

Projet de traitement d'image

Présentation du sujet

Pour le projet du module de traitement d'image, j'ai choisi d'implémenter une méthode de rendu non-photoréaliste.

Aaron Hertzmann propose dans son article, « Painterly Rendering with Curved Brush Strokes of Multiples Sizes », une méthode pour créer des images ayant l'apparence de peintures à partir de photographies. Cette méthode permet de simuler les coups de pinceaux d'un peintre en fonction de différents paramètres qui seront exposés plus loin. Ces paramètres permettent de donner un style de peinture précis à une image.

I - Résumé de l'article

1 – Méthode principale

```

function paint(sourceImage, R1 ... Rn)
{
    canvas := a new constant color image

    // paint the canvas
    for each brush radius Ri,
        from largest to smallest do
    {
        // apply Gaussian blur
        referenceImage = sourceImage * G(fσ Ri)
        // paint a layer
        paintLayer(canvas, referenceImage, Ri)
    }
    return canvas
}

```

2

Cet algorithme va simuler les méthodes utilisées par les peintres pour générer une image ayant l'aspect d'une peinture. L'image sera organisée en un ensemble de couches qui seront peintes successivement. Chaque couche sera associée à une taille de pinceau constante. La couche ayant la plus grande taille de pinceau est peinte en première, puis ensuite nous affinons les détails de l'image en peignant les couches ayant des tailles de pinceau plus petites.



Chaque couches sera peinte à partir d'une version lissée de l'image originale. Nous avons gardés le choix du filtre de Gauss comme méthode de lissage de l'image. A.Hertzmann précise que la diffusion non linéaire peut être utilisé à la place du filtre de Gauss pour obtenir de meilleur résultat.

Nous utilisons ces paramètres pour le lissage de l'image avec le filtre de Gauss :

Déviation standard : $\sigma = f_\sigma \cdot R_i$, avec R_i le rayon du pinceau et f_σ un facteur constant

Taille du noyau : $(6 \cdot \sigma + 1) \times (6 \cdot \sigma + 1)$

Une couche est formée par un ensemble de coups de pinceaux et chaque coups de pinceaux est représentés par un ensemble de points de contrôles formant une B-Spline. Le calcul des points de contrôles se fait à partir du gradient de la luminance de l'image lissée.

La luminance d'un pixel est calculée avec la fonction suivante :

$$L(r, g, b) = 0.30 * r + 0.59 * g + 0.11 * b$$

Les orientations x et y du gradient sont calculées en effectuant la convolution de l'image I avec les noyaux de Sobel :

$$G_x = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 * A \end{matrix} \text{ et } G_y = \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 * A \\ 1 & 2 & 1 \end{matrix}$$

On approxime l'amplitude G du gradient en calculant :

$$G = 0.5 * |G_x| + 0.5 * |G_y|$$

2 – Sélection des coups de pinceau

```

procedure paintLayer(canvas, referenceImage, R)
{
    S := a new set of strokes, initially empty
    // create a pointwise difference image
    D := difference(canvas, referenceImage)

    grid := fg R

    for x=0 to imageWidth stepsize grid do
        for y=0 to imageHeight stepsize grid do
        {
            // sum the error near (x,y)
            M := the region (x-grid/2..x+grid/2,
                            y-grid/2..y+grid/2)
            areaError := Σi,j ∈ M Di,j / grid2
            if (areaError > T) then
            {
                // find the largest error point
                (x1,y1) := arg maxi,j ∈ M Di,j
                s := makeStroke(R,x1,y1,referenceImage)
                add s to S
            }
            paint all strokes in S on the canvas,
            in random order
        }
}

```

Une couche est donc un ensemble de coups de pinceaux. Nous devons parcourir l'image pour sélectionner les coordonnées de départ de chaque coups de pinceau. Une fois les points de contrôles des coups de pinceau calculés, nous pouvons peindre chaque coups dans un ordre aléatoire.

Nous avons besoin d'abord de calculer la différence entre l'image originale lissée et l'image peinte. Initialement, l'image peinte est remplis par une par une couleur constante choisie de manière à ce que l'image entière soit traité par l'algorithme.

La différence des deux images est calculée de la façon suivante :

$$|(r_1, g_1, b_1) - (r_2, g_2, b_2)| = ((r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2)$$

On note D la différence entre ces deux images.

