LEFORT Vincent

Algorithme de calcul de rendu d'images et gestion des masques

2 Minutes

Stage effectué du 11/03/05 au 15/07/05 au sein de l'entreprise « 2minutes » à Paris 12ème.

Rapport de stage de Licence Professionnelle Ingénierie de l'Imagerie numérique
IUT de Reims-Châlons-Charleville

Remerciements

Je souhaite tout d'abord remercier chaleureusement, Mr Jean-Michel Spiner pour m'avoir accueilli dans son studio d'animation et d'avoir toujours su répondre à mes demandes.

J'exprime ma profonde reconnaissance à Monsieur Benoît DEHAENE, chef de l'équipe de Recherche et Développement qui a accepté d'être mon maître de stage et qui a toujours été là pour répondre à mes diverses questions.

Je remercie également Mr Jérôme Fromeaux, employé de 2minutes, d'avoir également répondu à mes questions et d'avoir été là en cas de problèmes d'ordinateur.

Je tiens aussi à remercier tous les gens que j'ai côtoyés durant ces quatorze semaines, Quentin FRANCOTTE, François Hurtaux, Fabrice Guevel, Yann Provost, Xavier Parrias, Stéphane Lezoray et Mireille Sarrazin, qui ont tous été très aimables envers moi et qui m'ont permis de m'adapter rapidement à ce nouveau cadre de travail.

J'aimerais aussi remercier Mr Pierre Emmanuel GOUGELET, vacataire à l'IUT Reims-Châlons-Charleville pour m'avoir permis de trouver se stage et renseigné sur différents points en traitement d'images.

Enfin je voudrais remercier tous mes enseignants de l'IUT Reims-Châlons-Charleville qui m'ont enseigné une grande partie des connaissances nécessaires à la réalisation de ce projet.

Résumé du stage

Ce stage a consisté à :

- Calculer des rendus d'images avec rotation, des changements d'échelles, des translations le plus rapidement et de la meilleure qualité possible

- Gérer les masques et filtres sur des images et créer différents filtres.

Mot clefs : rendu d'images, bilinéaire, masques, filtres, traitement d'image, RTTI.

Sommaire

Liste des annexes

Liste des figures

Introduction

Après un nombre important d'envois de lettres de candidature et de Curriculum Vitae dans la Marne et dans la région parisienne, j'ai reçu un avis positif du studio d'animation 2minutes à Paris, grâce aux connaissances de Mr Pierre Emmanuel GOUGELET.

J'ai eu par la suite un entretien avec le directeur de ce studio et mon futur maître de stage qui m'ont proposé de travailler sur des algorithmes de rendu d'images bilinéaire et par la suite de m'occuper de la gestion des masques d'images et des filtres d'images.

Ce travail m'a tout de suite intéressé car il me permettait de mettre en application l'enseignement que j'avais reçu lors de cette année de licence professionnelle à l'Institut Universitaire de Technologie de Reims-Chalôns-Charleville site de Reims.

Dans un premier temps, je présenterai le studio 2minutes : sa localisation géographique, les séries réalisées, ainsi que le personnel qui y travaille.

Dans un second temps, je développerai mon travail personnel : les méthodes, les résultats et les problèmes rencontrées lors de ce stage effectué durant ces 14 semaines au studio.

Présentation de l'entreprise

1. Présentation globale.

Le studio 2 minutes est une SARL au capital de 38112 € créée en octobre 2000 par Jean-Michel Spiner. Ce dernier ayant une expérience de plus de 15 ans dans le domaine de l'animation (notamment 2D), acquise dans les entreprises pionnières du secteur (Pixibox et TouTenKartoon).

L'activité principale de l'entreprise est la fabrication de séries d'animation pour la télévision et de films d'animation pour le cinéma.

L'objectif, de 2minutes dès sa création, a été de développer la capacité de prendre en charge l'intégralité de la chaîne de fabrication. En effet, le studio n'est que prestataire de service, mais 2minutes commence à produire ces propres séries.

Par la suite, le studio d'animation 2minutes s'est agrandi en ouvrant un second studio à ANGOULEME (département 16) en mars 2003, puis un troisième à TROIS RIVIERES au Canada en mars 2004.

Voici les différentes adresses des studios 2minutes :

2minutes Paris
9 rue Biscornet
75012 Paris
France

2minutes Trois Rivières
4905 rue Belle Feuille
Code Postal 22006
Trois Rivières (Québec)
G9A 6N6
Canada

2minutes Angoulême
1 rue Saint Cybard
16000 Angoulême
France

Le studio Parisien, se trouve à moins de 5 minutes à pied de la place de la Bastille dans le 12ème arrondissement.

2. Productions réalisées.

Vous trouverez en annexe A la chaîne de fabrication d'un dessin animé.

Je vais vous faire ici, une liste non exhaustive, des différentes productions et prestations réalisées par 2minutes :

Production

- YAKARI 52 x 13' © STORIMAGES/ 2 MINUTES / FRANCE3 (en cours)
- L'ÂNE TROTRO 78 x 3'30 © STORIMAGES/ 2 MINUTES / FRANCE5

Prestation

- INVISIBLE MAN 26 x 26 ' © ANTEFILMS / M6 (en cours)
- LES ZINZINS 2 52 x 13 ' © XILAM / FRANCE3 (en cours)
- TOMTOM ET NANA 1 x 26' © JUFOX / FRANCE3
- ATOMIC BETTY 26 x 13' © TELEIMAGES / M6
- PARKER & BADGER 52 x 1' © DUPUIS / CANAL J
- FLATMANIA 52 x 13' © FUTURIKON / FRANCE3
- LES DURS DU MUR 39 x 7' © B PRODS / CARRERE / FRANCE2
- LA FAMILLE BOBO 30 x 1' © LPA
- AUTRES LOUPS 4 x 7' © PRIMA-LINEA (35 mm)
- HP 3 x 1' © B PRODUCTIONS
- A COW A CAT, ... pilote 4' © FUTURIKON
- LES VOEUX DE FRANCE5 17 x 30" © FRANCE5
- CATWALK générique © LOOPING / NHK (HDTV)
- CEDRIC 52 x 13 ' © DUPUIS / FRANCE3
- TOTALLY SPIES 6 x 26 ' © MARATHON / TF1
- NORMAN NORMAL 2 26 x 26 ' © TELEIMAGES / TF1

Cette liste contient le nom de la série, suivi du nombre d'épisodes et de la durée de chacun (52x13' ➔ 52 épisodes de 13 minutes), des producteurs ou des autres boites de productions intervenant dans la création de la série.

Cette liste ne présente pas tous les travaux sur lesquels 2minutes a travaillé, elle ne contient pas tous les pilotes qu'ils ont réalisés.

Un pilote est comme une bande annonce au cinéma, son but est de montrer le type de dessins animés, les personnages principaux,… Dans le but de faire financer les productions de la série par les chaînes de télévision.

3. Organigramme des salariés.

L'entreprise 2minutes possède un nombre très variant de salariés. En effet, la plupart des employés sont des intermittents du spectacle ayant des contrats au mois et renouvelés suivant le besoin des productions.

L'entreprise comporte seulement cinq personnes embauchées à plein temps. Ceux-ci sont regroupés sur l'organigramme sur la figure 1, représentant la hiérarchie au sein du studio.

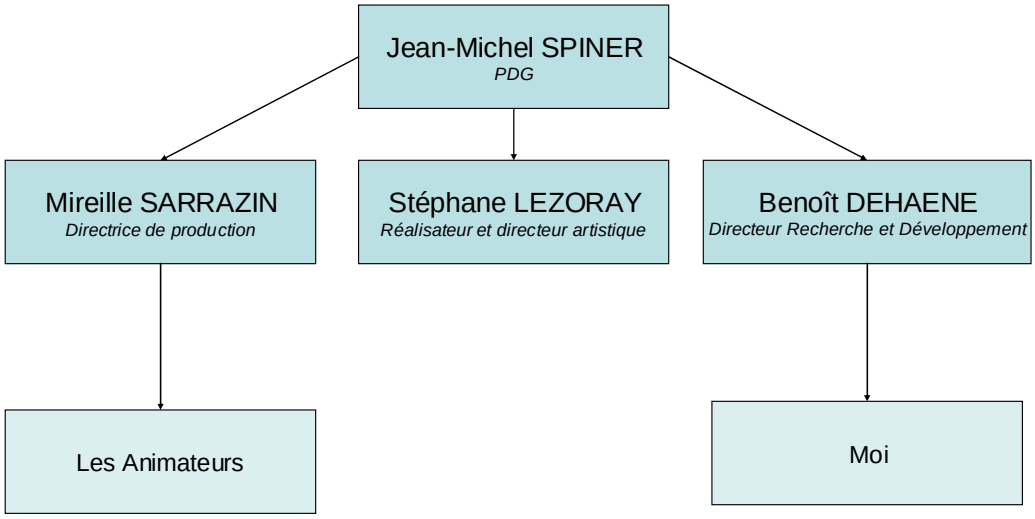

Fig 1. : Organigramme des salariés.

En ce moment, il y a environ vingt personnes travaillant à Paris (dont 2 stagiaires), cinquante à Angoulême et dix à Trois Rivières.

Pour l'anecdote, il est arrivé d'être jusque 60 personnes dans les locaux Parisiens.

Présentation du travail personnel

1. Contexte du stage.

L'entreprise 2minutes développe depuis plusieurs années, un logiciel d'animation nommé GASP qui permet de réaliser des séries en papier découpés.

Ce logiciel (qui avant était plusieurs logiciels séparés) a subit une refonte au début de l'année pour pouvoir agrandir la gamme des dessins animés produits. Cette version devrait être finie courant août pour pouvoir réaliser la série « Le chat, la vache et l'océan© » puis la série « Pop Secret© » produit par FUTURIKON.

Vous pouvez voir sur l'image suivante (Fig 2.) une capture d'écran de GASP.

