適用於硬體實現之非真實照片式繪圖演算法

李杰 郭威廷 張哲榮 呂宗諭 連志原 [1]
國立高雄應用科技大學電子工程系
[1]E-mail：cylien@kuas.edu.tw

摘要

非真實照片式繪圖 (Non-Photorealistic Rendering)，簡稱 NPR，乃是使用演算法產生類似油畫等特殊效果。然而 NPR 的計算複雜度極高，並不利於硬體實現，在本論文中，我們提出了一個低複雜度的 NPR 演算法，我們使用高斯模糊與修改色彩深度來簡化流程，實驗結果顯示我們的方法簡單有效率，且效果接近原始 NPR 的特效。由於此方法計算複雜度很低，所以很適合應用在硬體實現上。

關鍵詞：非真實照片式繪圖、油畫、影像處理

Abstract

Non-Photorealistic Rendering (NPR) is the method to make the painting effect of images. However, the computational complexity of NPR is high and not suitable for hardware implementation. In this paper, a low complexity NPR is proposed. We use Gaussian Blur and image color depth to simply the method. Compared with the NPR technique, our method can make the painting effect simply and efficiently. Since the proposed method requires low computational complexity, it can be applied to many real-time applications.

Keywords: Non-Photorealistic Rendering, painting, image processing.

1. 前言

近年來 3C 產品的蓬勃發展，過去手機與相機都全力追求相片的真實性，不過在經過多年的發展後，真實相片的呈現已經快達到巔峰，所以不少人開始朝著非真實圖片的方向去做發展，近期大家開始用藝術的視角去做研究，其中之一，非真實照片式繪圖 Non-Photorealistic Rendering，NPR 理論因此因應而生。

NPR 是近年來影像處理領域中發展相當迅速的理論之一，主要目的在於使用演算法產生類似油畫、水彩畫等特殊效果。不同於一般所熟知的影像處理，一般影像處理主要是修正破損或失真之影像，修正後的影像越接近原影像就越成功，然而 NPR 理論則是相反，主要是將一般的正常影像套用各種演算法進而實現出各種特效，並以藝術的視角來做修正追求完整。

本論文主要在於改良傳統的 NPR 理論方法，使其適合硬體實現。因為傳統的 NPR 理論方法的計算複雜度極高，所以運算中所需要的時間會十分龐大，不利硬體實現。我們將傳統的 NPR 理論分析並模組化，將部分步驟或或模組使用較簡單的方法替換，進而改良出盡量不失其效果並增加運算速度的演算法。

2. 主要內容

本節一開始我們先介紹何謂非真實照片式繪圖 NPR，接著介紹傳統的方法，以及我們改良的步驟或模組。

2.1 非真實照片繪圖 NPR

近年來，非真實照片式繪圖 NPR 有許多的研究都被提出，並在這幾年 3C 產品的蓬勃發展中扮演重要的角色，在現在隨處可見的智慧型手機中，相機不再是扮演純粹拍照的功能，而是可以依照使用者的喜好套用各種特效，經由 NPR 處理的特效十分的多，包括素描、水彩畫、國畫、雕刻與油畫等等，本論文主要探討油畫的部分。

油畫的影像處理技術就屬 Aaron Hertzmann[1] 所提出的方法較廣為人知，在觀察畫家繪畫時，發現畫家繪畫會針對不同的物體使用不同的筆刷大小及技巧，如圖 1 中，房子、草地、樹木與人都是使用不同的筆刷與使用不同的技巧所繪製而成，因此特別在這兩點上做研究並實現。

