Style Transfer for Interactive 3D Environments

by

Jianghan Xue, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2014

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Jianghan Xue

September 2, 2014

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Jianghan Xue

September 2, 2014

Acknowledgments

This project would not have been progressing successfully so far without the support of my supervisor. I wish to express my gratitude to my supervisor, Dr. John Dingliana, who was abundantly helpful and offered invaluable assistance, support and guidance.

I really appreciate the help from my friends, who have supported me over the last few years.

Finally, I am indebted to my beloved parents and grandparents for their continuous, selfless support and encouragement.

Jianghan Xue

University of Dublin, Trinity College

September 2014

Style Transfer for Interactive 3D Environments

Jianghan Xue, M.Sc.
University of Dublin, Trinity College, 2014

Supervisor: John Dingliana

After over 40 years of development on computer graphics, many problems of generating photorealistic images by computer have been solved. In the early 1990s, some researchers turned to a new area called Non-Photorealistic Rendering, which focuses on creating digital art with a wide variety of artist styles, such as painting, and cartoon.

With so many NPR techniques available for various works at stylisation, we begin to think about not only rendering a scene in a random style, but also transferring styles. In the field of computer vision and 3D rendering, style transfer generally refers to the transfer of characteristics from one image or 3D scene to another. The common characteristics include colours, shapes, textures, brush-strokes, painting media, etc. An ideal style transfer process could be used in 3D modelling to decrease the amount of manual authoring work developing specific assets for 3D models. It could also help to blend a 3D virtual scene and a stylised background together in a coherent style in a film.

This paper investigates the current states of style transfer, especially that applies the artistic style of an image onto a target 3D scene. It verified the possibility of transferring colours of a style image onto the textures of 3D objects based on their positions, and the

integration of watercolour rendering after transferring colours. But many problems are still unsolved if we move onto transferring style-specific features.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

After over 40 years of development on computer graphics, many problems of generating photorealistic images by computer have been solved. Even very complex scenes with a large number of highly detailed objects can be generated. In the early 1990s, some researchers turned to a new type of quest. In contrast to traditional computer graphics, they focus on creating digital art with a wide variety of artist styles, such as painting, and cartoon. This new area of endeavour has become known as *Non-Photorealistic Rendering*, or NPR for short. [1]

Many NPR techniques now exist and have already been implemented in some popular games (e.g. The Legend of Zelda, XIII, Team Fortress 2, Prince of Persia 4), and movies (e.g. Tarzan). It is also used for technical illustration, medical analysis, etc.

1.1 Motivation

With so many NPR techniques available for various works at stylisation, we begin to think about not only rendering a scene in a random style, but also applying the artistic style of a painting onto a target 3D scene, in an efficient and optimised way.

In the field of computer vision and 3D rendering, style transfer generally refers to the transfer of characteristics from one image or 3D scene to another. The common

characteristics include colours, shapes, textures, brush-strokes, painting media, etc.

An ideal style transfer process could be used in 3D modelling to decrease the amount of manual authoring work developing specific assets for 3D models. It could also help to blend a 3D virtual scene and a stylised background together in a coherent style in a film.

1.2 This Dissertation

Chapter 2 discusses the related work in field of art theory, colour theory, image processing, style transfer, non-photorealistic rendering, and style identification.

Chapter 3 describes the details of the method designed for achieving the objective of this project.

Chapter 4 discusses the options of implementation platform, the decision made, and the reasons to justice the decision.

Chapter 5 evaluates the performance of every independent steps and the overall style transfer process.

Chapter 6 concludes results and contributions. Challenges and future works is discussed in this chapter as well.

Chapter 2

State of the Art

2.1 Definition of Style

In the visual arts, the style of a painting is a "distinctive manner which permits the grouping of works into related categories." [2]. Some categories are explicit:

- Implement: brush, pencil, sponge, knife, airbrush, etc. (and their size, shape, edge)
- Medium: oil, watercolour, pastel, ink (and their colour)
- Surface: canvas, paper, wall
- Object: figure, portrait, landscape, still life
- Specified purpose: poster, cartoon/comic/manga, illustration

Some categories are more subjective and difficult to describe, such as realism, impressionism, abstract expressionism, surrealism, photorealism, etc.

2.2 Image Processing

2.2.1 Colour Model

Colour is the most basic style for an image. In order to store colours in computers, we introduced *Colour Space*, which allows to reproduce colours from their representations in the computers. A colour space is based on a colour model, an abstract mathematical model describing the way colours can be represented. The most well-known colour spaces, models and their features are listed below.

- RGB: an additive colour model, which mixes red, green and blue (three additive primary colours) in different ratio to produce various colours. RGB colour model is the most common representation used in digital devices to capture, store, and display images.

- CMY: a subtractive colour model. The values of the cyan, magenta, and yellow are subtracted from pure white in different ratio in order to get the required colour. For this reason, it is often used as a colour space in printers where the white of paper is the original colour.

- YUV: a colour model that originates from the early age of colour television [3]. The analogue signal of black-and-white television has only the Y (luminance, luma, brightness) value. Colour television added U and V values as the chrominance (colour) components, in order to be compatible with black-and-white.

- YCbCr: a compressed, shifted and digital version of YUV. As human vision system is much more sensitive to brightness than chrominance, the size of Cr and Cb components can be reduced. For this reason, YCbCr is often used in image and video compression.

- HSL and HSV: representing RGB colours in a cylindrical-coordinate system. Colours are represented as H (hue), S (saturation), and L (luminance) or V (value) values.

HSV is also called HSB (brightness). These two colour spaces are designed to be more intuitive. We can tell a colour's name easily by its hue, the most important component of a colour. For this reason, these two colour spaces are often used in colour pickers in image-editing software.

- Lab: a colour model designed to approach human vision system. L stands for lightness and a and b for the colour component. The Lab colour space used in contemporary days is designed by CIE (the International Commission on Illumination) in 1976, so it is also called CIE-LAB.

2.2.2 Colour Difference

Colour difference is a metric used to quantify the difference between two colours. Before colour difference metric was introduced, only subjective adjectives can be used to describe colour differences. The colour difference metric is very useful in image clustering, which will be described in the next section.

CIEDE76 is the first colour difference formula that applies to colours in CIE-LAB colour space. It is essentially the Euclidean distance between two colour vectors.

Given (L_1^*, a_1^*, b_1^*) and (L_2^*, a_2^*, b_2^*), two colors in CIE-LAB colour space, CIE76 defines their difference ΔE_{ab}^* as

$$\Delta E_{ab}^* = \sqrt{(L_2^* - L_1^*)^2 + (a_2^* - a_1^*)^2 + (b_2^* - b_1^*)^2}$$

$\Delta E_{ab}^* \approx 2.3$ corresponds to a just noticeable difference.

Because CIE-LAB colour space turned out to be not as perceptually uniform as intended, CIEDE76 is replaced by its successors CIE94 and CIEDE2000 [4], by introducing several weights to the formula. However, the cost of more accurate colour difference is the increase of algorithm complexity and calculation time.

2.2.3 Image Clustering

In the field of style transfer, image clustering can be used to find different objects which has notably different colours in a simple image. Image clustering is the task of grouping the pixels in an image in such a way that the colour of the pixels in the same group (also called a cluster) are more similar to each other than to those in other groups.

k-means clustering is one of the most popular methods for image clustering. It is created by several people independently [5] [6] [7]. It aims to group pixels into k (a user-controlled parameter) clusters in which each pixel is closer to the mean of its cluster than that of the others.

The standard algorithm of k-mean clustering uses an iterative refinement technique. It first chooses k pixels as the initial centres of k clusters. The centres can be selected randomly, or through a well-designed algorithm like k-means++ [8]. The algorithm then assigns each remaining pixel to a nearest cluster according to the distance between the pixel and the cluster centre. The distance between two pixels can take into account both the spatial distance and the colour difference. It then recalculates the mean of the cluster to be the new cluster centre.

After a few iterations, the process is terminated once it meets one of the termination criteria: a) the algorithm converges (the accuracy is smaller than a user-controlled threshold), or b) the maximum number of iterations is reached. The accuracy V of the clustering is defined as

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2$$

where k is the number of clusters, S_i is the set of pixels in cluster i, x_j is a pixel in S_i, μ_i is the mean (centre) of pixels in S_i.