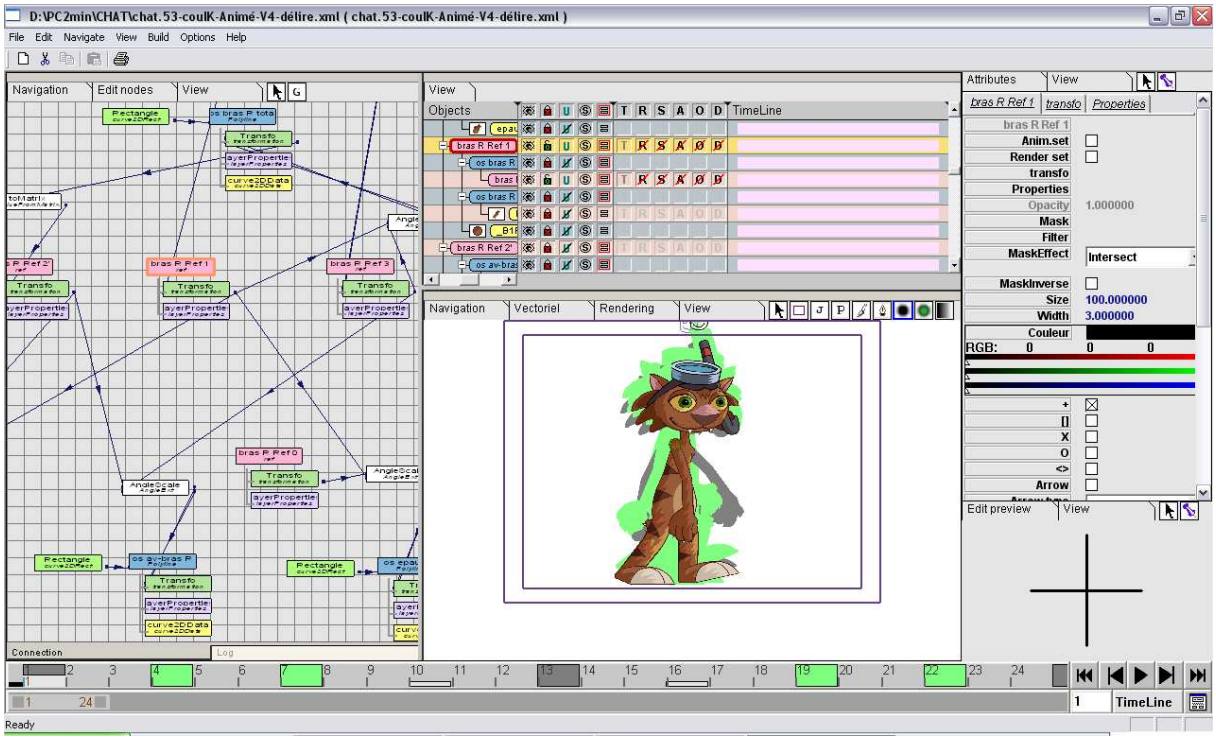

Fig 2. : Capture écran GASP.

Une démonstration du logiciel sera présentée lors de la soutenance orale.

Dans ce logiciel, on pouvait animer des images, mais on ne pouvait pas sortir les animations images par images ou par vidéos : c'est donc là que j'interviens, afin de pouvoir sortir une image de la scène actuelle de la meilleure qualité possible.

2. Résumé du travail demandé.

Dans un premier temps, il m'a été demandé de réaliser un petit algorithme qui permet de construire un arbre, de la hiérarchie des classes du programme. Le but de ce travail, fut de prendre connaissance avec le logiciel, les différentes fonctions existantes, les différentes classes…

Dans un second temps, j'ai réalisé un algorithme de rendu d'image, qui permettait de rendre des images ayant subi des transformations quelconques. Ce rendu devait gérer les fragments de pixels, la transparence… Le but de ce travail été de pouvoir sortir des images de la scène actuelle, ou même sortir une image sur laquelle était composité (rendu) plusieurs images.

Dans un troisième et dernier temps, j'ai géré des masques et des filtres (toujours dans le rendu) puis, créé différents filtres qui sont utilisés dans les dessins animés. Ce travail avait pour but de développer, un peu plus le rendu, pour essayer de le rendre le plus complet possible.

3. Hierarchy Chart.

Le premier travail qui ma été demandé, était de sortir dans un ficher texte l'arborescence des objets qui étaient présents dans le programme.

Dans GASP, il y a un « vector » (qui n'est rien d'autre qu'un tableau), qui stocke tous les objets qu'il est possible de créer dans l'application. Tous les objets étant munis d'une fonction Create, permettent de pouvoir créer dynamiquement tous les objets voulus directement du programme. Cela permet à l'utilisateur de pouvoir créer intéractivement tous les objets qu'il désire à l'aide d'un menu où sont répertoriés tous les éléments que l'on peut construire.

Le principal intérêt de ce travail fut d'apprendre les RTTI mais aussi de prendre mes repères dans la centaine de fichiers qui constituaient le projet.

3.1. Les RTTI

RTTI signifie « Run Time Type Information » soit en français « Informations de Type disponibles à l'Exécution ». Les RTTI sont une suite de fonction, standard au langage C (il existe un débat sur les RTTI, certain disent qu'ils font partie de la librairie standard C++, d'autre que non, mais là n'est pas le sujet), qui permettent « d'identifier » les pointeurs ou plutôt de savoir de quels types il sont.

Je ne vais pas entrer plus dans le détail, en effet cette partie n'est pas évidente à expliquer et il faut avoir certaines bases en C++ pour comprendre.

Pour plus d'informations, il existe beaucoup de sites expliquant ce jeu de fonctions sur Internet.

3.2. Résultat

Vous pouvez voir sur l'image suivante (Fig 3), un résultat de l'algorithme.

Sur cette figure, on peut voir la hiérarchie des classes présentes dans GASP, ainsi que les attributs de ces classes. Cette figure représente une petite partie des 20 pages.

```
   unsigned Resolution
   float Radius
   float Radius(small)
   ObjAttribute data *
class COpalCurve2DRectObject
   ObjReference Rectangle *
   float Width
   float Height
   ObjAttribute data *
class COpalRawDataObject
   ObjReference rawData *
class COpalDataSourceObject
   ObjReference dataSource *
   class COpalRsrcObject
      ObjReference rsrc *
      class COpalFileRsrcObject
         ObjReference fileRsrc *
         string Filename *
         string Pathname *
         string Extension *
         unsigned Mill.sec *
         class COpalSoundFileRsrcObject
            ObjReference soundFile *
            string Filename *
            string Pathname *
            string Extension *
            unsigned Mill.sec *
         class COpalImageFileRsrcObject
            ObjReference imageFile *
            string Filename *
            string Pathname *
            string Extension *
            unsigned Mill.sec *
            unsigned Reso.X *
```

Fig. 3 : Hierarchy Chart.

4. Rendu d'images en software.

4.1. Présentation.

4.1.1. Présentation globale.

Pour faire une vidéo, il faut, pour ne pas avoir de gêne visuelle, avoir un défilement de minimum 25 images par secondes.

GASP est un logiciel d'animation en temps réel, qui permet de voir le résultat de la scène sans pré calcul. Il utilise OpenGL pour réaliser ses affichages d'images, ces masques… OpenGL ne permet pas de sortir des images de la scène actuellement visible (il le peux mais la qualité n'est pas de tous reproche).

C’est donc là que j’interviens, pour effectuer le calcul de la scène actuelle et sortir une image avec le minimum d’imperfection possible. Et c’est à partir des ces images, mises bout à bout, que cela va créer un film.

Jusqu’à présent, 2minutes utilisait des logiciels commerciaux comme Flash pour l’animation, et Adobe After Effect ® pour réaliser les vidéos, les différents effets spéciaux…

J’avais comme cahier des charges, réaliser ce travail et faire en sorte que le calcul de rendu d’images soit plus rapide et de meilleur qualité que After Effect® (car celui-ci n’est pas irréprochable non plus). Je vous parlerai, durant mes explications, des différentes optimisations réalisées.

Ce travail, s’est déroulé en plusieurs étapes. La première consistait a calculer le rendu d’images le plus basiquement possible, c'est-à-dire avec une qualité moindre. Puis de réaliser la même chose avec une qualité de rendu meilleur.

Il y a donc un rendu réalisé avec une qualité moindre que l’autre, cela permet d’avoir un aperçu de la scène rapidement. En effet, le temps de calcul est plus long avec une qualité meilleure, c’est pour cela que j’ai réalisé deux qualités de rendu différentes, afin de pouvoir visualiser la scène quasi-finale plus rapidement.

4.1.2. Outils à ma disposition.

Pour calculer le rendu d’une image dans mon buffer résultat (qui est un champ de ma classe), le programme appelle la fonction `process` qui prend en paramètre l’image dont il faut que je calcule le rendu.

De plus il y a en champ private (privé → accessible uniquement par les fonctions internes à ma classe) de ma classe, une matrice de transformation qui permet de passer de l’image de base (sans rotation, sans changement d’échelle, sans translation) à l’image transformée. Cette matrice est très utile car c’est elle qui me permet de calculer les quatre coins de l’image transformée pour en calculer son rendu.

De plus, toutes les images que je gère, sont des images 32 bits avec une couche alpha. Chaque pixels de l’image à une composante RGBA qui correspond au Red(rouge), au Green (vert), au Blue(bleu) et de l’Alpha. Chacune de ces valeurs est codé sur un unsigned char dont les valeurs vont de 0 à 255. L’alpha permet de savoir si le pixel est plus ou moins visible (Fig 4).

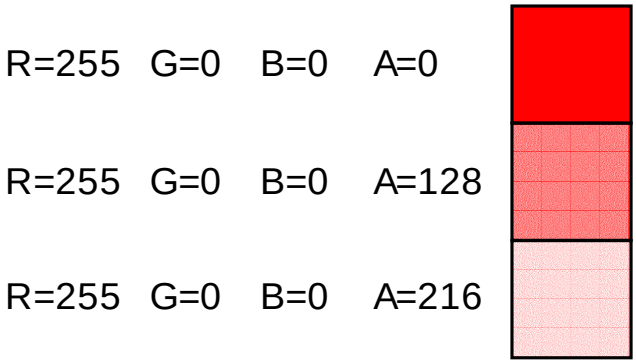

Fig 4. : Explications Alpha.

