant-propos

Votre travail doit être rendu via les dossiers ~/rue/incendie/, ~/rue/plasma/ et ~/rue/indexpic/ 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 en binôme. 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/.**

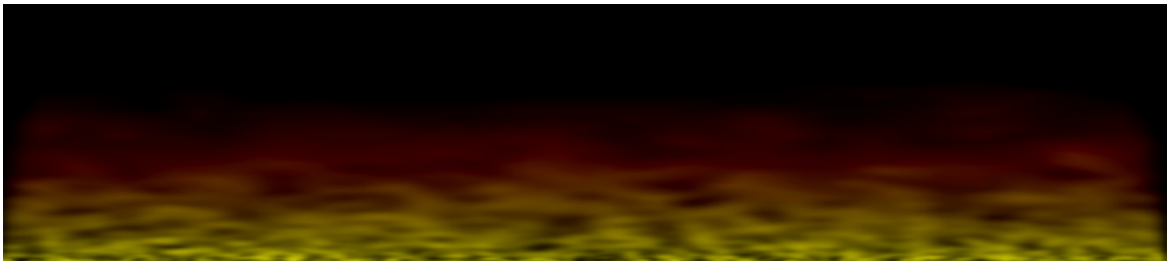
N'oubliez pas d'entretenir avec soin votre dossier **libstd/** de sorte à ce qu'il soit toujours propre, respecte la norme et soit en état de compiler… sans quoi elle fera obstacle à la correction.

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 – Flammes

Ce projet aborde le sujet des palettes de couleur ainsi que les **automates cellulaires**. Son principal objectif est de vous amener à manipuler de manière inhabituelle une image en vue d'y produire des effets que *vous ne pensiez pas arriver à produire*. À l'issu de cette ruée, vous devriez trouver l'utilisation de l'écran plus confortable. Le premier exercice consiste à écrire un programme **incendie** affichant des flammes à l'écran.



Commencez par construire une palette de couleur : il s'agit d'un tableau d'une certaine taille contenant des couleurs. Votre tableau pour cette méthode devra mesurer 128 couleurs de long.

La couleur à la position 0 est noire.
Les couleurs de 1 à 32 forment un dégradé de noir à rouge.
Les couleurs de 33 à 64 forment un dégradé de rouge à jaune.
Les couleurs de 65 à 96 forment un dégradé de jaune à blanc.
Les couleurs de 97 à 128 sont toutes blanches.

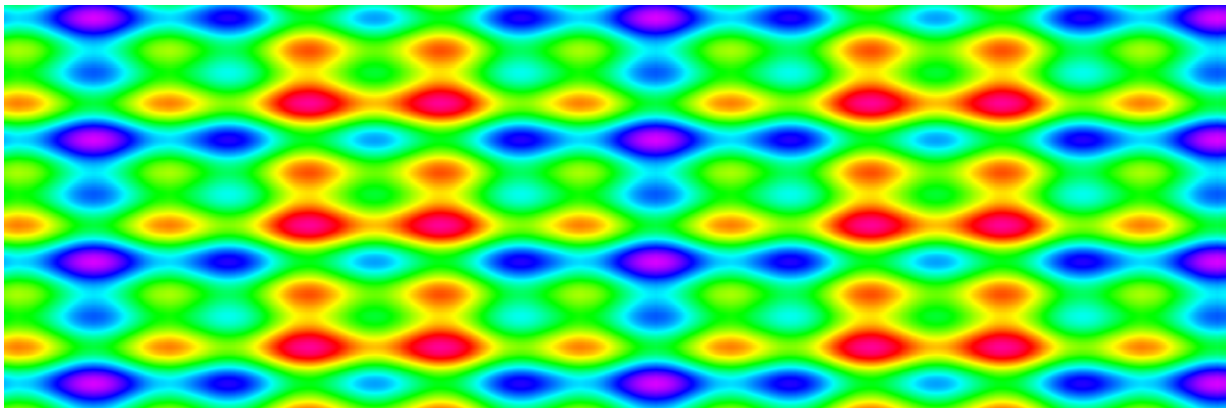
Créez ensuite un espace mémoire faisant la taille de votre fenêtre plus l'équivalent de 2 lignes de hauteur. Le type des cases de cet espace devra pouvoir contenir au moins des valeurs entre 0 et 128. Au départ, toutes ses valeurs devront être égales à zéro. Cet espace mémoire correspond à votre image à l'écran mais n'en est pas une : vous devrez le convertir pour l'afficher.

Dans les 2 lignes situés au bas de cet espace, appliquez des nombre aléatoires allant de 0 à 128, cela pour correspondre à votre planche de couleur.
Ensuite, pour chaque **« pixel »** de cet espace à l'exception des lignes de nombre aléatoire, calculez la moyenne des voisins et assignez cette valeur au pixel au-dessus.

Copiez ensuite cet espace dans une image et affichez le.
Vous êtes évidemment libre de modifier cet algorithme.
L'important est l'esthétique du programme.
Une bonne idée : Mettre de l'aléatoire dans sa moyenne.

05 – Plasma

La seconde étape consiste à programmer un effet plasma, dans le programme **plasma** :



Vous aurez besoin de créer une palette de couleurs. Vous pouvez faire sans, mais vous aurez alors à jouer avec les composantes de couleurs et cela peut s'avérer plus complexe.

Vous aurez besoin d'utiliser cos et/ou sin et d'additionner plusieurs sinusoïdes ensemble : Imaginez par exemple que vous parcouriez l'image de gauche à droite avec la valeur d'un cosinus, évoluant en fonction de X, multiplié par 128 pour chaque colonne pour trouver la couleur associée.

Pouvez-vous croiser des sinusoïdes ? Bien sur.

Comment améliorer cet effet ? Pourquoi ne pas faire varier doucement la longueur des sinusoïdes ?

06 – Couleurs indexées

Vous trouverez des fichiers .dab. La structure de ce fichier suivra le principe suivant :

```
Width = 10
Height = 10
Frequency = 10

{Colors
      Rouge, Vert, Bleu,
      Rouge, Vert, Bleu,
      Rouge, Vert, Bleu,
      Rouge, Vert, Bleu
}

{Pixels
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
      3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
      3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
      3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
}

{Rotation
      {
            2,
            3
      },
      {
            3,
            2
      }
}
```

Votre programme indexpic prendra en paramètre un fichier .dab et ouvrira une fenêtre de la taille de l'image renseignée dans ce fichier. Ici, l'image fait 10 pixels sur 10 pixels. Chaque pixel est représenté par un numéro correspondant à un triplet « Rouge vert bleu ».

Le tableau rotation indique quel permutations doivent être faites à la fréquence donnée (Dans un premier temps, contentez vous de changer à chaque affichage). Par exemple, ici, les pixels de valeur 2 deviennent des pixels de valeur 3, avant à nouveau de rechanger.