On parcourt ensuite l'image en largeur et hauteur par pas de grid pixels.

*Avec grid = f_g * R, R le rayon des coups de pinceau et f_g une constante.*

On délimite l'image en un ensemble de région M contenant grid × grid pixels.

On calcule ensuite l'erreur moyenne dans chaque zone M :

$$\text{areaError} = \sum_{i,j \in M} D_{i,j} / \text{grid}^2$$

*Si l'erreur moyenne dépasse un seuil T fixé par l'utilisateur,
alors on calcule un nouveau coup de pinceau.*

*Les coordonnées de départ de ce coup de pinceau sont :
(x₀, y₀) = max_{i,j ∈ M} (D_{i,j})*



3 – Calcul des coups de pinceau

Un coup de pinceau est initialisé avec un premier point de contrôle (x,y) , un rayon R constant et une couleur C constante. Cette couleur C est la couleur de l'image originale lissée aux coordonnées (x,y) .

Un coup de pinceau est une courbe B-Spline représentée par un ensemble de points de contrôle.

Le calcul des points de contrôle s'effectue comme les calculs des termes d'une suite récurrente.

Un coup de pinceau commence par le point P_0 de coordonnées (x_0, y_0) .

Le point suivant P_1 est calculé à partir de la direction du gradient au point P_0 .

Les coordonnées de P_1 seront placés sur la normale du gradient P_0 de manière à ce que la courbure s'

Cette courbure peut être accentuée ou minimisée avec un filtre RII (Réponse Impulsionnelle Infinie)

Les coordonnées de P_1 sont également placé à une distance R (en pixels) de P_2 .

On effectue ces calculs jusqu'à atteindre le nombre maximal de points de contrôle définis par l'utilisateur et également si on atteint une autre condition d'arrêt.

5

Le calcul s'arrête prématurément si l'amplitude du gradient est nulle en un point (x,y) .



II – Implémentation et résultats

1 – Choix d'implémentation

L'implémentation a été réalisé en C++ avec la bibliothèque OpenCV. OpenCV nous fournit les fonctions essentielles aux chargements et l'enregistrement d'image, aux calculs de gradient et au lissage d'image utilisées par les différents algorithmes présentés précédemment.

Les algorithmes présentés par A.Hertzmann sont suffisamment génériques pour nous laisser choisir des structures de données adaptés.

Les rayons des pinceaux simulés par l'algorithme de rendu sont calculés à partir de 3 constantes : le rayon maximum du pinceau, un facteur qui permettra de calculer les rayons suivants à partir du rayon maximum et le nombre de rayon utilisé pour peindre l'image(c'est à dire le nombre de couches peintes successivement).

Concernant le calcul de la différence entre l'image de référence lissée et l'image peinte, la différence calculée pour la première couche est fixée à la constante $2*T$, de manière à ce que l'erreur moyenne de chaque région M soit toujours supérieure au taux d'approximation T. Ainsi l'image résultante sera peinte sur toute sa surface. La constante T est comprise entre 0 et 255 et correspondant à l'approximation à atteindre par rapport à l'image originale. Plus T est élevé, plus l'approximation sera grossière, et plus T est faible, plus l'image résultante sera fidèle à l'image d'origine.

Nous avons choisi de peindre chaque coups de pinceau juste après l'avoir calculé, ce qui évite des stockages inutiles. Nous utilisons un Z-Buffer pour peindre les coups de pinceau de manière aléatoire mais aussi pour réduire le nombre de coups de pinceau à calculer. Le Z-Buffer est une image en niveau de gris initialisée à 0 sur tous ses pixels. Nous attribuons à chaque coups de pinceau une valeur z aléatoire comprise entre 1 et 255. Si cette valeur z est supérieure à la valeur présente dans le Z-Buffer nous pouvons peindre le coups de pinceau, sinon ce coups de pinceau n'a pas besoin d'être calculé puisqu'il ne sera pas peint. Lorsque nous peignons l'image, nous peignons aussi dans le Z-Buffer mais avec un rayon R plus petit pour éviter que certains coups de pinceaux ne soient trop espacé les uns des autres dans certaines zone de l'image.

Par rapport à la version proposée par A.Hertzmann, il manque la gestion de la transparence entre les couches qui se superposent et aussi il manque une part d'aléatoire propre à certains style de peinture. Nous avons limité cette part d'aléatoire au minimum, en modifiant la couleur des coups de pinceau à partir du z aléatoire généré pour le Z-Buffer. Cette variation de la couleur donne à l'image un aspect moins parfait.