La couleur finale du pixel dépend de la couleur de fond de l'image, en effet pour obtenir la couleur finale d'un pixel on effectuer pour chaque composante RGB le calcul suivant :

$$composanteFinale = \frac{composanteInitial * Alpha + (255 - Alpha) * couleurFond}{255}$$

Cette formule, est très utile car c'est grâce à elle que l'on va obtenir la véritable couleur du pixel.

4.1.3. Vocabulaire utilisé fréquemment.

Je vais essayer d'instaurer un vocabulaire que je vais définir ici et que j'utiliserai tout au long de mes explications :

- J'appellerai buffer l'image dans laquelle je stocke mon rendu final.
- imgData ou image Data l'image dont je cherche à calculer le rendu.
- Je nommerai par RGBA les différentes composantes des pixels.

4.2. Calcul de rendu d'images sans rotation.

4.2.1. Rendu de faible qualité.

Dans ces conditions, le rendu est le plus facile à calculer. En effet, il suffit de parcourir le buffer résultat, de regarder pour chaque pixel dans ce buffer la couleur du pixel dans l'image Data et faire le traitement de l'alpha expliqué au dessus.

Pour se décaler correctement dans l'image Data il faut tout d'abord calculer son ratio, c'est-à-dire le coefficient de changement d'échelle sur les deux axes X et Y (largeur et hauteur de l'image).

Exemple :

Nous souhaitons écrire dans une zone de 100 pixels de largeur et 100 pixels de hauteur une image *Im* qui fait 500 pixels de largeur et 500 pixels de hauteur. Cela revient à dire que l'image *Im* a été réduite de 5 en largeur et de 5 en hauteur. L'opération pour trouver la couleur du pixel courrant consiste à multiplier la position courante en x par le ratio de changement d'échelle en largeur et la position courante en y par le ratio de changement d'échelle en hauteur.

Pour la position (x,y) on écrit le pixels *Im* en position (x*ratioX,y*ratioY).

L'algorithme est le suivant :

Algorithme basse qualité :

```
Fonction remplitBufferBQ(Image Buffer, Image imgData, BoiteEnglobante bE)
Début
        Variable ratioX = imgData.width() / (bE.right-bE.left);
        Variable ratioY = imgData.height() / (bE.bottom-bE.top);

        Pour y= bE.top à bE.bottom faire
        Début
                Pour x= bE.left à bE.right faire
                Début
                        Buffer(x , y) = imgData(x * ratioX , y * ratioY) ;
                        x = x + 1 ;
                Fin

                y = y + 1 ;
        Fin
Fin
```

Dans cette algorithme, bE correspond à la boite englobante de l'image data (imgData), correspondant aux valeurs où l'image Data doit être inscrite dans le buffer résultat. Si dans notre exemple, l'image avait subi une translation de 52 suivant x et 97 suivant y, en plus du changement d'échelle :

bE.top = 97 ;
bE.bottom = 197 ; (97 + largeur que l'on écrit ici 100)

bE.left = 52 ;
bE.right = 152 ; (52 + largeur que l'on écrit ici 100)

Pour pouvoir renseigner toutes ces valeurs, j'ai dans ma classe une matrice de transformation qui permet de passer des coordonnées « images » aux coordonnées « buffer ».

La boite englobante d'une image non modifié est :

Left = 0 ;
Right = img.width() ;
Top = 0 ;
Bottom = img.height() ;

Pour obtenir les quatre coins modifiés de l'image il suffit de les multiplier par la matrices de transformation. Les coordonnées de cette boite englobante sont utiles et nécessaires pour le calcul du rendu ainsi que par la suite pour les optimisations.

Le ratio de changements d'échelles sont eux aussi calculés en fonction de la zone où l'on écrit et de la taille originale de l'image, ici on écrit dans une zone de 100 pixels une largeur originel de 500 pixels, ratioX = 500/100 = 5, donc lorsque l'on se déplace de 1 dans le buffer résultat on se déplace de 5 dans l'image Data ;

« Buffer(x,y) = imgData(x * ratioX , y * ratioY) » veut dire que l'on va lire la couleur du pixel à la coordonnée (x * ratioX , y * ratioY) dans l'image Data et que cette valeur on l'écrit à la position (x , y) dans le buffer résultat. N'oublions pas qu'il faut réaliser le calcul de l'alpha expliqué au dessus (Fig. 4 page 13).

Au final, nous obtenons bien dans le buffer résultat, les différentes images réduites et/ou translatées. Vf Fig. 5.

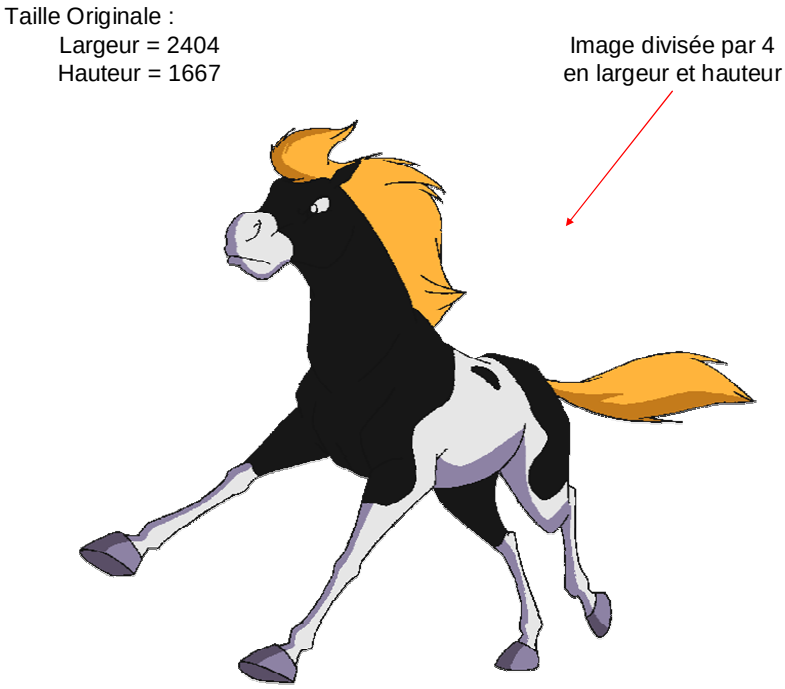

Fig. 5 : Rendu de faible qualité sans rotation.

Sur cette image, on se rend compte que l'algorithme fonctionne correctement. Mais le resulat qualitatif n'est pas au rendez-vous. En zoomant sur la queue du cheval, on peut voir que les bords sont francs. Fig 6.

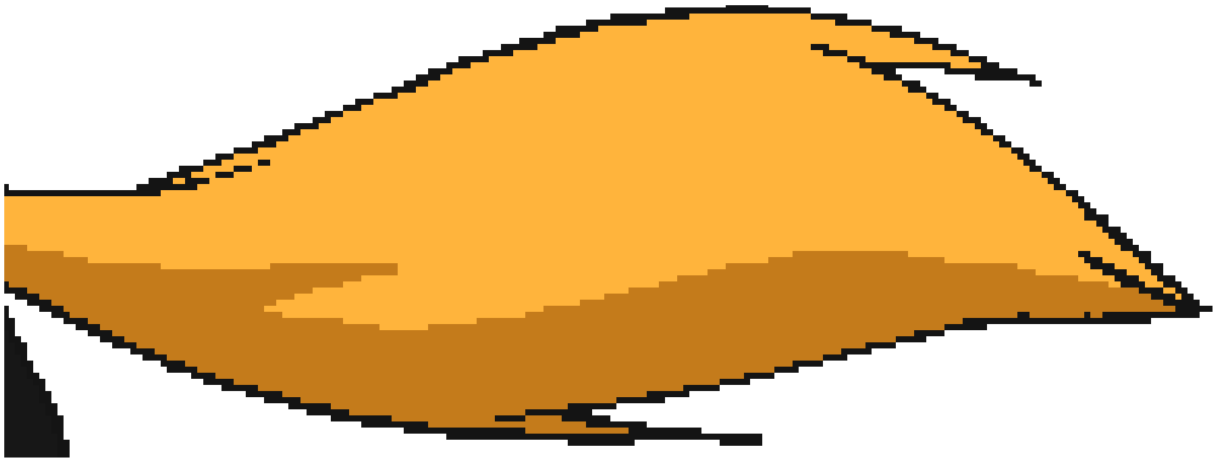

Fig. 6 : Zoom queue cheval de faible qualité.

Pour voir l'image originale du cheval, voir Annexe B.

4.2.2. Rendu de haute qualité.

4.2.2.1. Présentation.

Le rendu de haute qualité, permet d'obtenir un meilleur résultat du rendu de l'image que dans le cas précédent. En effet, dans la méthode de rendu précédente, si l'image est réduite de 5, pour calculer, on ne regarde qu'1 pixel sur 5 dans l'image Data et donc nous perdions de l'information (Vf démonstration Fig 7).

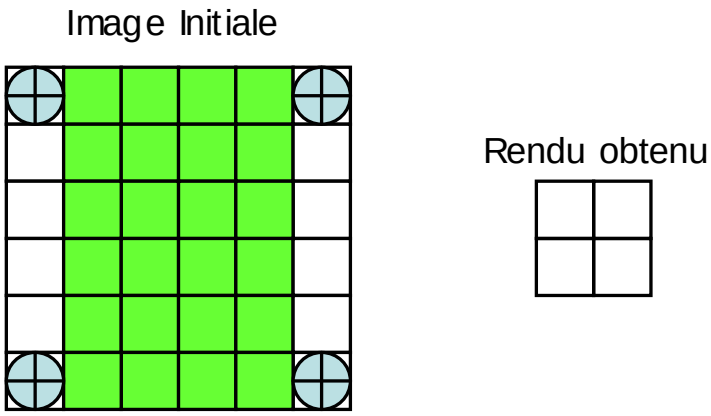

Fig 7 : Exemple de perte d'informations avec rendu de basse qualité.

Dans cette figure, on peut remarquer, que le rendu de basse qualité fait perdre de l'information, car ici on perd toute l'information de vert. Il ne faut pas croire que l'on obtient une image complètement fausse, on obtient bien l'image de départ, réduite, mais de qualité moindre.

4.2.2.2.Explications.