2.3 Style Transfer

2.3.1 Colour Transfer between Images

Colour transfer between images aim to apply a colour palette, mood or style from one image to another [9]. One of the earliest paper in this field is published by Reinhard et al. [10] in 2001. This paper introduced a global colour transfer method. They use a simple statistics analysis method to impose one image's colour characteristics on another. The images are firstly converted from RPG colour space to l$\alpha\beta$ colour space [11]. After then, they compute the images' mean and standard deviation, and use them to globally shift the target image's colour.

They chose l$\alpha\beta$ colour space instead of RGB because different channels' values are correlated in RGB, as well as many other colour spaces. For example, in RGB space, most pixels will have large values for the red and green channel if the blue channel is large. This feature means if we want to change the appearance of a colour coherently, we must modify all channels together.

Given the assumption that decorrelated colour spaces would perform better, Reinhard et al. published another paper [9] in 2011, discussing more detailedly about the correlation of various colour spaces. The chosen colour spaces are L, CIE-LAB (using both D65 and E illuminants), Yxy, Yuv, Yuv, XYZ, RGB and HSV. They use about 350 images altogether in four categories as the sample. They then transfer the colours of the images in different colour spaces and calculate the covariance matrix and the average covariance to measure the degrees of channel correlation of colour spaces.

As the conclusion, they found the covariance of a colour space with respect to a set of images, is a good measure of how well such a space is likely to perform better colour transfer. More importantly, they found CIE-LAB, used with illuminant E as the white point, showed the best performance on average, giving a plausible colour transfer in about 77 % of all tested samples, followed by, CIE-LAB (D65; 73% success) and L (71%) success. All other chosen colour spaces do not perform acceptably well in their tests.

The shortcoming of CIE-LAB, however, is the cost of conversion. Because RGB colour model is device-dependent, it has to be transformed to a specific absolute colour space, such as sRGB, which is device-independent, allowing data to be converted to CIE 1931 colour space and then to CIE-LAB. For this reason, other colour spaces like YCbCr [12] are seen to be used.

2.3.2 Colour Transfer from Image to 3D Scene

Nguyen et al. [13] implemented the colour transfer techniques between 2D and 3D. They extract multiple Phong materials (ambient, diffuse and specular colour [14]) from the source image and assign the materials to 3D objects based on statistical cost-based functions taking into account the positions and the shapes of objects.

While extracting the materials, they first split the style image into diffuse and specular radiance using the specular highlight removal method of Yang [15]. The diffuse radiance is then split into a base and a texture part using the fast bilateral filtering of Durand [16]. The base part of diffuse pixels are then clustered using k-means, applying the CIEDE2000 colour difference metric. After merging similar neighbour clusters, one material is generated for each cluster. The diffuse colour of the material is specified by the colour of the cluster.

For the material assignment, they introduced a cost function d(A, V):

$$d(A, V) = w_i d_i(r(A, V)) + w_g d_g(A)$$

where A is an material assignment, V is a camera view, d_i is an image cost function that calculates the histogram difference of the cluster and the object under view V, d_g is a view-independent geometry cost function that compares the shapes, w_i and w_g are the relative weights between d_i and d_g respectively.

According to the definition of the cost functions, the smaller d is, the more similar the object and the cluster are, under assignment A and view V. The material assignment

stage in their paper is to loop through all Cartesian products of clusters in the style image and objects in the 3D scene, and to find the pairs with the smallest cost for every objects.

2.3.3 Style Rendering

In Nguyen's paper, after extracting and assigning Phong materials, they render the 3D objects using Phong lighting, which is a basic photorealistic rendering technique. On the other hand, there are many non-photorealistic techniques available, suitable for rendering objects in artistic styles.

One of the most basic NPR techniques is toon shading (also called cel shading) [17], which is used to mimic the lighting style of a comic book or cartoon. In comic books, the part of an object that is facing the light source is generally brighter, and the part that is back to the light source is darker. In order to mimic this feature, toon shading calculates the angle between the normal of a vertex and the light direction, classifies the angle into several levels of brightness from black to white, and modifies the colour of the pixel accordingly.

Another example of a basic NPR technique is silhouettes and outlines [18]. They are used to emphasise the edges of an object in comic books. They can be achieved by darkening the vertices whose corresponding points in the depth map or the normal map change significantly from their neighbours.

These two basic NPR techniques can serve as the starting point of other advanced techniques, like oil, ink, pastel, and watercolour rendering.

In 1997, Curtis et al. [19] described the specific features that make watercolour style so distinct among painted media, and they introduced a model to simulate the physical flow of pigment and the impact of paper to reproduce watercolour painting. Different types of pigment effects, like drybrush, edge darkening, backruns, granulation, flow effects, and gazing, are generated in different layers. They are merged together to create the final effect of watercolour.

The shortcoming of Curtis's method is the complexity of calculation. In the original paper, they stated that it took them 7 hours to generate a watercolour painting that consists of 11 glazes, using a total of 2750 iterations of the simulator, rendered at a resolution of 640 by 480 pixels on a 133 MHz SGI R4600 processor.

Bousseau et al. [20] decreased the complexity of the algorithm by introducing several pre-generated textures that filters the colours of 3D model in fragment shader and during post-processing. Pre-generated textures turned out to be adequate to replace the physical simulation of watercolour pigment for reproducing most of the distinct watercolour effects defined by Curtis. In their paper, Bousseau et al. also proposed two methods to produce temporally coherent animations that keep a uniform pigment repartition while avoiding the shower door effect.

2.3.4 Style Identification

No general-purpose style extraction technique has been proposed so far. The current style extraction techniques are targeting colours or a specified style, like watercolour. For this reason, we need to manually specify what kind of style we want to extract from an image. Theoretically the specification can be done by algorithms, which is called style identification or classification.

The work of Shamir et al. [21] shows that "the automatic computer analysis can group artists by their artistic movements, and provide a map of similarities and influential links that is largely in agreement with the analysis of art historians". Their method is based on WND-CHARM scheme [22], which contains a set of 4027 features that reflects many aspects of the visual content, such as shapes, textures, edges, colours, etc.

Chapter 3

Design

This chapter describes the details of the method designed for achieving the objective of this project: applying the artistic style of an image onto a target 3D scene.

3.1 Method Overview

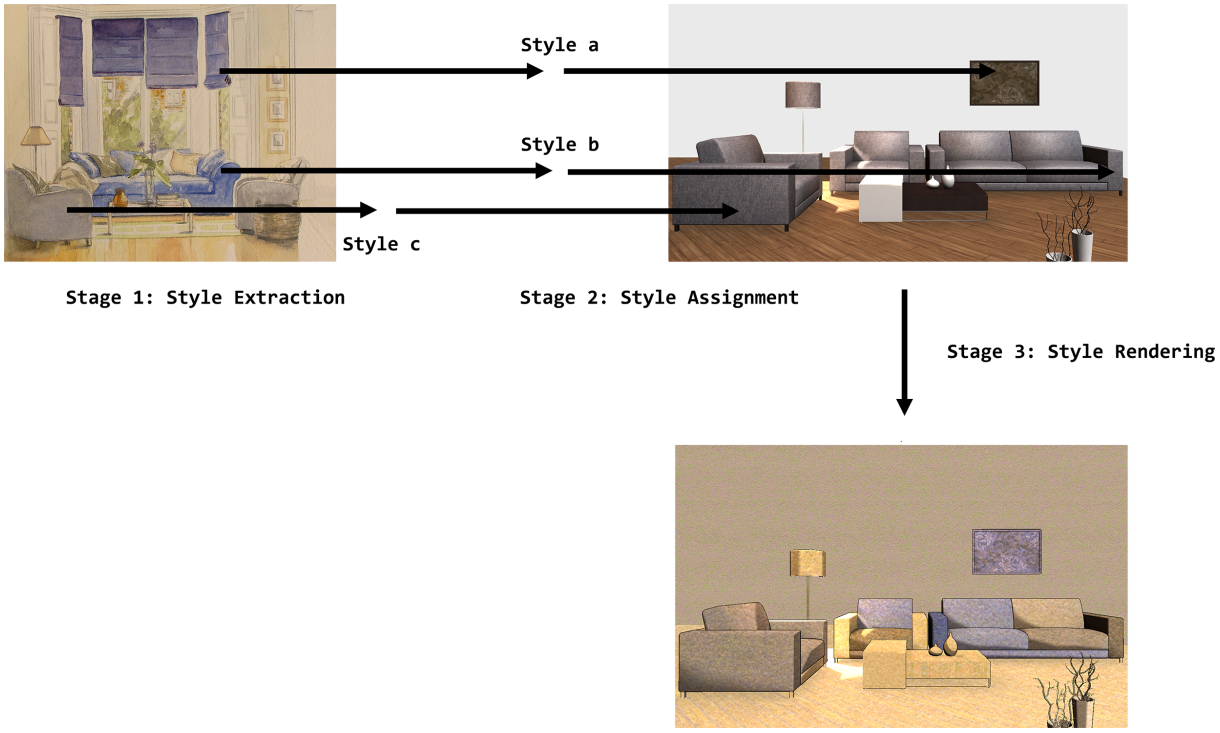


Figure 3.1: The outline of the method designed for achieving the objective of this project.