2 – Résultats

1 – Paramètres

Les résultats dépendent des paramètres suivants :

- T, le seuil d'approximation de l'image, exprimé en valeur de niveaux de gris. Plus T est élevé, plus l'image peinte est grossière par rapport à l'image d'origine.
- Min et Max, respectivement la taille minimale et la taille maximale d'un coup de pinceau
- Rmax, ratio, et n : respectivement la taille maximale du pinceau(en pixels), le ratio utilisé pour calculer la taille du pinceau des couches suivantes, n le nombre de couches à peindre
- fc, le filtre de courbure, permet d'atténuer ou d'exagérer la courbure des coups de pinceau
- fo, constante utilisée pour augmenter ou diminuer la taille du noyau utilisé pour le lissage de l'image
- fg, constante utilisée pour augmenter ou diminuer l'espacement entre les coups de pinceau

Par défaut on utilisera les valeurs suivantes :

- T = 100
- Min = 4, Max = 16
- Rmax = 8, ratio = 0.5, n = 3
- fc = 1.0
- fo = 1.0
- fg = 1.0



2 – Approximation de l'image originale



Illustration 1: Image originale, utilisée par A.Hertzmann pour illustrer le fonctionnement de ses algorithmes, 736x459

Nous ferons varier l'approximation de l'image de manière à avoir 3 résultats assez différents :

8

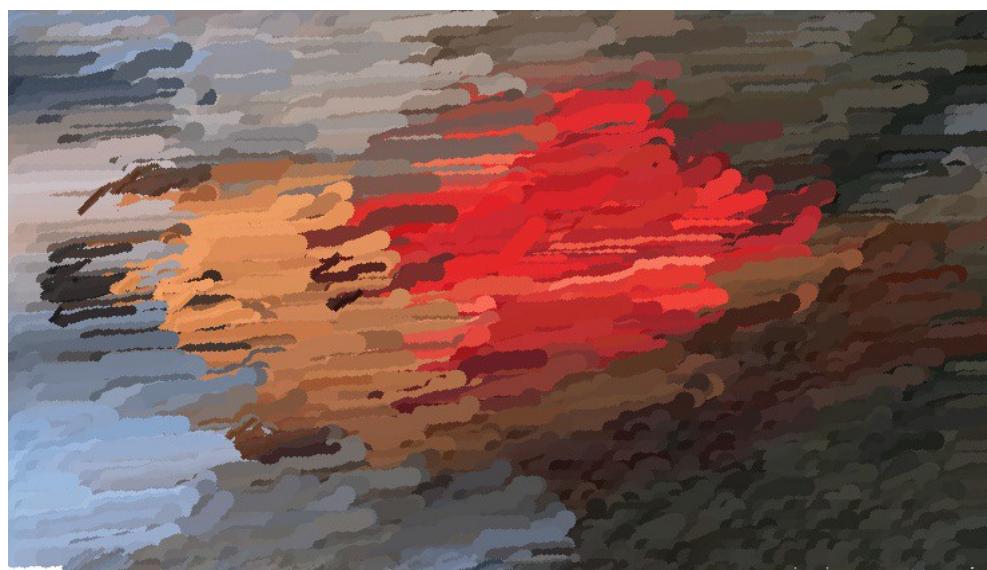


Illustration 2: T = 200, l'image résultante ne reprends que très grossièrement l'apparence de l'image originale

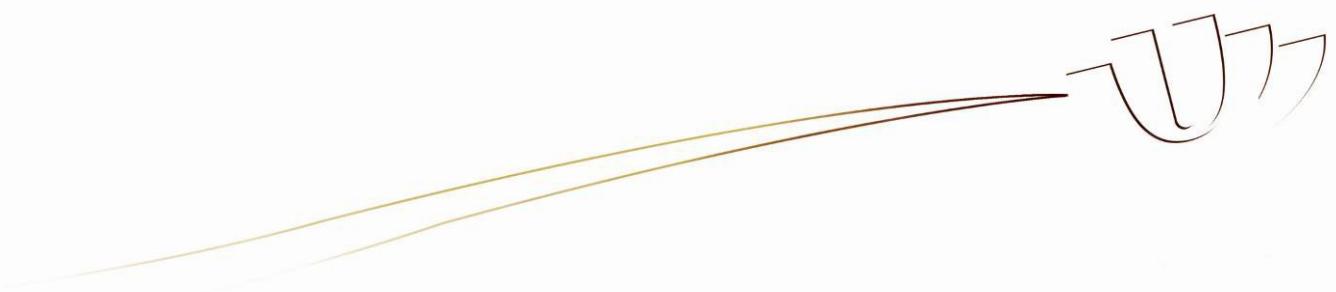




Illustration 3: $T = 100$, A.Hertzmann utilise ces paramètres pour reproduire le style impressioniste