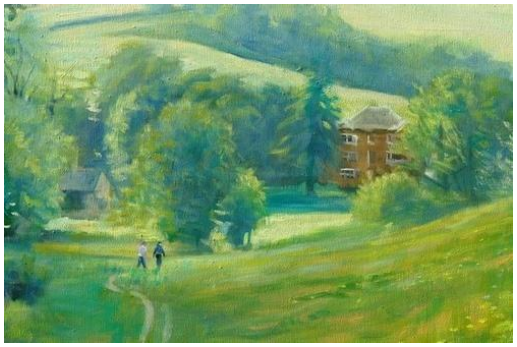

圖 1. 真實油畫

圖 2 是 Aaron 製作筆觸的流程，是利用 B-Spline 產生出具彎曲效果的筆觸。

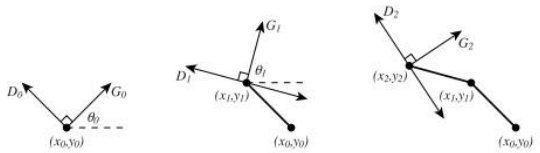

圖 2. 利用 B-Spline 產生之筆觸流程

圖 3 是 Aaron 運用不同的筆刷，他分別運用大中小三種筆刷，在圖片中左邊為原圖右圖則為運算後圖片，由上至下分別為大筆刷、中筆刷與小筆刷。

圖 3. 利用不同筆刷產生之圖片

圖 4 為 Aaron 在觀察上述步驟後，所提出的油畫演算法。

function PAINT(I_s, *// source image*
I_p, *// canvas; initially blank for still images*
$R_1 ... R_n$, *// brush sizes*
firstFrame) *// boolean;* ***true*** *for still images*
Create a summed-area table A from I_s if necessary
refresh ← *firstFrame*
foreach brush size R_i, from largest to smallest, **do**
Compute a blurred reference image I_{R_i} with blur size $f_\sigma R_i$ from A or by convolution
grid ← R_i
Clear depth buffer
foreach position (x, y) on a grid with spacing *grid*
M ← the region $[x - grid/2 ... x + grid/2,$
$y - grid/2 ... y + grid/2]$
areaError ← $\sum_{(i,j)\in M} \|I_p(i,j) - I_{R_i}(i,j)\|$
if *refresh* **or** *areaError* > T **then**
$(x_1, y_1) \leftarrow \arg\max_{(i,j)\in M} \|I_p(i,j) - I_{R_i}(i,j)\|$
PAINTSTROKE($x_1, y_1, I_p, R_i, I_{R_i}$)
refresh ← **false**

圖 4. Aaron 所提出的演算法

可以發現 Aaron 所提出的方法在於將原始圖片套用各種大小的筆刷，並將圖片作曲線的分析，在依照分析後的結果進行繪製。

另外一個由 Michio Shiraishi[2]所提出的演算法，則是先畫大筆刷，在畫中筆刷，最後在畫小筆刷的方式，如圖 5 所示。

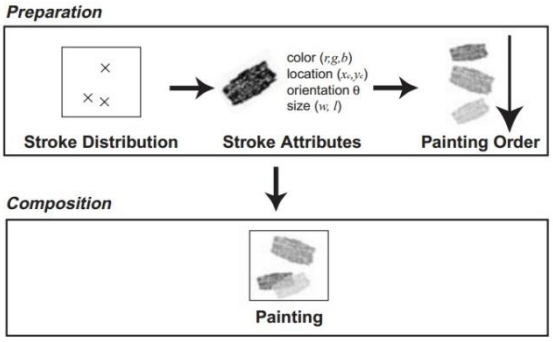

圖 5. Michio 所提出的演算法流程

依照 Michio Shiraishi 所提出的演算法流程，他將所要運算的原始圖片，分成大中小等筆刷所要繪製的部分，接下來再依照大中小的順序進行繪製，繪製的流程如圖 6。

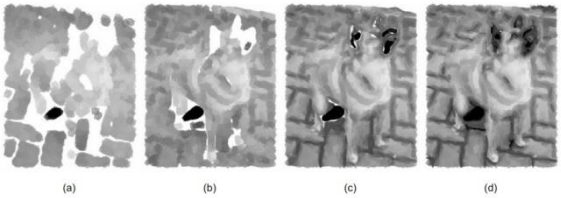

圖 6. 大中小筆刷繪製過程

這種演算的方式，可以讓圖片的渲染力增加不少，不過這跟 Aaron 所提出的演算法有同樣的問題存在，假如要分析再加上繪製的話，所需要的處理時間會變多，由於太複雜的演算，使得運算一張圖片的時間都要花上幾個小時。

因此我們決定捨棄掉分析圖片的工作，不做分析所以也沒有筆刷的選擇，也沒有筆觸的技巧，所以在運算上可以省去大半的時間，在筆刷的部分，我們使用較簡單的分析法，依序取 5x5 的遮罩出來，並尋找遮罩中出現機率最大的顏色來取代中心點的顏色值。