Le principe de fonctionnement de ce rendu est simple à comprendre mais pas évident à faire marcher de façon optimisée (le plus rapidement possible). Lors de déplacements, aucun scintillement (ou presque car suivant les images il peut exister des scintillements où d'autres traitement post-rendu ou pré-rendu sont nécessaires comme l'anti-flicking) algorithme permettant de réduire les effets d'aliasing, de scintillements en vidéo) par exemple.

Pour éviter de perdre de l'information, au lieu de prendre 1 pixel sur 5, nous allons prendre en compte tous les pixels contenus dans la zone et faire une moyenne pondérée de la valeur de chaque pixel. L'algorithme de parcours de l'image est le même que l'algorithme de basse qualité (Page 15) avec quelques lignes en plus.

Voici l'algorithme qui remplace du calcul de rendu haute qualité :

```
Fonction remplitBufferHQ(Image Buffer, Image imgData, BoiteEnglobante bE)
Début
        Variable ratioX = imgData.width() / (bE.right-bE.left);
        Variable ratioY = imgData.height() / (bE.bottom-bE.top);
        Variable indexX = 0, indexY = 0 ;
        Variable basPixelYdata = 0, hautPixelYdata = 0 ;

        Pour y= bE.top à bE.bottom faire
        Début
                basPixelYdata = indexY * ratioY ; // on récupère la 1ere valeur de
parcours en Y de l'image data
                hautPixelYdata = basPixelYdata + ratioY ;  // on récupère la dernière
valeur de parcours en Y de l'image data

        indexX = 0 ;

                Pour x= bE.left à bE.right faire
                Début
                        gauchePixelXdata = indexX * ratioX ; // on récupère la valeur en
X dans l'image data
                        droitePixelXdata = gauchePixelXdata + ratioX ; // on récupère la
valeur en X dans l'image data

                        Variable compteurSommeRouge = 0 ;
                        Variable compteurSommeVert = 0 ;
                        Variable compteurSommeBleu = 0 ;
                        Variable sommeCoefficients = 0 ;

                        Pour yData = basPixelYdata à hautPixelYdata faire
                        Début
                                Pour xData = gauchePixelXdata à droitePixelXdata faire
                                Début
                                        Couleur pixelsCourant = imgData(xData , yData) ;
                                        compteurSommeRouge +=  pixelsCourant.rouge ;
                                        compteurSommeVert +=  pixelsCourant.vert ;
                                        compteurSommeBleu +=  pixelsCourant.bleu ;
                                        sommeCoefficients += 1 ;

                                        xData += 1 ;
                                Fin
                                yData += 1 ;
                        Fin

                        Buffer(x,y).rouge = compteurSommeRouge /sommeCoefficients ;
                        Buffer(x,y).vert = compteurSommeVert /sommeCoefficients ;
                        Buffer(x,y).bleu= compteurSommeBleu /sommeCoefficients ;

                indexX = indexX + 1 ;
                Fin

                indexY = indexY + 1 ;
                y = y + 1 ;
        Fin
Fin
```

Cette algorithme, n'est pas encore l'algorithme final et permet juste de comprendre comment fonctionne le rendu. Avec les schémas qui suivent, je vais vous expliquer tel ou tel point de l'algorithme.

Exemple :

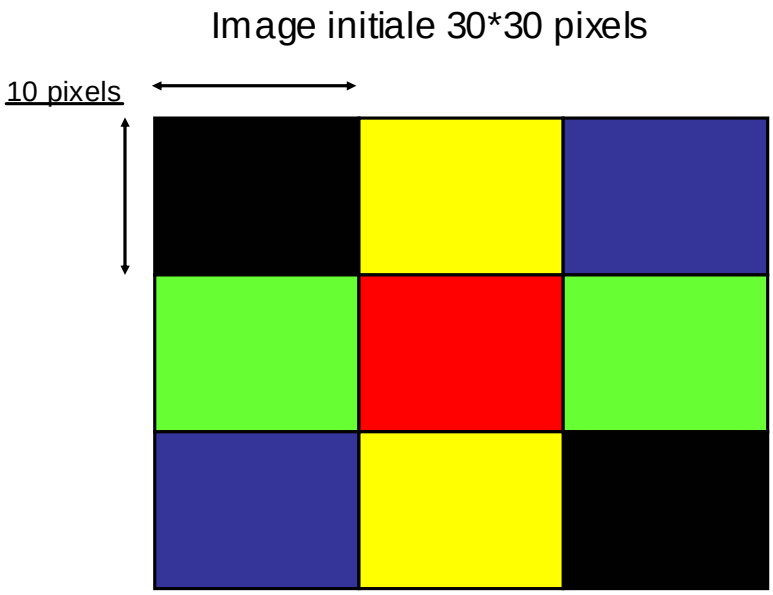

Fig. 8 : Image initial de 30*30 pixels.

Cette image servira d'image data, c'est l'image dont je cherche à calculer le rendu de haute qualité. On réduit cette image de 5 en largeur et en hauteur. C'est-à-dire que l'image de 30*30 pixels va, après le rendu être d'une taille de 6*6 pixels.

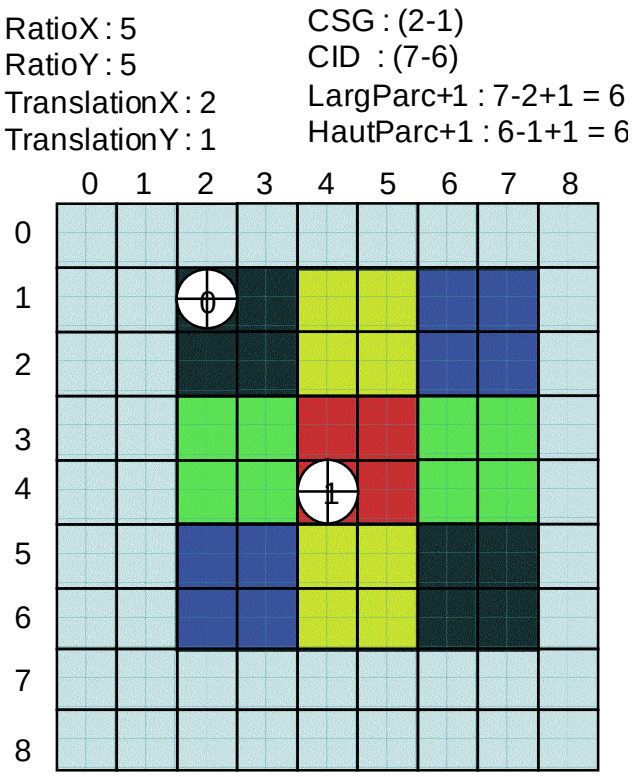

Fig. 9 : Exemple Image translater d'une valeur entière.

Sur cette figure, nous avons effectuer une translation de 2 en X et de 1 en Y. Je vais dérouler l'algorithme pour les cibles 0 et 1 pour mieux le comprendre.

Cible 0 :

basPixelYData = 0 * 5 = 0.
hautPixelYData = 0 + 5 = 5 ;
gauchePixelXData = 0 * 5 = 0 ;
droitPixelXData = 0 + 5 = 5;

La boucle de parcours de l'image data va donc faire une boucle de 5 en Y et 5 en X (ce qui est logique car maintenant, le carré noir est représenté sur 2 pixels de largeurs et 2 de hauteur alors qu'il était sur une surface de 10*10 avant réduction). Ce qui veut bien dire que maintenant un pixel est la moyenne des 5*5 pixels.

Le pixel dans le buffer aux coordonnées (2-1) aura la valeur 0 (noir) ➔ car on fait la moyenne de pixels noirs donc la valeur finale vaut 0 soit noir.

Cible 1 :

basPixelYData = 3 * 5 = 15.
hautPixelYData = 15 + 5 = 20 ;
gauchePixelXData = 2 * 5 = 10 ;
droitPixelXData = 10 + 5 = 15;

Pour la cible 1 on va donc parcourir la zone dans l'image data de 15 à 20 en Y et de 10 à 15 en X.

Comme pour la cible 0 on obtient le même rouge car on parcourt une zone unicolore rouge donc on obtient du rouge.

Dans cet exemple on obtient exactement la même image que celle de départ. La translation et le changement d'échelle on été choisis pour tomber exactement sur des valeurs entières, pour une meilleure compréhension.

RatioX : 5 CSG : (1-1)
RatioY : 5 CID : (7-7)
TranslationX : 1,4 LargParc+1 : 7-1+1 = 7
TranslationY : 1,2 HautParc+1 : 7-1+1 = 7

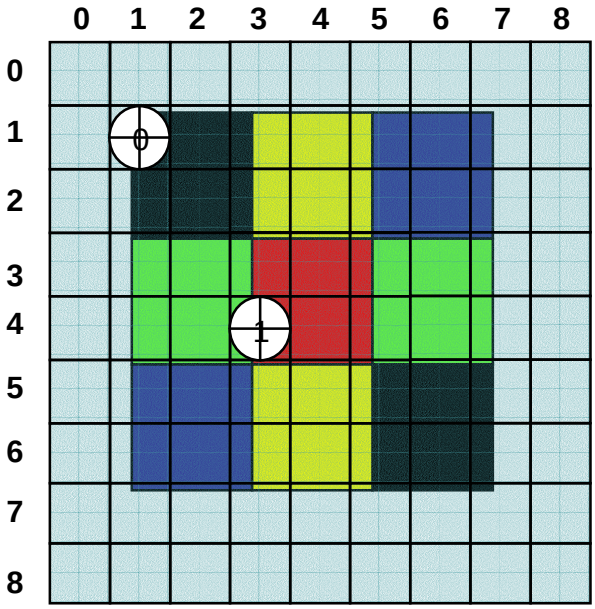

Fig. 10 : Exemple Image translater d'une valeur non entière.

Le schéma suivant est déjà un peu plus compliqué, en effet, nous avons décalé l'image de 1,4 en X et de 1,2 en Y. Ce schéma va mettre le doigt sur certains problèmes et sur les éléments à rajouter pour un fonctionnement optimal.

Cible 0 :

basPixelYData = 0 * 5 = 0 ;
hautPixelYData = 0 + 5 = 5 ;
gauchePixelXData = 0 * 5 = 0 ;
droitPixelXData = 0 + 5 = 5;

On met déjà le point sur un problème de l'algorithme. Ici, on obtient les mêmes valeurs que tout à l'heure alors que la translation est différente et qu'on peut voir que la couleur finale ne devra pas être le noir mais plutôt un mélange entre le noir et la couleur initiale du pixel (1-1).