The process of this method can be classified into three stages: style extraction [13], assignment [13], and rendering. See Figure 3.1

1. Style Extraction: Clustering the style image into several groups and extracting a set of rendering parameters, like colours, from each cluster.

2. Style Assignment: Finding the corresponding 3D objects in the original scene for each cluster, and transferring the colour to the textures of the 3D objects.

3. Style Rendering: Rendering objects with given parameters in a specific style.

3.2 Style Extraction

First of all, the RGB values in the style image are converted from RGB to CIE-LAB colour space using OpenCV built-in functions. The reason of choosing CIE-LAB is described in Section 2.3.1 Colour Transfer Between Images.

The style image is then clustered using k-means provided by OpenCV.

Before clustering, the pixels should be transformed in order to meet the requirement for better result of clustering. CIE-LAB values are scaled to the range from 0 to 1. In order to taking into account positions, the parameters passed to k-means function should also include the position coordinates, which scaled to the range from 0 to 1 as well by dividing the width or height of the image.

For each pixel, two position coordinates and three channels of CIE-LAB values form a five-element vector. All pixels form a list of vectors, which serve as the samples and are then passed to the k-means function. See Figure 3.2.

We choose k-means++ [8] as the centre initialization algorithm. Other parameters include maximum iteration 100, epsilon 0.1, and 5 attempts. The number of clusters k is a user-controlled parameter.

In Nguyen's paper [13], CIEDE2000 is used as the colour difference metric during clustering. But considering the cost of calculating CIEDE2000 formula, they are not

$$
\begin{gathered}
(x_0, y_0, l_0, a_0, b_0) \\
(x_1, y_1, l_1, a_1, b_1) \\
(x_2, y_2, l_2, a_2, b_2) \\
(x_3, y_3, l_3, a_3, b_3) \\
..... \\
(x_n, y_n, l_n, a_n, b_n)
\end{gathered}
$$

Figure 3.2: The sample data passed to k-means function

applied in this project. Instead, CIE76 colour difference, which is essentially the Euclidean distance between two colours, is applied.

Figure 3.3 shows the style image before and after being clustered.

Figure 3.3: Left: The style image before clustering. Right: The style image clustered into 30 groups

After clustering, the mean and the standard deviation are calculated for each cluster. These 2 parameters will be used in the following stages.

3.3 Style Assignment

In this stage, we first loop through all objects in the scene.

According to the screen position of each object, we find the corresponding cluster that

the object belongs to in the style image. The corresponding position (x_{style}, y_{style}) in the style image is calculated the formula:

$$x_{style} = \frac{x_{orig}}{w_{orig}} w_{style}$$
$$y_{style} = \frac{y_{orig}}{h_{orig}} h_{style}$$

where (x_{orig}, y_{orig}) is the screen position of the object, (w_{orig}, h_{orig}) is the resolution of the original texture, (w_{style}, h_{style}) is the resolution of the style image.

Once we find the cluster, we know the mean and the standard deviation of the pixels in that cluster. We then take the main texture of the object, convert it into CIE-LAB colour space, and shift each pixel by the mean and the standard deviation of the corresponding cluster, using the method in the paper of Reinhard et al. [10]. The shifting algorithm is described below.

We first subtract the mean from the colour of the original texture:

$$l^* = l - \overline{l_{orig}}$$
$$a^* = a - \overline{a_{orig}}$$
$$b^* = b - \overline{b_{orig}}$$

Then, we scale the colour by the ratio of the standard deviation of the style image and that of the original texture, and we add the mean of the style image.

$$l' = \frac{\sigma^l_{style}}{\sigma^l_{orig}} l^* + \overline{l_{style}}$$
$$a' = \frac{\sigma^a_{style}}{\sigma^a_{orig}} a^* + \overline{l_{style}}$$
$$b' = \frac{\sigma^b_{style}}{\sigma^b_{orig}} b^* + \overline{l_{style}}$$

where

- (l', a', b') is the result colour of the pixel in the original texture after transferring,
- (l, a, b) is the original colour of that pixel,
- $\sigma^l_{style}, \sigma^a_{style}, \sigma^b_{style}$: the standard deviation of three channels in the style image,

- $\sigma^l_{orig}, \sigma^a_{orig}, \sigma^b_{orig}$ is the standard deviation of three channels in the original texture,
- $\overline{l_{orig}}, \overline{a_{orig}}, \overline{b_{orig}}$ is the means of three channels in the original texture.

After transferring, the texture is converted back to RGB colour space. Figure 3.4 shows the process of style assignment and transfer.

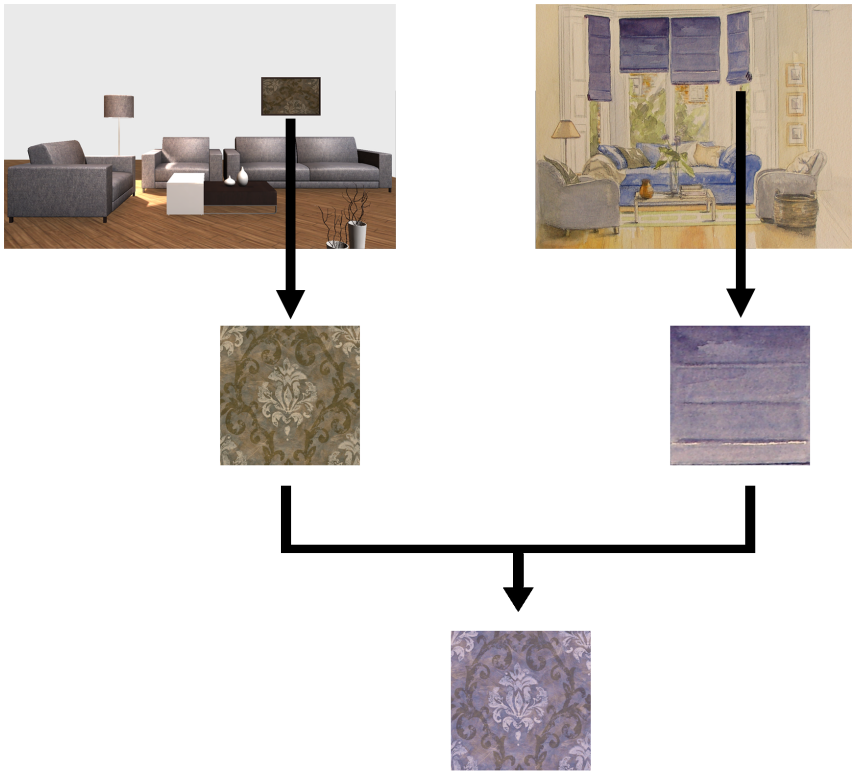


Figure 3.4: Colour transfer from a cluster in the style image (right) to the texture of a corresponding object (left).

3.4 Style Rendering

This project uses a similar method to Bousseau et al. [20] to render the scene in watercolour style. Figure 3.5 shows the watercolour rendering pipeline for the sample 3D scene.