9

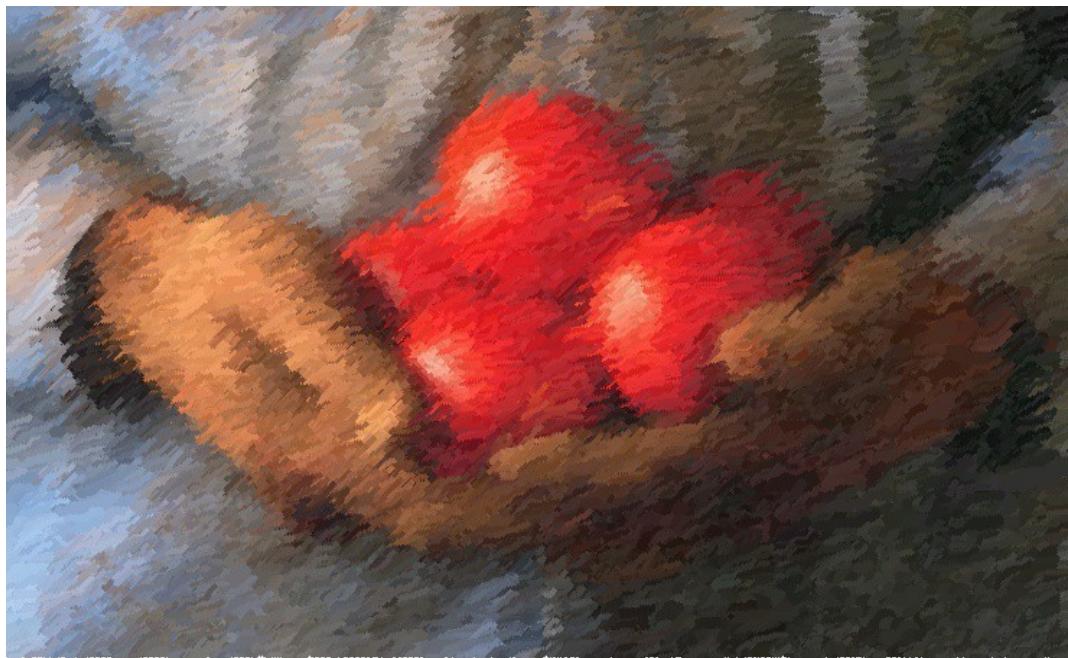


Illustration 4: $T = 25$, l'aspect peinture est plutôt réussi



L'approximation de l'image ne dépend pas que de dépend, elle dépend également de la taille maximale des coups de pinceaux et de la taille des pinceaux.



Illustration 5: $T = 200, n = 1, Rmax = 2$, on obtient un résultat similaire à l'image précédente

10

On peut s'approcher d'un style pointilliste en gardant une approximation faible($T > 100$) et en limitant la taille des coups de pinceau ($\text{Max} < 4$) :