Pour faciliter les choses, dans un premier temps, on va considérer que lors des parcours de boucle on tombe sur des pixels entiers (voir la différence avec pixel non entier au 4.2.2.3. page 24).

Pour résoudre ce problème, il va falloir prendre en compte le décalage de la translation par rapport aux valeurs entières. Pour cela on va modifier les valeurs d'initialisation des index.

indexX = – (ValeurSuperieur(translationX) – translationX) ;
indexY = – tranlationY + ValeurInferieur(translationY);

ValeurSuperieur est une fonction qui retourne la valeur entière directement supérieur au nombre passé en paramètre.
10,1 ➔ ValeurSuperieur(10,1) = 11 ;
12,99 ➔ ValeurSuperieur(12,99) = 13 ;

ValeurInferieur est une fonction qui retourne la valeur entière directement inférieur au nombre passé en paramètre.
10,1 ➔ ValeurInferieur (10,1) = 10 ;
12,99 ➔ ValeurInferieur(12,99) = 12 ;

Reprenons notre exemple, cible 0, voir si cela fonctionne :

indexX = – (2 – 1,4) = – 0,6 ;
indexY = – 1,2 + 1 = – 0,2 ;

gauchePixelXData = -0,2 * 5 = -1 ;
droitPixelXData = -1 + 5 = 4;
basPixelYData = -0,6 * 5 = -3 ;
hautPixelYData = -3 + 5 = 2 ;

On obtient des valeurs négatives pour le parcours d'image data, cela indique que l'on est en dehors de l'image et donc qu'au lieu de prendre la valeur RVB du pixel dans data on prend la valeur RVB du pixel du buffer (soit la couleur de fond du pixel courant). Si l'on obtenait des valeurs plus grandes que l'image data, cela signifierait que l'on est plus dedans et qu'il faut prendre en compte la valeur du fond.

Donc pour la cible 0, la couleur dans le buffer à la position (1-1) va être un mélange entre le noir et le cyan.

Cible 1 :

gauchePixelXData = 1,4 * 5 = 7;
droitPixelXData = 7 + 5 = 12;
basPixelYData = 2,8 * 5 = 14;
hautPixelYData = 14 + 5 = 19;

Ici, le calcul de moyenne va donc s'effectuer sur le vert et le rouge et donc le résultat sera le mélange des deux.

Je vous ai détaillé ici le principe général du calcul du rendu de meilleure qualité, on peut vite comprendre que ce calcul sera plus lent que celui de basse qualité, du fait qu'il fait beaucoup plus de calculs. L'algorithme ci-dessus n'est pas complet. En effet, plusieurs problèmes sont apparus dont la gestion des fragments de pixels que je vais vous expliquer dans une partie séparée.

4.2.2.3. Fragments de pixels.

4.2.2.3.1. Qu'est-ce qu'un fragment de pixel ?

Comme son nom l'indique, un fragment de pixel est un morceau de pixel, cette idée n'est pas évidente à admettre car un pixel est la plus petite quantité représentable sur un ordinateur. En effet un pixels est un point de l'image, plus exactement un carré.

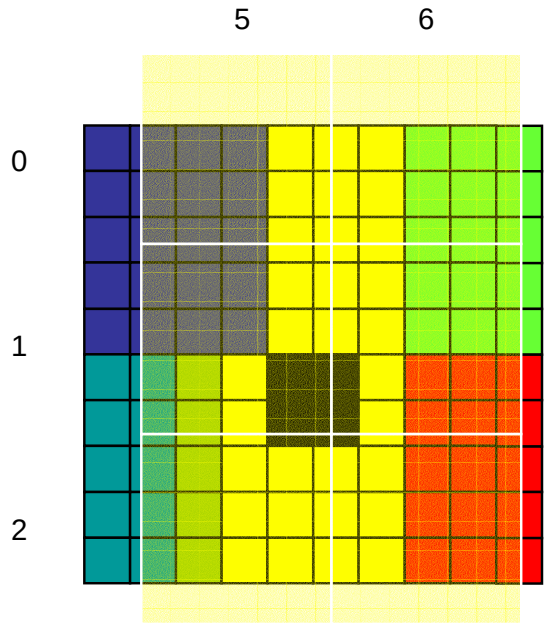

Fig. 11 : Exemples fragments de pixels

On cherche ici a calculer le rendu de l'image de départ qui fait 10*10 pixels. Dans le pixel (5-1) du buffer, on cherche à calculer la moyenne des pixels qui vont être inscrits dedans. On voit que certains pixels de l'image de départ sont « découpés » et c'est cela que l'on appelle fragment de pixel. Il n'est pas logique qu'un pixel soit entièrement pris en compte dans le calcul de moyenne alors qu'il n'est pas entièrement intégré dans le pixel buffer calculé. Donc on va prendre comme coefficient la surface occupée par le pixel.

4.2.2.3.2. Calcul correct de la couleur du pixel final.

Le schéma suivant est un zoom effectué sur le pixel (5-1) du buffer.

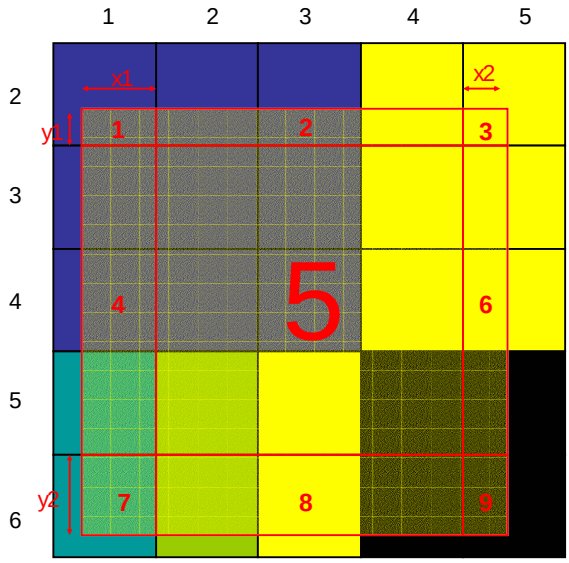

Fig. 12 : Zoom pixel (5-1) de la figure 11.

Pour pouvoir effectuer correctement le calcul de moyenne, il va falloir découper la zone de calcul en 9 parties comme montrée sur la figure précédente.

Calcul de moyenne pondérée : $$moyenne = \frac{c1*v1 + c2*v2 + ... + cX*vX}{c1 + c3 + ... + cX}$$

Dans chacune des 9 zones, il n'y a que les coefficients qui changent, en effet, dans le calcul de moyenne pondérée, les valeurs v1,…, vX correspondent aux valeurs du pixel courant dans l'image data. Ensuite, pour chaque zone, c'est le coefficients c1,…, cX qui va changer suivant la surface occupée.

Pour les zones 1, 3, 7, 9, le coefficients multiplicateur, est la surface occuper dans le pixel :

- Zone 1 : coefficient = x1 * y1 ;
- Zone 3 : coefficient = x2 * y1 ;
- Zone 7 : coefficient = x1 * y2 ;
- Zone 9 : coefficient = x2 * y2 ;

Pour les 2 et 8, il faut effectuer une boucle sur la largeur, en prenant comme coefficient :

- Zone 2 : coefficient = 1 * y1 ;
- Zone 8 : coefficient = 1 * y2 ;

Pour les 4 et 6, il faut effectuer une boucle sur la hauteur, en prenant comme coefficient :

- Zone 4 : coefficient = x1 * 1 ;
- Zone 6 : coefficient = x2 * 1 ;

Enfin la zone 5 correspond à la zone où tous les pixels sont intégrés entièrement, donc le coefficient vaut 1. Cette zone correspond à l'accolade de l'algorithme de haute qualité expliqué plus haut.

Cette explication correspond au traitement final réalisé lors d'un calcul de rendu de haute qualité.

Voici l'image de Petit Tonnerre de haute qualité :

Fig. 13 : Rendu de haute qualité.

En zoomant sur la queue, comme dans l'exemple de basse qualité, on peut voir que les bords sont moins francs, dû aux calculs de moyenne.

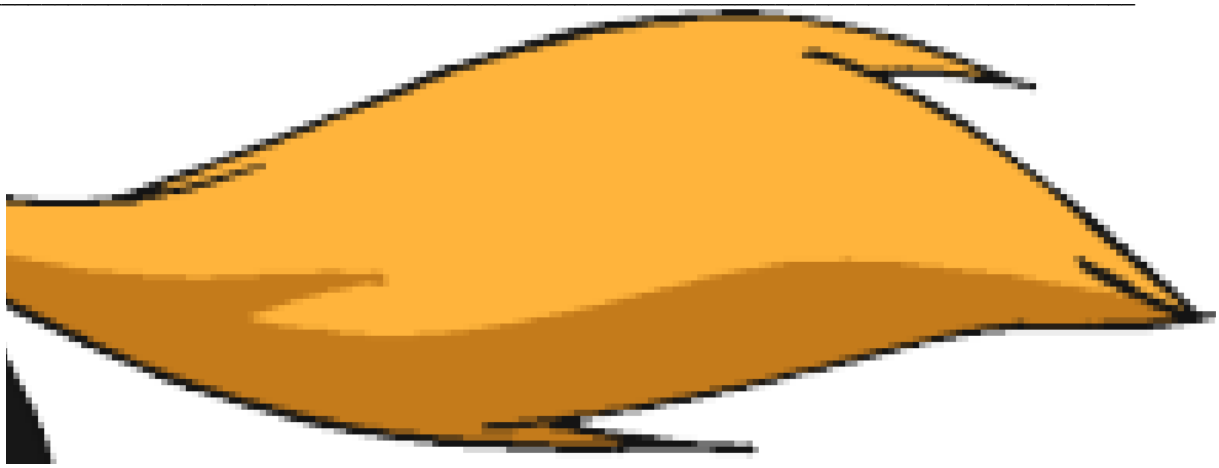

Fig. 14 : Zoom queue cheval de haute qualité.