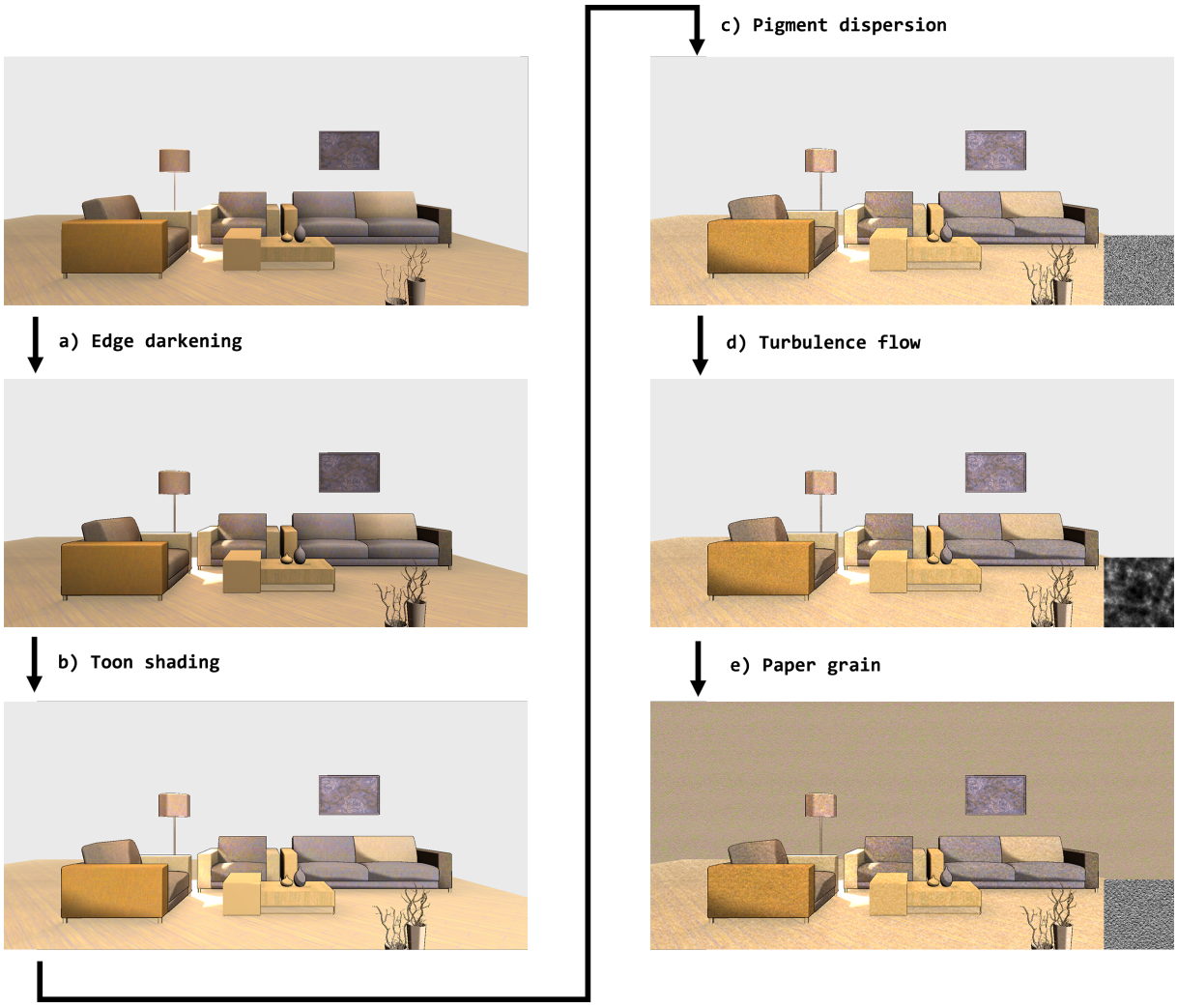


Figure 3.5: The watercolour rendering pipeline for the sample 3D scene.

3.4.1 Edge Darkening

The first step of watercolour rendering is edge darkening. Edges are detected on farme buffer in GPU using a fast, symmetric, first derivative edge detector:

$$\delta_1(i,j) = p_{x-1,y} - p_{x+1,y}$$
$$\delta_2(i,j) = p_{x,y-1} - p_{x,y+1}$$

and the gradient of each channel is computed by:

$$\Delta(p_{x,y}) = |\delta_1(i,j)| + |\delta_2(i,j)|$$

The average of the gradients of three channels is used as the overall gradient of a pixel. If the gradient is larger than a user-controlled threshold, we use a specific outline colour to darken the pixel.

This edge detector is clearly focusing on the efficiency. Other gradient computations can be used to get better edge detecting result but with the cost of efficiency decrease.

3.4.2 Toon Shading

We first calculate the dot product d of the vertex normal n and the light direction l:

$$d = n \cdot l = |n||l|cos(\theta_{n,l}) = cos(\theta_{n,l})$$

The value range of d is the range of cosine which is -1 to 1. We then multiply d by 0.5 and add 0.5 to change its range to 0 to 1.

$$d' = d \times 0.5 + 0.5$$

The larger d' is, the closer the pixel is facing the light source. We then create a three-level ramp texture to modify the colour of pixels. See Figure 3.6.

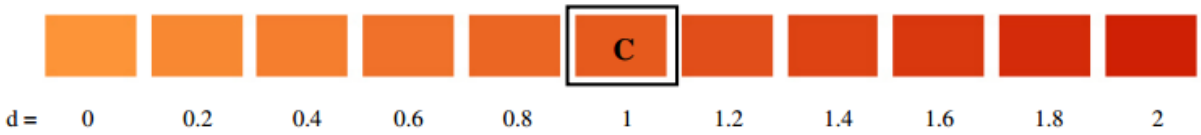
Figure 3.6: Three-level ramp used in toon shading.

3.4.3 Colour Modification

In order to decrease the complexity of physical simulated watercolour rendering, Bousseau et al. [20] introduced three pre-generated textures that modify the colours of pixels for

reproducing most of the distinct watercolour effects defined by Curtis.

These three textures are designed to mimic pigment dispersion, turbulence flow and paper grain respectively. Each texture is a grey-scale image whose intensity $T \in [0, 1]$ gives the pigment density $d = 1 + \beta(T - 0.5)$, where β is a global parameter specifying the weight. Pigment density is used to modify the brightness of the original colour. The larger (whiter) the intensity or the pigment density is, the darker the brightness will be. Pigment density 1 (Intensity 0.5) yields the original colour. See Figure 3.7.



d = 0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2

Figure 3.7: Darkening and brightening of a base colour C (0.90, 0.35, 0.12) given different density parameters d, using Bousseau's original colour modification formula

Bousseau's paper calculates the result colour C' from the original colour C using the formula:

$$\begin{aligned} C' &= C(1 - (1 - C)(d - 1)) \\ &= C - (C - C^2)(d - 1) \end{aligned}$$

Although this formula is quite efficient, it has a serious flaw: it does not work for colours of which all the three channels are either 1 or 0, for example red (1, 0, 0). The reason of this flow is that 1 or 0 makes $(C - C^2)$ always equals to 0, which means the formula always gives the original colour. Figure 3.8 shows a few examples of colours modified by the original formula.

The other flaw of this formula is that it adds 1 to T to get d: $d = 1 + \beta(T - 0.5)$, but it always subtract 1 from d again to modify the colour $C' = C - (C - C^2)(d - 1)$.

The improved colour modification method used in this project solves the problem. We convert colours into HSL colour space, and modify lightness values according to the given density. The details of the conversion and the modification are described below.