Illustration 6: Image originale, 2500x1600

JJ

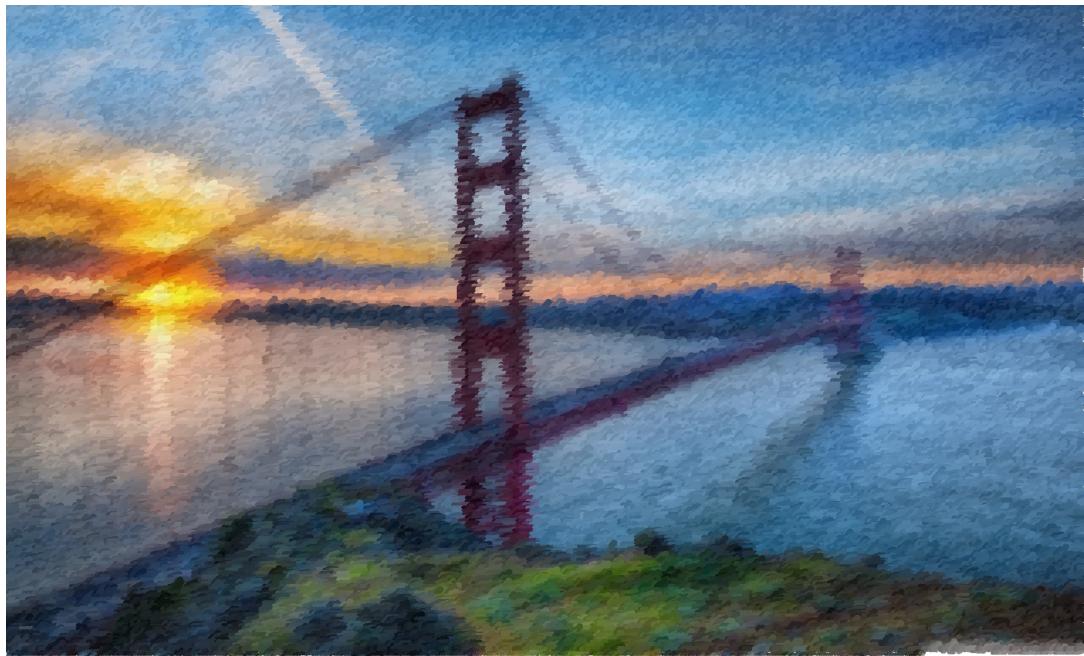


Illustration 7: $T = 200$, $\text{Max} = 4$

11

3 – Le cas particulier des visages

La méthode proposée par A.Hertzmann donne des rendus assez proche de peinture pour des objets ou des paysages par exemples. Mais les visages humains sont souvent déformés lorsque l'on utilise cette méthode. Comme les coups de pinceau sont dessinés aléatoirement, les visages sont rarement mis en valeur. Cette méthode ne donne des résultats satisfaisant pour les portraits que si l'on choisit une approximation très fidèle de l'image originale($T < 50$). Ou alors l'image originale doit avoir une résolution assez grande(supérieur à 1920X1080).



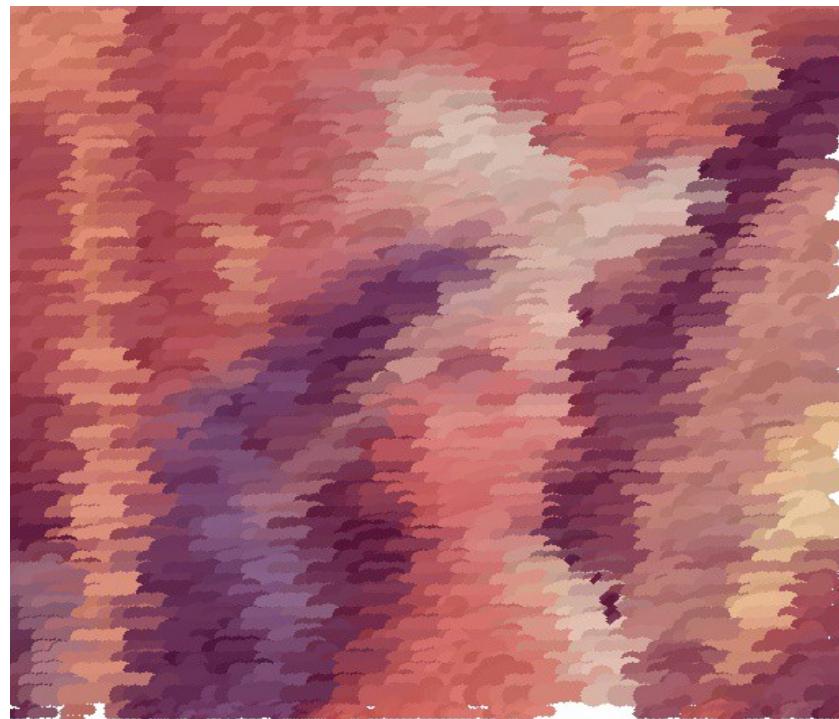


Illustration 8: Lenna, en gardant le style "pointilliste" de l'illustration 7, $T = 200$, $\text{Max} = 4$, 512×512