2.2 色彩深度

前述的取代方法經我們實驗後發現出來的效果並不佳，因為 RGB 分別都有 256 色階[3][4]，所以造成在尋找出現機率最大的顏色時，並不能出現我們想要的結果，以圖 7 所示，這是柯達(Kodak)公司提供的全彩無失真影像，我們把它的 RGB 取樣後排列做分析，如圖 8 所示，會發現顏色的資訊過多，因為我們的 NPR 演算部分是在 mask 中尋找出現機率最大的顏色，然後在取代位於中心點的顏色，當色彩的深度過高，就會造成出現的顏色變化過多，一個所選取的 mask 中可能無法找到一種顏色出現兩次以上，這樣就會造成在套用 NPR 演算法後，出現的油畫效果可能會不如預期。

圖 7. 測試用圖

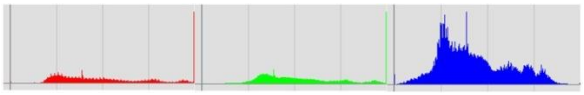

圖 8. 測試圖用 RGB 取樣結果

因此我們決定試著減少 RGB 的色彩深度，並考量漸層感與色彩失真程度，經實驗後選定色彩深度為 5bit，如圖 9 所示。

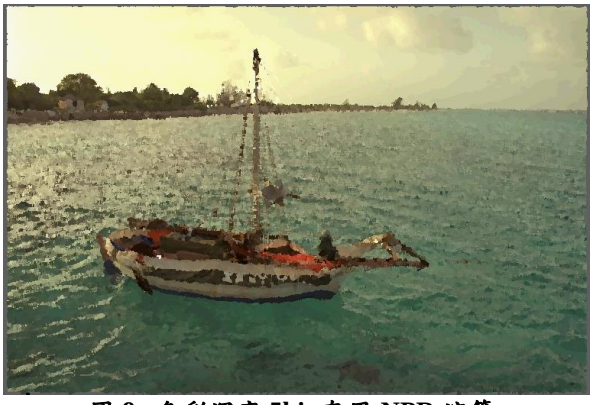

圖 9. 色彩深度 5bit 套用 NPR 演算

2.3 高斯模糊

在改變色彩深度後，我們發現在細節部分處理的不是很好，主要如圖 9 所示，可以發現船上的繩索消失了，由於上述方法是尋找最常出現的顏色，因此細線部分，在遮罩裡所佔的顏色過少所以很容易被周圍取代，在此我們利用高斯模糊(Gaussian Blur)，把邊緣或細節部分的色彩擴散開，來達成我們想要的功效。

高斯模糊是一種使用常態分布的方式，控制每個點的權重，如式 1 所示，這是一個二維空間的高斯函數，高斯模糊就是利用高斯函數計算每一點的權重值，並形成如圖 10，一個高斯常態分布的模型，可以看到中心點的權重較重，並且向外遞減，這樣可以有效的改善在均值模糊遇到的問題。

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \qquad (1)$$

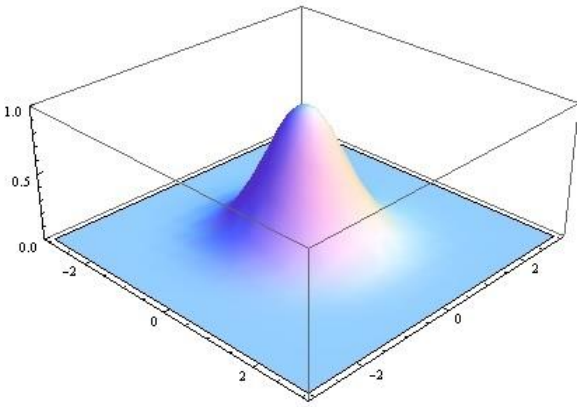

圖 10.高斯模型

圖 11 與圖 12 分別為均值模糊與高斯模糊結果圖，可以發現在均值模糊部份，圖片中所有元素的邊緣都因為模糊化的緣故，都被模糊掉了，造成氛不太清楚元素的邊緣，然而在高斯模糊部分卻大大改善這方面的問題，但是高斯模糊的部份需要取遮罩大小 7x7 才有明顯的模糊化。