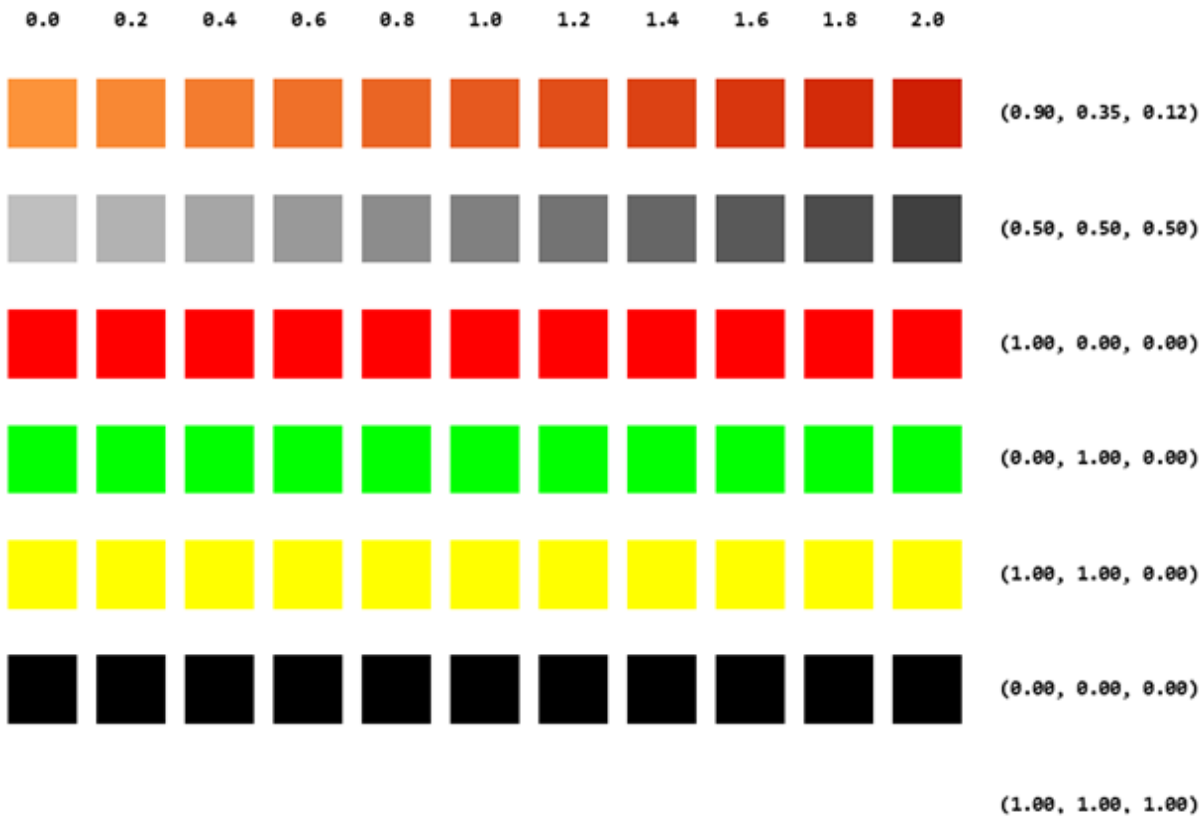


Figure 3.8: Darkening and brightening of a series of colours given different density parameters d (at the top of the figure), using Bousseau's original colour modification formula

We first calculate the pigment density d similarly: $d = \beta(T - 0.5)$, and convert RGB values on the object textures to HSL using the following formula: [23]:

$$h = \begin{cases} 0^\circ & \text{if } max = min \\ 60^\circ \times \frac{g-b}{max-min} + 0^\circ, & \text{if } max = r \text{ and } g \geq b \\ 60^\circ \times \frac{g-b}{max-min} + 360^\circ, & \text{if } max = r \text{ and } g < b \\ 60^\circ \times \frac{b-r}{max-min} + 120^\circ, & \text{if } max = g \\ 60^\circ \times \frac{r-g}{max-min} + 240^\circ, & \text{if } max = b \end{cases}$$

$$l = \frac{1}{2}(max + min)$$

$$s = \begin{cases} 0 & \text{if } l = 0 \text{ or } max = min \\ \frac{max-min}{max+min} = \frac{max-min}{2l}, & \text{if } 0 < l \leq \frac{1}{2} \\ \frac{max-min}{2-(max+min)} = \frac{max-min}{2-2l}, & \text{if } l > \frac{1}{2} \end{cases}$$

We then modify the lightness channel:

$$l' = l + d$$

and convert HSL values back to RGB [23]:

$$q = \begin{cases} l \times (1+s), & \text{if } l < \frac{1}{2} \\ l + s - (l \times s), & \text{if } l \geq \frac{1}{2} \end{cases}$$

$$p = 2 \times l - q$$

$$h_k = \frac{h}{360}$$

$$t_R = h_k + \frac{1}{3}$$

$$t_G = h_k$$

$$t_B = h_k - \frac{1}{3}$$

$$\text{if } t_C < 0 \rightarrow t_C = t_C + 1.0 \quad \text{for each } C \in \{R, G, B\}$$

$$\text{if } t_C > 1 \rightarrow t_C = t_C - 1.0 \quad \text{for each } C \in \{R, G, B\}$$

As Figure 3.9 shows, the improved colour modification formula not only modifies the colours smoothly like the original formula, but also supports "pure" colours.

3.4.4 Three Pigment Density Variation

As discussed in the previous section, Bousseau introduced three textures to mimic pigment dispersion, turbulence flow and paper grain respectively. Bousseau called these three watercolour effects as pigment density variation.

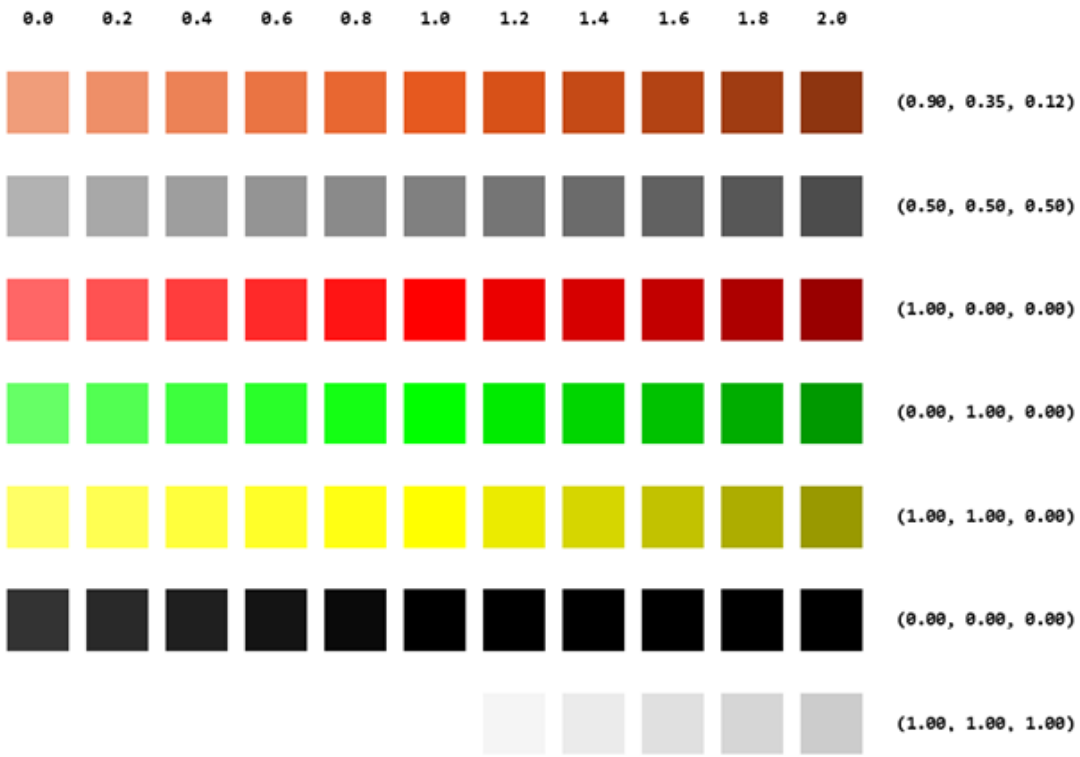


Figure 3.9: Darkening and brightening of a series of colours given different density parameters d (at the top of the figure), using the improved colour modification formula in this project. Note that pigment densities shown here is still from 0 to 2, in order to be compatible with the previous figures.

We are free to choose any kind of grey-level textures for the three effects. Bousseau found it plausible to use a sum of Gaussian noises at different scales for pigment dispersion, Perlin noise textures [24] for the turbulence flow, and scanned paper for the paper grain.