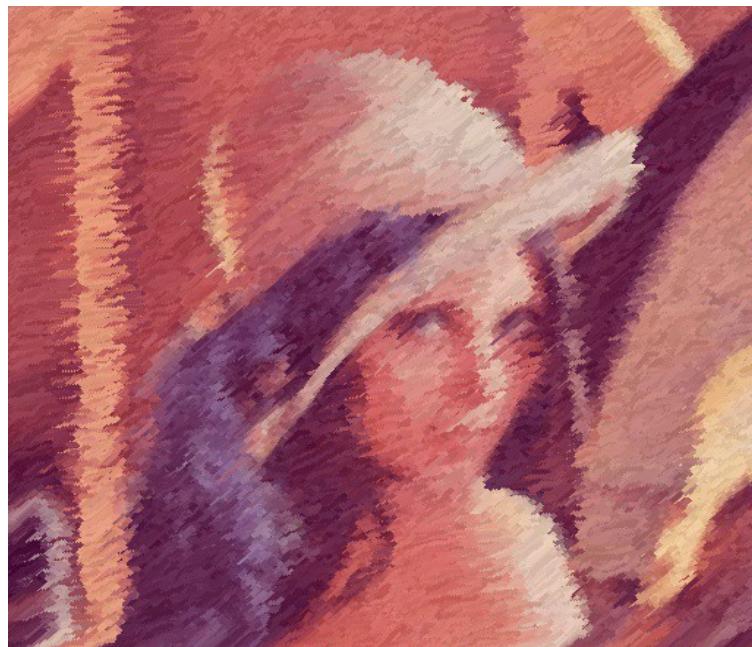
12



Illustration 9: $T = 100$, on reconnaît l'image originale mais — le visage n'est pas mis en valeur comme dans une peinture dessinée par l'Homme, 512×512



On peut obtenir un meilleur résultat en augmentant la résolution de l'image originale et en augmentant la taille du noyau utilisé pour le lissage :



13

Illustration 10: $T = 0, fo = 5, 512x512$

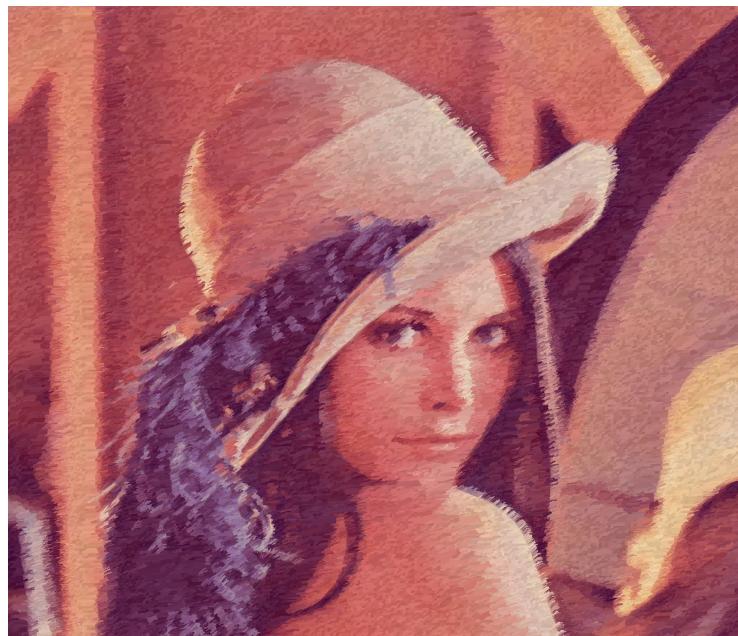


Illustration 11: $T = 50, fo = 5, 2048x2048$



4 – Conclusion

La méthode proposée par A.Hertzmann dans son article publié en 1998 produit des résultats assez proche des peintures faites par l'Homme. Mais cette méthode a un certains nombre de limites. La modélisation des coups de pinceau est limité car les paramètres utilisés restent statiques, ce qui peut justement donner l'impression que ces images ont effectivement été générées par une machine.

Cette méthode n'est pas adapté pour « peindre » des portraits, la machine n'ayant pas la capacité de reconnaître les visages, les résultats sont souvent approximatifs(voir difforme dans certains cas). Il serait intéressant de modifier cette méthode avec des techniques de reconnaissance des visages pour obtenir un résultat plus proche de la réalité.