4.3. Calcul de rendu d'images avec rotation.

4.3.1. Parcours de l'image data.

La première idée qui nous vient à l'esprit (qui est rarement la meilleure) est de parcourir tous les pixels de la boite englobante.

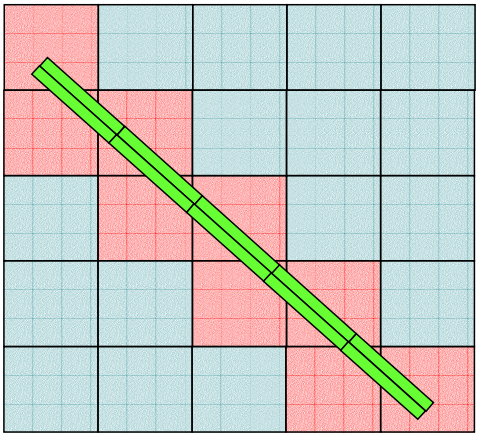

Fig. 15 : Image expliquant le parcours avec rotation.

Le cadre bleu et rouge représente la boite englobante de l'image dans mon buffer résultat. On remarque que seul les pixels de ce buffer marqués de rouge seront modifiés si l'on cherche à rendre l'image verte. On se rend compte que si l'on parcourt tous les pixels de la boite englobante, on parcourt des pixels inutiles (pixels bleus) et donc on perd du temps.

L'idée, est de ne parcourir que les pixels utiles, qui seront modifiés lors du rendu (pixels rouge). L'idée de parcours est la suivante :

```
Pour toutes les lignes de la boite englobante faire
Début
    Calculs d'intersections (ligneCourante, xMin, xMax) ;

    Pour x= xMin à xMax faire
    Début
            Calcul de rendu avec rotation ;
    Fin
Fin
```

La fonction qui calcule les intersections, affecte à xMin, le point d'intersection minimum et à xMax le point d'intersection maximum de la boite englobante et de la ligne courante.

Calcul d'intersection :

Pour savoir à quel endroit on a une intersection entre la ligne courante et un des bords de l'image, on effectue un calcul d'intersection mathématique. S'il y a intersection, on stocke les coordonnées minimales et maximales qui correspondront au coté gauche et droit.

Après il reste « juste » à calculer la valeur du pixel courant.

4.3.2. Calcul du rendu de basse qualité.

La méthode la plus simple est : « Pour chaque pixel de la zone, regarder la valeur du pixel dans l'image data (en multipliant la position courante par la matrice de projection) et l'écrire dans le buffer ».

Effectivement, cette méthode fonctionne et donne un résultat correct mais l'optimisation n'est pas au rendez-vous. Une multiplication de la position courante par la matrice de projection fait intervenir 12 multiplications et 12 additions de nombre à virgule.

Il faut donc réduire le nombre de calculs et effectuer le moins de multiplications de matrice possible.

La solution que j'ai retenue : à partir des coordonnées de parcours, trouver les 2 pixels « extrêmes » pour la ligne donnée dans l'image data d'origine (sans transformation) et calculer le vecteur entre ces deux positions, puis se balader sur ce vecteur.(Fig. 16).

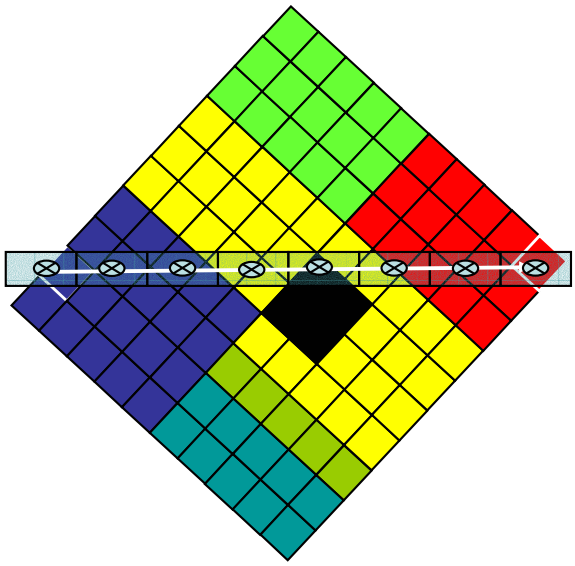

Fig. 16 : Rendu basse qualité avec rotation.

Cette figure nous aide à comprendre, que lorsque nous sommes sur la ligne où l'on peut voir les pixels du buffer, on se décale le long du vecteur blanc, puis, à chaque position intermédiaire, on regarde la valeur du pixel dans l'image. Cette technique est beaucoup plus rapide que celle avec toutes les multiplications de matrice et de qualité égale.

4.3.3. Calcul du rendu de haute qualité.

Le rendu de haute qualité, à été le « plus simple » à réaliser du moment que les 3 autres rendus fonctionnent. En effet en « trichant » un petit peu, on obtient un rendu de haute qualité sans scintillement, sans problèmes visuels…

Nous effectuons le même parcours que celui expliqué au titre 4.3.1., et le même algorithme de rendu haute qualité sans rotation que celui expliqué au 4.2.2.

Pour que l'algorithme de haute qualité sans rotation fonctionne correctement, il suffit de modifier le calcul des quatre coins. On peut remarquer sur la figure 17 que pour le pixel du buffer, la technique consiste à calculer le gros carré cyan dans le sens de l'image d'origine (et non dans le sens de la rotation comme cela devrait être).

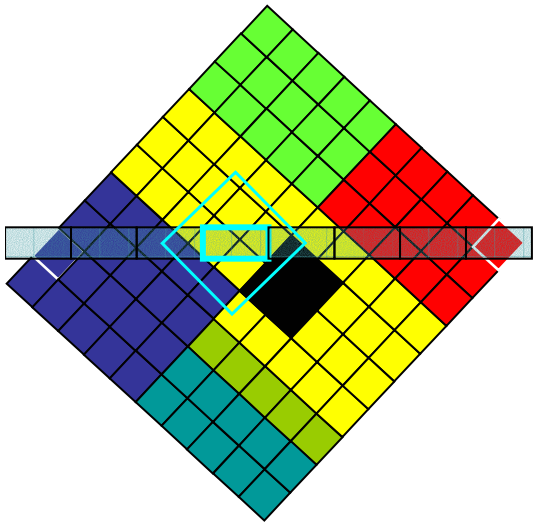

Fig. 17 : Explication rendu haute qualité avec rotation.

Cette technique donne des résultats corrects, avec des temps de calculs corrects. Si nous avions voulu effectuer le calcul de rendu de haute qualité avec une boite dans le sens de la rotation, cela aurait été monstrueux en temps de calcul (ce qui n'est pas recherché) pour un résultat à peine meilleur.

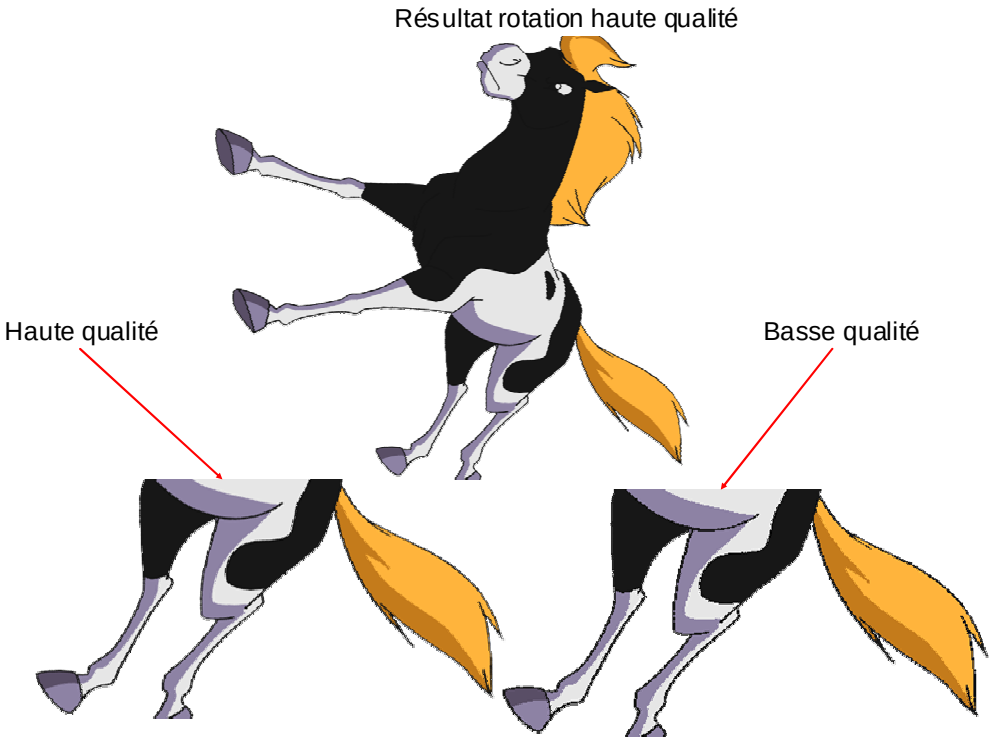

Fig. 18 : Résultat final, rendu avec rotation.

Sur cette figure, on voit le résultat du rendu avec rotation, ainsi que le zoom sur le train arrière du cheval où l'on peut remarquer une différence visuelle.

4.4. Différents problèmes rencontrés.

Durant cette partie du travail, à laquelle j'ai consacré beaucoup de temps, de nombreux problèmes ont été surmontés. En effet, je ne vous ai pas détaillé les différentes étapes par lesquelles je suis passé, mais ce ne fut pas facile.

Ce sont les bords qui m'ont posé le plus de problèmes : soit il manquait une ligne ou une colonne, soit j'avais des scintillements sur les bords ou encore une mauvaise gestion de la couleur des bords… Ce qui n'est pas très évident à résoudre. La plupart de ces problèmes résidaient dans le fait qu'il fallait tout le temps penser à inverser l'image car les coordonnées du logiciel et les coordonnées images étaient inversées. De plus lors des parcours, il fallait faire des bonds invariants de boucles…C'est souvent une source d'erreur si l'on ne fait pas très attention.