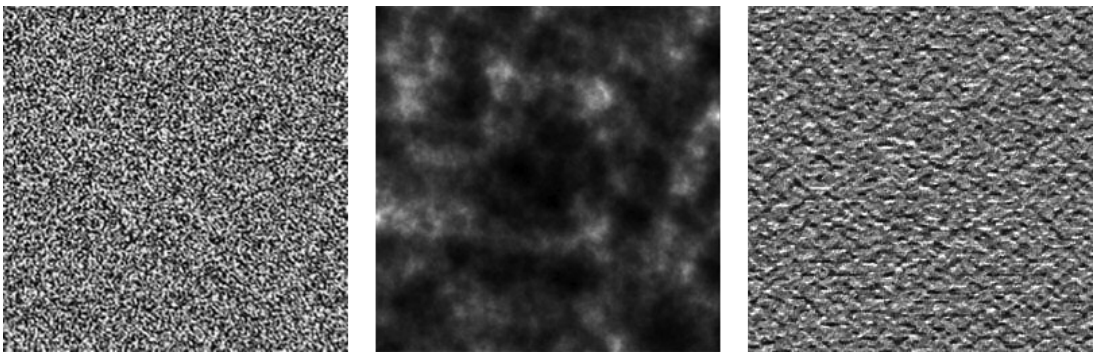


Figure 3.10: Three pigment density variation textures. Left: pigment dispersion. Middle: turbulence flow. Right: paper grain

Chapter 4

Implementation Details

In this chapter we discuss various implementation platforms, their advantages and shortcomings. We also describe all the development and runtime environments we use.

4.1 Platform

As this project needs to use the techniques of 3D rendering and computer vision, it requires the support of related libraries. Many frameworks can be used as the rendering platform. Because of the difficulty in investigation, there are not many formal researches into the exact market shares of the frameworks. Table 4.1 and the following list show some of the most popular options and their features related to this project.

- Low-level graphics APIs, for example OpenGL, DirectX, Cg, and Mantle. The advantages of using low-level graphics API include high performance, richness of third-party libraries. But the shortcomings are very notable as well, which is mostly about low development efficiency. Developers have to write lots of redundant code to implement basic functionalities, like model loading, shader creation, shader property assignment, etc. Additionaly, the lack of an interactive editor makes it hard to create and debug scenes, animations, and GUI.

- Ogre. Like many other open source game engines, Ogre provides a set of API that

Table 4.1: Comparison of popular game frameworks and their features realated to this project.

	Functionalities	Interactive Editor	Learning Curve	Scripting Languages
Low-level APIs	From scratch or 3rd-party libraries	No	Steep	C, C++
Ogre	Good	No	Fair	C++
Cocos2d-x	Good	Yes	Fair	C++, Lua, JavaScript
XNA	Fair	No	Gradual	C#
Unreal, CryEngine	Excellent	Yes	Steep	C++, C#, UnrealScript
Unity	Good	Yes	Fair	C#, Boo, JavaScript

encapsulates basic functionalities, which hugely increases developers' efficiency. The interactive editor, however, is still missing for most of the open source game engines like Ogre. Another shortcoming of Ogre is that it is only a graphics library, which lacks many other important functionalities, like physics, AI, GUI, audio.

- Cocos2d-x is also an open source game engine. Unlike others, it has its own IDE, which was released on March 2014.

- XNA is a free proprietary game engine provided by Microsoft. Due to its simplicity and light weight, it is very popular among beginners. But it is not as feature-rich as other game engines. One of the most important missing component is object management. Interactive editor is not provided as well. Moreover, no new versions have been released since 2010.

- Unity is a proprietary game engine, released under both free and commercial licenses. It has been very popular especially in the field of mobile game and indie game development. It is a game development solution rather than a graphics middleware, as it supports lots of common functionalities adequate for small and medium game development. Unity also provides an interactive IDE, which allows developers

to not only create scenes, animations interactively, but also to debug program at runtime. It supports C# as the scripting language, which is very useful for rapid development. But this trade-off decreases its runtime performance. The cost of the interpolation between managed and native code is quite high.

- Unreal Engine, and CryEngine are two of the most successful commercial game engines that target high-end games. They support lots of advanced rendering techniques and provide a full solution of game development. UDK and CryEngine Free are their free versions respectively. But both of them have a steep learning curve. Beginners need to have a clear understanding of the two large frameworks.

Taking into account development efficiency, runtime performance and supported functionalities, we excluded the options except Unity and Cocos2d-x. After the overall consideration of popularity, support, and developer community, we finally chose Unity as our implementation platform.

4.2 Other Libraries

OpenCVSharp is used as the image processing library in this project. It is a .Net wrapper of OpenCV, one the most popular libraries in the field of computer vision.

OpenCV is cross-platform, free for use, and open-source under BSD license. It supports lots of common image processing algorithms, for example colour space conversion, noise generation and smoothing, histogram calculation, edge detection, corner detection, feature detection, etc. It began to support CUDA in 2011.

4.3 Development and Runtime Environment

- CPU: Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz (8 CPUs) 2.3GHz
- GPU: NVIDIA GeForce GTX 660M

- OS: Windows 7 Home Premium SP1
- Programming language: C#, ShaderLab, Cg
- Frameworks and libraries:
  - Unity 4.5.1f3
  - .Net 2.0 Subset
  - OpenCVSharp 2.4.9.20140719 (wrapper of OpenCV 2.4.8 for .Net Framework)
  - Visual Studio Tools for Unity
  - Git 1.8.4 and BitBucket
- Assets: Free Furniture Props, Lenna

Chapter 5

Evaluation

In this chapter, we evaluate the performance of every independent steps and the overall process of style transfer.

5.1 Colour Transfer between Images

5.1.1 Results

Figure 5.1 shows several test results obtained by the colour transfer technique implemented in this project.

5.1.2 Runtime Efficiency

The most time-consuming steps of this process are:

- converting the style and the original image from RGB to CIE-LAB
- calculating the statistics of the style and the original image
- shifting the original image
- converting the result image from CIE-LAB to RGB