圖 11. 5×5 均值模糊

圖 12. 7×7 高斯模糊

3. 實際方法、製作與結果

圖 13. 油畫演算流程

圖 13 為本論文實際油畫演算流程，先經過高斯模糊化將圖片模糊，以達到將細節表現出來，在經過色彩深度演算，將色彩的深度改變，方便之後的 NPR 演算，最後則是套用 NPR 演算。

在模糊化部分，我們決定使用 7x7 的高斯模糊化，因為這樣既能達到模糊效果，也不會因為模糊而改變太多模圖的輪廓。在色彩深度部份，我們使用色彩深度 5bit 的方式，這樣可以維持原圖的基本色彩，不至於發生色彩失真或有明顯漸層感，而且也可以有效的讓 NPR 在尋找色彩時方便運算。最後在 NPR 的部份則是使用 5x5 的 mask 來進行運算，每次尋找 mask 中最常出現的顏色，並取代中間的色彩，這樣的方法捨棄了傳統 NPR 演算中繁瑣的過程。

圖 14 為只有 NPR 演算的圖，可以發現與原圖差異不大，許多邊緣部份都沒有 NPR 演算。圖 15 為加入色彩深度演算後的結果圖，可以發現在邊緣的 NPR 演算部分得到了改善，並且在船上的一些部份，原本看起來很像真實圖片的部分，也在經過運算後開始有油畫的感覺，但是圖片本身的顆粒感還是很嚴重，並且船上的繩索經過運算後，幾乎看不見了。

圖 14. 只有 NPR 演算

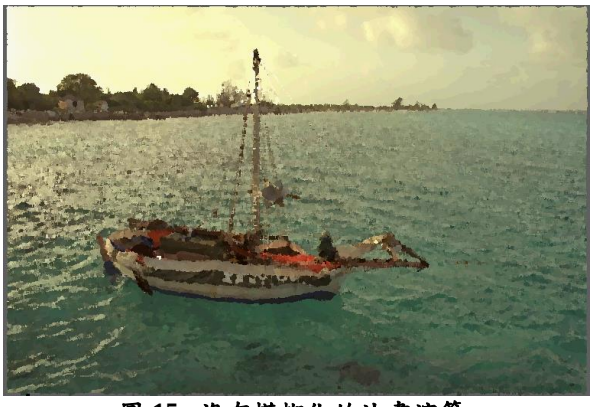

圖 15. 沒有模糊化的油畫演算

圖 16. 油畫演算結果圖

圖 16 為加入模糊化後，本論文真正要表達的結果圖，可以發現在加入模糊化後，之前會發生的顆粒感幾乎看不到了，並且在邊緣的鋸齒感也因為模糊化而平滑掉了，最後在繩索之類的這些細節部分，也因為模糊化的關係而顯現出來。

4. 結論與未來展望

本論文使用了較簡單的方法處理影像，相較於 Aaron 所使用的演算方法，在迭代中每層加入 NPR 的渲染，在處理時間上付出較大的代價，此外，顏色深度的部分也是論文方法的特色，可以有效增強影像效果並縮小儲存大小，在現代手持式裝置追求速度與儲存容量的優點都具備相當的競爭性，未來可以朝這方面去研究，加入更多參數來調整影像，讓此方法更臻至完美。

在未來，如果可以加入硬體的運算，將可以大幅度的提高運算速度，因為如高斯模糊化或色彩深度改變等，都是可以進行平行運算的演算法，若在硬體中加入管線化，再搭配軟體製作同步處理，在現今 3C 產品發達的時代，可以讓油畫、水彩畫或國畫等特殊效果，也能在手持式產品中有所發展。

參考文獻

[1] Aaron Hertzmann, “Painterly rendering with curved brush strokes of multiple size,” in Proc. ACM SIGGRAPH, p.453-460, 1998.

[2] Michio Shiraishi and Yasushi Yamaguchi, “An Algorithm For Automatic Painterly Rendering Based On Local Source Image Approximation,” in Proceedings of the first international symposium on Non-photorealistic animation and rendering, p.53-58, 2000

[3] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 2nd Edition, NJ: Prentice-Hall, 2002.

[4] A. K. Jain, Fundamentals of Digital Image Processing, NJ: Prentice-Hall, 1989.