De plus, trouver où se situaient les problèmes n'était pas aisé. Lorsque l'on travaille avec des images il y a souvent plusieurs centaines de pixels, ce n'est donc pas facile à analyser et trouver les problèmes. La plupart du temps, j'utilisais une image de 3*3 pixels, voire 10*10 pixels pour essayer de trouver une solution, cela permet de limiter le nombre de valeurs.

L'alpha, n'est pas non plus évident à utiliser. La transparence cause de nombreux problèmes (assez vite résolus malgré tout).

La gestion des fragment de pixels m'a également pris beaucoup de temps, surtout pour être sur que celui-ci fonctionne correctement et soit optimisé c'est-à-dire évité de faire des calculs inutiles.

5. Gestions des masques et des filtres.

J'expliquerai en quoi consiste cette partie, sans rentrer dans le détail technique. En effet, une présentation globale du logiciel sera effectuée lors de la soutenance orale, et les différentes fonctionnalités des masques et filtres y seront présentés. Cette partie des travaux, a demandé beaucoup de temps pour une fonctionnalité optimale et efficace.

En première partie je vous expliquerai leur fonctionnement et leur utilité et en deuxième partie, j'énumérerai les différents filtres que j'ai réalisés.

5.1. Gestion des masques.

Un masque est une image, qui permet de cacher plus ou moins une image. Pour savoir si une image est visible à travers le masque, il suffit de regarder la valeur de l'alpha du masque. Figure 19.

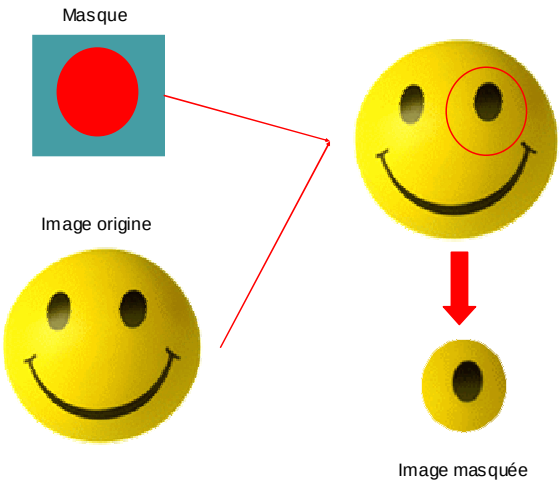

Fig. 19 : Gestion des masques.

Sur la figure 19, le masque à une valeur d'alpha égale à 255 dans la zone rouge. Lorsque que nous déplaçons le masque sur une image, ne sont visibles que les zones où il y a de l'alpha. Suivant la valeur de l'alpha, la zone est plus ou moins visible.

Le but des masques est de cacher certaines zones d'images non désirées, mais l'on veut pouvoir masquer des masques. C'est-à-dire voir l'image finale à travers plusieurs masques. J'ai donc codé différentes opérations possibles entres les masques comme vous le pouvez le voir sur la figure 20.

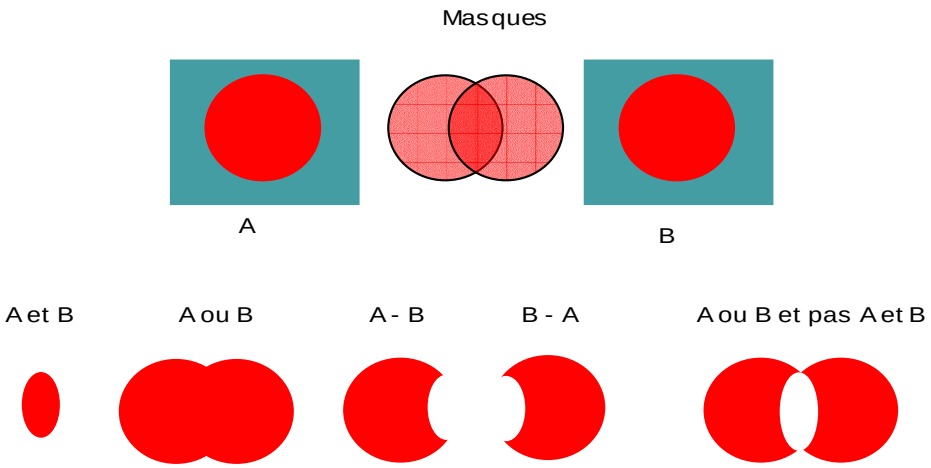

Fig. 20 : Opérations sur les masques.

Il y a 5 opérations possibles sur les masques :

- ET : on voit tous ce qui est à l'intersection des masques.
- OU : on voit tous ce qui passe a travers l'alpha des masques.
- Différence : on voit ce qui passe à travers un masque en enlevant ce qui passe à travers l'autre masque
- Différence Inverse : comme la Différence mais en inversant les deux masques
- OU Exclusif : On voit se qu'il y a au travers des masques sauf leur intersection.

5.2. Gestion des filtres.

Un filtre permet de modifier l'image originale par un effet voulu. J'ai donc créé différents filtres permettant de modifier l'image. Ceux ci peuvent être appliqués à travers des masques ou sur toute l'image.

Tous les filtres réalisés, n'ont pas été réalisés dans le but de m'occuper, ce sont des filtres qui sont utilisés pour les différentes productions de dessins animés.

5.2.1. Filtre de couleurs.

Ce filtre, permet de multiplier (diviser, additionner, soustraire) la couleur du pixel original avec la couleur choisie.

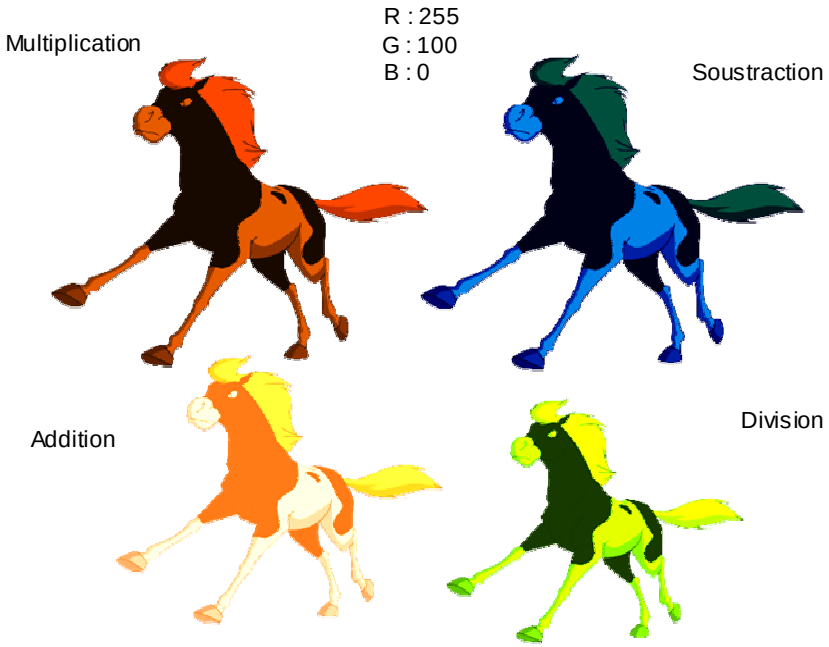

Fig. 21 : Filtre de couleurs.

On voit, sur cette image, le cheval et la couleur (255, 100, 0) appliquée avec les différentes opérations mathématiques correspondantes.

5.2.2. Filtre niveau de gris.

Ce filtre est un filtre assez simple : il converti l'image en niveau de gris. La véritable formule de conversion en niveau de gris est :

$$niveauDeGris = 0.30 * Rouge + 0.59 * Vert + 0.11 * Bleu;$$

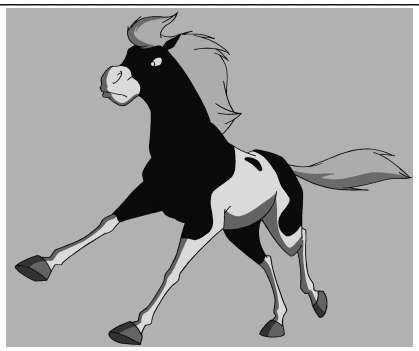

Fig. 22 : Filtre Niveau de gris

5.2.3. Filtre négatif.

Ce filtre permet d'inverser les couleurs de l'image. Une couleur étant comprise entre 0 et 255, se filtre applique cette formule :

$$\begin{aligned} Rouge &= 255 - Rouge; \\ Vert &= 255 - Vert; \\ Bleu &= 255 - Bleu; \end{aligned}$$

Il y a aussi un coefficient entre 0 et 1 qui permet de mélanger la couleur d'origine avec la couleur modifiée, pour faire des fondus notamment.

Fig. 23 : Filtre négatif

5.2.4. Filtre aplat de couleur.

Le filtre Aplat de Couleur sert à remplacer toute l'image (ou seulement la zone masquée) par une couleur que l'utilisateur peut changer à tout moment.

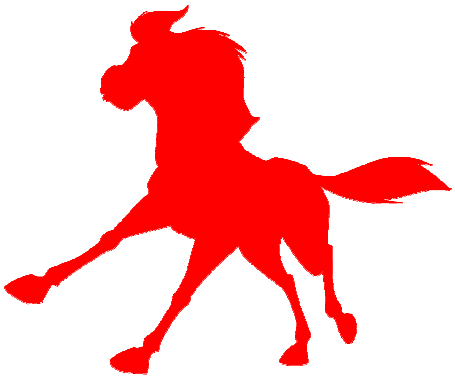

Fig. 24 : Filtre Aplat de couleur Rouge

5.2.5. Filtre HSL.

Le HSL est un espace de couleurs (comme le RGB), mais la couleur est définie autrement :

- H : Hue c'est-à-dire la Teinte en français.
- S : Saturation
- L : Luminance c'est-à-dire la Luminosité en Français.

Je ne vais pas détailler plus cette espace de couleur car cela peut prendre plusieurs pages et je ne pense pas tout connaître.

J'ai utilisé cette espace de couleurs car il est plus intuitif pour l'utilisateur que le RVB.

Fig. 25 : Filtre HSL

Sur cette figure, a été appliqué un filtre HSL où l'on a réduit le H de 101 et le S de 7.