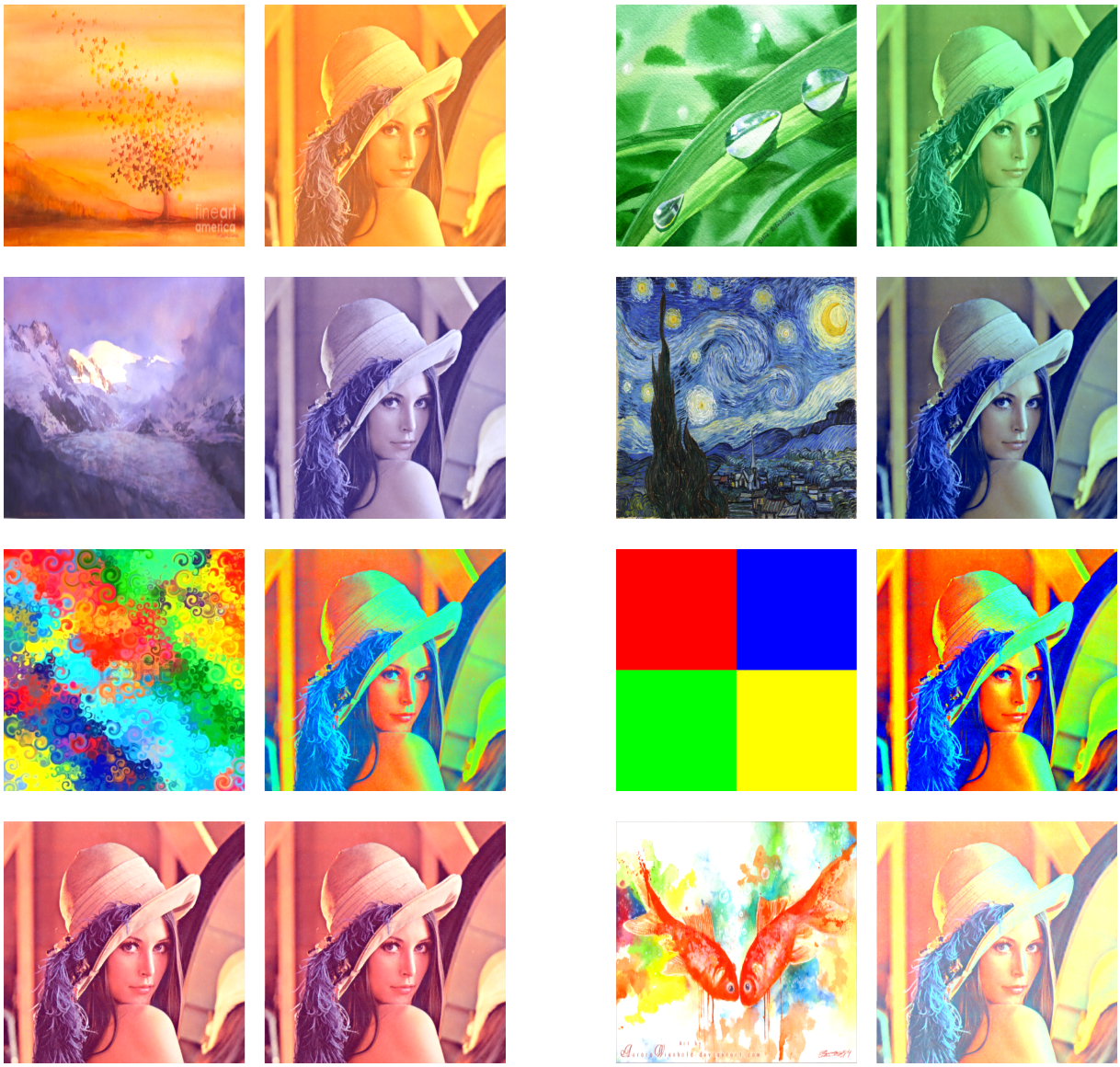


Figure 5.1: Evaluation of colour transfer between images. Odd columns are style images, even columns are colour transfer results. Lenna is used as the original image for every test cases.

All these steps are done by the CPU. Table 5.1 shows the average calculation time in seconds for different resolutions and different steps. The specifications of the testing machine can be found in Section 4.3 Development and Runtime Environment.

We can see from the table that the calculation time is linearly dependent to the resolution of image. Statistics calculation consumes the most time among the four colour transfer steps. Shifting is the least time-consuming step. It takes more time for converting

Table 5.1: The average calculation time (unit: second) of colour transfer for different resolutions and different steps.

Resolution	RGB to CIE-LAB	Statistics	Shifting	CIE-LAB to RGB
512*256	0.085	0.1634	0.0213	0.1071
512*512	0.1560	0.3098	0.0392	0.1835
1024*512	0.3195	0.6210	0.0749	0.3740
1024*1024	0.6403	1.2353	0.1485	0.7728
2048*1024	1.3507	2.6259	0.2560	1.5795
2048*2048	2.7255	5.3816	0.9102	3.3840

CIE-LAB to RGB than the other way around.

For 1024*1024 resolution, we find that the mean calculation costs 0.5399s (43.5%) and the standard deviation costs 0.7016s (56.5%). It seems that the multiplication in calculating standard deviation does not significantly increase the calculation time.

5.2 Watercolour Rendering

Figure 5.2 shows the results of a teddy bear model rendering in the watercolour style implemented in this project. All the five steps of watercolour rendering are done in GPU in real-time and they are designed to focus on efficiency. For the Stanford dragon model, which has 101,256 triangles, the rendering pipeline runs at an average of 497 FPS for 60 seconds. As the comparison, the Phong lighting shader runs at average of 628 FPS, and the diffuse shader runs at 646 FPS.

5.3 Colour Transfer from image to 3D scene

5.3.1 Results

Figure 5.3 shows a test case of colour transfer from image to 3D scene. The style image of this case is created by rendering the 3D scene with diffuse shader. The colours of objects are randomly generated. The final style image is watercolourised using PhotoShop. As we

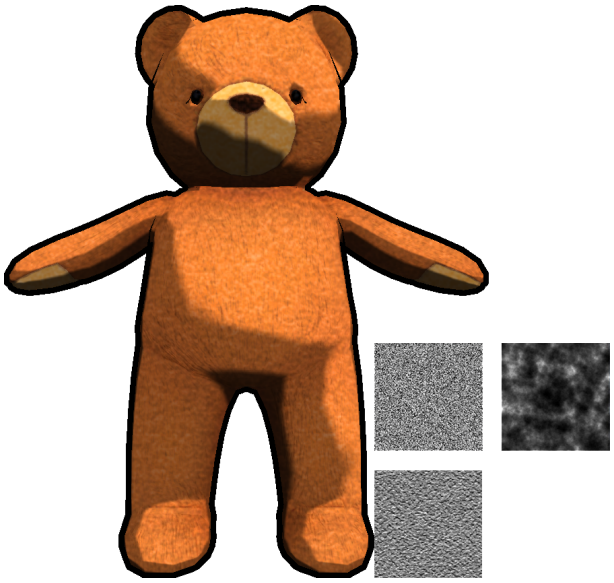


Figure 5.2: Teddy bear model rendered in watercolour style.

can see from the result, most objects successfully find their corresponding colours. The reason of some failure style assignment will be discussed in Section 6.2 Accurate Style Assignment.

5.3.2 Runtime Efficiency

The runtime efficiency of colour transfer from image to 3D scene consists of two parts, 1) style extraction and assignment 2) style rendering.

Style extraction and assignment is pre-processed before rendering. The most time-consuming steps of this process are:

- clustering the style image using k-means
- converting the style image and the object textures from RGB to CIE-LAB
- calculating the statistics of the style image and the object textures
- shifting the object textures
- converting the object textures from CIE-LAB to RGB

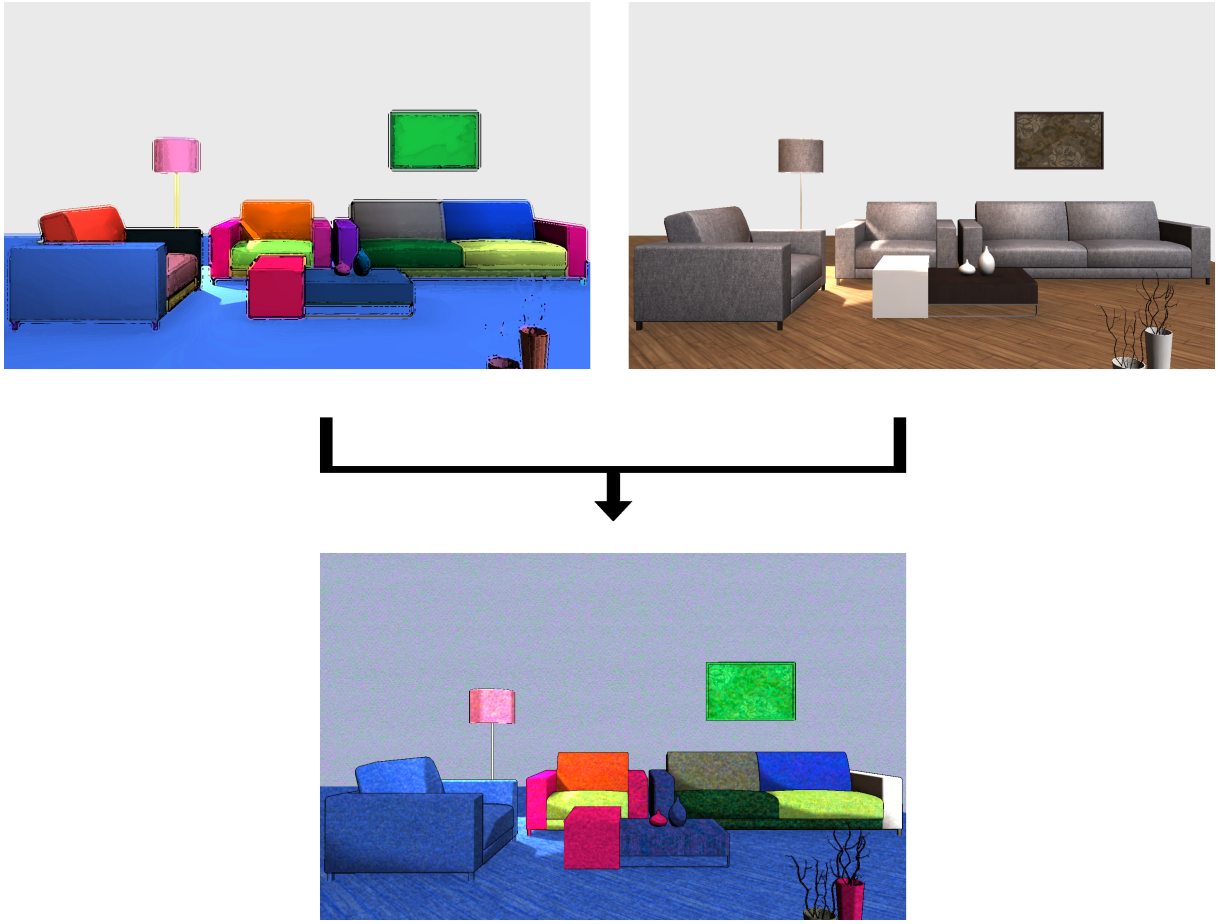


Figure 5.3: Colour transfer from the image of a 3D scene to that scene (rendering in watercolour style).