5.2.6. Filtre Rampe.

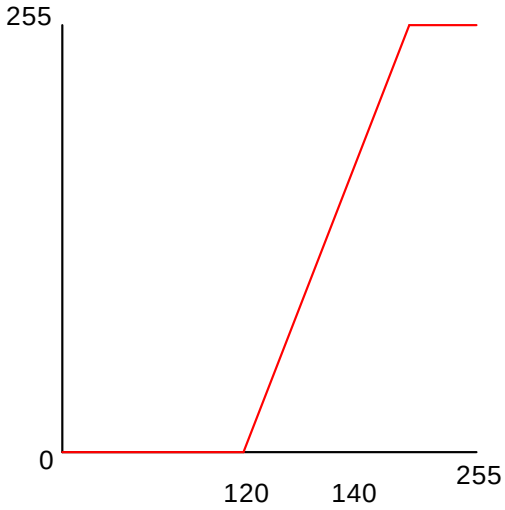

Fig. 26 : Exemples de rampe.

La figure 27 montre ce que signifie une rampe. Ce schéma explique ce que fait se filtre. L'utilisateur choisit deux valeurs : une minimal (ici 120) et une maximal (ici 140) et ce que fait le filtre est :

- pixel < min : pixel=0;
- pixel > max : pixel=255;
- min < pixel < max :

$$pixel = \frac{(pixel - \min) * 255}{\max - \min};$$

Quand la valeur du pixel courant est supérieure au minimum et inférieure au maximum, on fait une interpolation pour remettre cette valeur entre 0 et 255.

5.2.7. Filtre Luminosité/Contraste.

5.2.7.1. Filtre Luminosité.

Ce filtre, augmente « la lumière » par une opération assez simple, l'utilisateur choisit une valeur entre -255 et 255, la formule est la suivante :

$$Couleur = val + Couleur;$$

En appliquant cette formule, si val = -128, chaque composante Rouge, Vert, Bleu va être diminuée de 128 et donc assombrir l'image et inversement avec +128 ;

5.2.7.2. Filtre Contraste.

Le filtre contraste permet de rehausser le contraste, comme sur les télévisions.

5.2.7.3. Exemple.

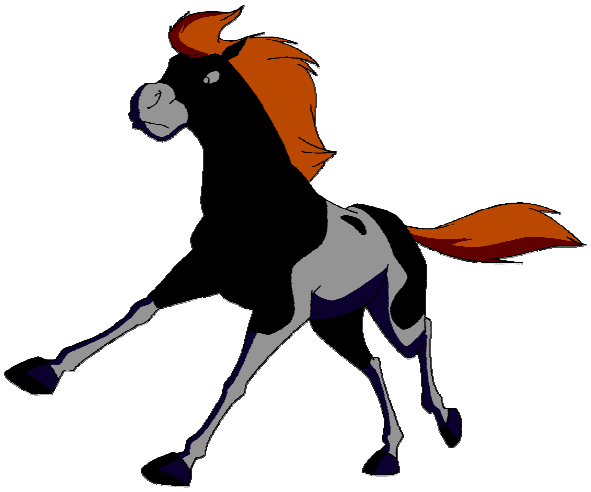

Fig. 27 : Filtre Luminosité/Contraste

A cette image, on a appliqué une luminosité de -71 et un contraste de 35.

5.2.8. Filtre Flou.

Le filtre flou, comme son nom l'indique, permet de rendre l'image floue.

Fig. 28 : Filtre flou

5.2.9. Filtre Back Light.

Le filtre Back Light, que l'on peut traduire par contre jour, permet de faire apparaître une sorte « d'aura » autour de l'objet. Ce filtre est une combinaison des filtres aplat de couleur et flou.

La réalisation de ce filtre est la suivante :

- On aplatit l'image avec une couleur voulue.
- On effectue plus ou moins de flou de force voulu (le tout paramétrable) sur l'image aplatie.
- Puis on met par-dessus l'image originale.

Voici le résultat :

Fig. 29 : Filtre Back Light.

5.2.10. Filtre Chroma Key.

Vous avez déjà du entendre parler des Chroma Key mais sans le savoir, tous les jours vous voyez le résultat de filtres similaires. Ce filtre permet de retirer une couleur d'une image. Cette technique est utilisée tous les jours par la météo. Les présentateurs météo sont devant un écran bleu en train de faire leur bla-bla alors qu'on les voit devant des cartes diverses et variées. Ils remplacent le bleu par les cartes voulues, c'est le principe du Chroma Key.

On pourrait traduire Chroma Key par Clef Chromatique. Son but est de remplacer une (ou plusieurs) couleurs soit par rien soit par une autre couleur.

Ce filtre permet de sélectionner une ou plusieurs couleurs, de rendre transparent les pixels (juste en changeant l'alpha) ou de changer leur couleur grâce au filtre HSL.

Vous pouvez voir sur la Figure 30 le résultat de ce filtre :

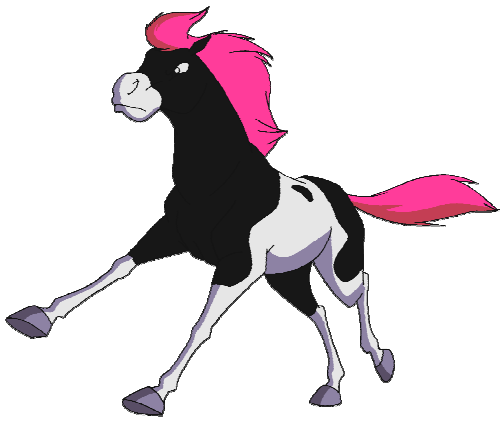

Fig. 30 : Filtre Chroma Key.

Dans cette image, j'ai sélectionné des couleurs (celles de la crête et de la queue) fixé une erreur de couleur (pour élargir plus ou moins la zone désirée), et réduit la teinte de 66.

Ce filtre n'est pas évident à réaliser, pour qu'il soit le plus correct possible et demande d'y consacrer du temps.

5.2.11. Filtre Remplacement de couleur.

Le filtre Remplacement de couleur a exactement le même principe que le filtre Chroma Key, mais au lieu de jouer sur la valeur HSL ou sur l'alpha, je remplace la zone détectée par une couleur définie par l'utilisateur.

5.3. Problèmes

La chose la moins évidente à corriger :
- Un masque peut avoir un masque,
- Un masque peut être filtré
- Un filtre peut être masqué.

Il a fallu gérer cela le mieux possible afin de pouvoir être le plus rapide possible.

Conclusion

Ce stage a été très bénéfique pour moi, il m'a permis d'apprendre :

- le processus global des films d'animation et des dessins animés en général,
- le logiciel Microsoft Visual Studio, qui permet de développer des programmes écrits en C++,
- à travailler dans une véritable équipe de développement avec tout ce que cela implique

Je suis globalement satisfait des résultats finaux qui ont parfaitement su répondre aux attentes de l'entreprise, bien que de nombreuses améliorations soient encore possibles telles que l'ajout de filtres par exemple.

L'apprentissage à l'IUT m'a permis l'acquisition de nombreuses connaissances grâce à un enseignement de qualité.

Pour ne pas laisser le logiciel et cette aventure tomber, suite à une demande d'embauche de 2minutes, mon aventure va continuer avec eux à ANGOULEME où je suis embauché en CDI.

Annexe A : Chaîne de fabrication des dessins animés.

Le dessin animée 2 dimensions, se décomposent en trois grosses étapes : la pré-production, la fabrication et la post-production.

I) La pré-production.

La pré-production, comprend les phrases de conception et de réalisation :

- L'écriture : il permet de rassembler les différents points important de la série. L'écriture se décompose en 3 étapes : le synopsis qui est la trame de la série, le séquencier qui reprend l'histoire scène par scène et le scénario qui combine les décor et le dialogue.

- Le design : il regroupe l'ensemble des éléments graphiques.

- Le storyboard : est un descriptif graphique du film, combine scénario et les éléments du design. Il consiste en une succession de vignettes illustrant le déroulement de l'action dans chaque plan. Il donne une description de l.action, les indications des différents effets éventuels (visuels ou sonores), les dialogues, les mouvements de caméra et la numérotation des plans, des séquences et des décors. Le storyboard est un travail particulièrement créatif, il apporte des indications précises de mise en scène et de rythme. C'est le document de référence pour la suite de la chaîne.

- La détection : les voix (définitives ou maquettes) sont enregistrées avant l'animation, de sorte que l'animateur puisse dessiner correctement les bouches des personnages.

- L'animatique : est un enregistrement du storyboard synchronisé sur la bande-dialogues. Le principe est l'élaboration d'une maquette visuelle permettant de vérifier notamment la correction du minutage et la pertinence des raccords.

- La feuille d'exposition : c'est un document de communication fondamental dans le processus de fabrication d'un dessin animé, particulièrement lorsque l'animation doit être sous-traitée. A partir de son émission et jusqu'à la prise de vue finale, elle synthétise tous les éléments pertinents concernant chaque plan, et les actualise au fur et à mesure de leur création.

- Le layout : s'effectue à partir des différentes feuilles de modèles, du storyboard et des données consignées dans la feuille d.exposition. Il s'agit de dessins, taille réelle, représentant chaque scène du storyboard en version animée. Cette étape permet de préparer chaque plan dans le bon cadrage en vue de l'animation et du tournage, et de séparer les différents niveaux (décors de fond, overlays et underlays, animations) préalablement à leur traitement. Le layout facilite la réalisation du décor, de l.animation et du tournage.

II) La fabrication.

La fabrication se décompose en 3 étapes :

- Le décor : correspond à l'ensemble des éléments graphiques qui ne sont pas animés. Le décorateur transfère le layout décor sur papier et le met en couleur, en respectant les règles du raccord et de la continuité.

- L'animation : qui consiste à animer tous les personnages, objets…

- Le segment post-animation : permet d'améliorer l'animation (corriger les défaut de couleurs, de formes…), de rajouter des effets spéciaux…

III) La post-production.

La post-production se décompose en 2 étapes :

- Le mixage : la « chaîne du son » pour l'animation comprend l'enregistrement des voix, de la musique, et des bruitages. Le mixage combine les différentes bandes-son pour obtenir la bande-son définitive.

- Le montage : est l'assemblage des plans et des sons.

Annexe B : Image Petit Tonnerre.

Largeur : 2404 pixels.
Hauteur : 1667 pixels.
9 Couleurs différentes.