The efficiency of the last four steps is already discussed in the previous sections. Here we only talk about the efficiency of k-means. As Figure 5.2 shows, the number of attempts significantly influences the runtime efficiency. Once the number of attempts is doubled, the calculation time is almost doubled as well, except when the number of attempts changes from 1 to 2. The reason may be when the number of attempts is small, the major part that costs time lies in the creation of sample matrix instead of the k-means iteration. Maximum iteration and epsilon are two correlated values. When maximum iteration is large enough, epsilon becomes the salient parameter, vice versa. The number of clusters has relatively minor but still notable impact on the efficiency.

The efficiency of style rendering of a scene is similar to that of a model. The key

Table 5.2: The average calculation time (unit: second) of k-means given different parameters. Sample image size is 1024*1024

Number of Clusters	Maximum Iteration	Epsilon	Attempts	Calculation Time
30	100	0.1	4	8.7960
15	100	0.1	4	6.7413
10	100	0.1	4	5.7911
5	100	0.1	4	4.8873
1	100	0.1	4	3.8269
30	100	0.1	4	8.7960
30	50	0.1	4	8.7510
30	10	0.1	4	8.6738
30	1	0.1	4	8.6504
30	100	1	4	8.1751
30	100	0.2	4	8.5927
30	100	0.1	4	8.7960
30	100	0.05	4	9.8669
30	100	0.02	4	11.5927
30	100	0.01	4	13.4533
30	100	0.001	4	32.4842
30	100	0.1	8	15.5912
30	100	0.1	4	9.5641686
30	100	0.1	2	6.6043
30	100	0.1	1	5.2713403

influence is the number of triangles in the 3D models.

Chapter 6

Conclusion and Future Work

This project verified the possibility of applying the colours of a style image onto the textures of 3D objects based on their positions in the image and the scene. Watercolour style can be integrated to render the scene after transferring colours.

6.1 Watercolour Style Transfer

One of the major problems of this style transfer process is that style extraction does not extract style-specific parameters, for example the three pigment density variation textures used in watercolour rendering, which means it can only affect the colour of the final results but not the style of watercolour.

Extracting watercolour-specific textures is different from most previous work in the field of texture synthesis, like feature guided texture synthesis proposed by Xie et al. [25]. Those texture synthesis method can only generate one texture per style image, but generating watercolour-specific texture requires the algorithm to somehow split different pigment effects into different layers and generate one texture per layer. It is hard even for skilled human painters.

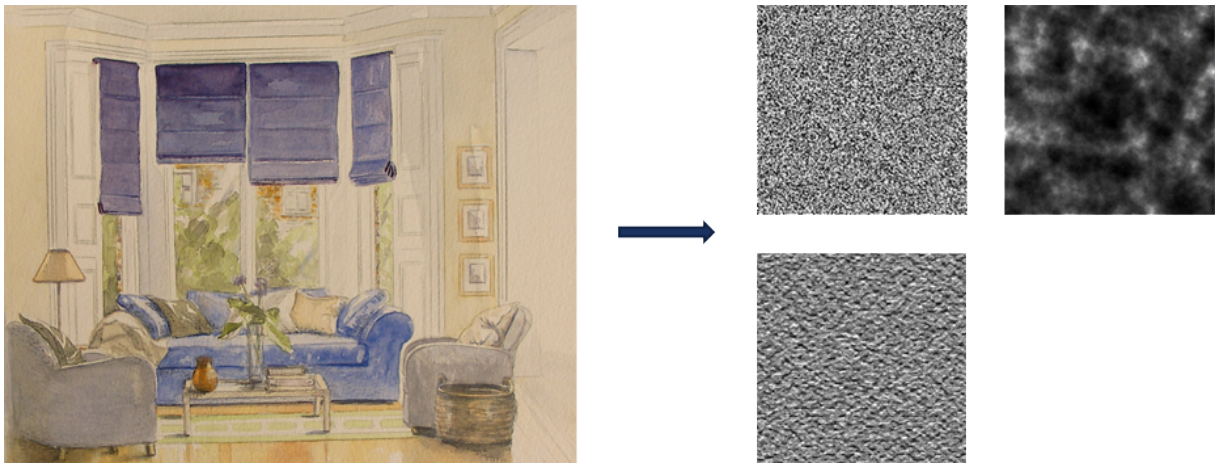


Figure 6.1: An ideal style extraction can extract three pigment density variation textures from the style image.

6.2 Accurate Style Assignment

The current style assignment algorithm has many flaws. Firstly it is not able to handle objects mostly or completely occluded by others. A possible improvement to this flaw is to add another camera and determine object position by taking both cameras into account. This method can handle partially occluded objects to some extents, but is incapable for completely occluded objects.

Secondly, this method requires the object positions in the style image and the scene to be highly correlated. But it is usually not the case. Very few style image and 3D scene are created together. This method can be improved by introducing object shape matching, which links objects in the similar shape even if their positions are very different in the style image and the 3D scene. A simple implementation would be matching the 2D shapes of objects in a static view. An advanced shape matching may take 3D shapes into account, as Ma et al. [26] introduced.

Thirdly, this method cannot handle moving object. As the style assignment is pre-processed before rendering and is done only once, any position or other state change of the object will not affect the assignment result. This requires objects to be static or be located at the right position at the beginning. But sometimes we also want to assign styles

to objects that move into the camera later. If we just simply do the style assignment in every frame, the styles would be incoherent as time goes on.

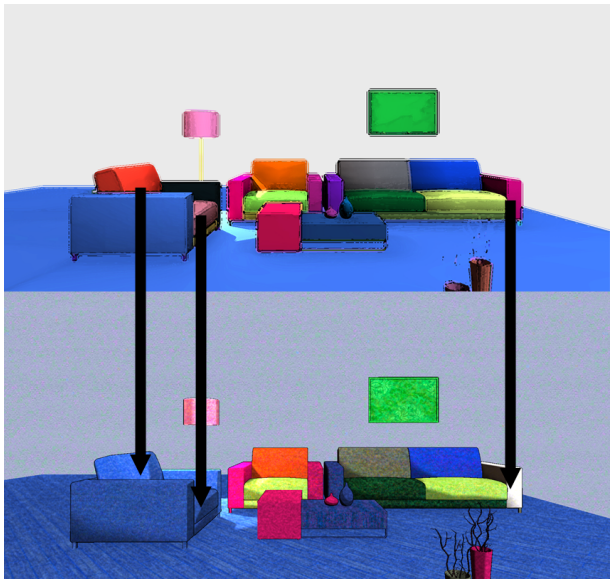


Figure 6.2: The current style assignment algorithm is not able to handle objects mostly or completely occluded by others.

Appendix A

Abbreviations

Short Term	**Expanded Term**
CIE	International Commission on Illumination (French: Commission internationale de l'clairage)
CMY	Cyan-Magenta-Yellow colour model or space
CPU	Central Processing Unit
FPS	Frames Per Second
GPU	Graphics Processing Unit
HSB	Hue-Saturation-Brightness colour model or space
HSL	Hue-Saturation-Lightness colour model or space
HSV	Hue-Saturation-Value colour model or space
IDE	Integrated Development Environment
NPR	Non-Photorealistic Rendering
RGB	Red-Green-Blue colour model or space

Bibliography

[1] T. Strothotte, *Non-photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann, 2002.

[2] E. C. Fernie, *Art history and its methods: a critical anthology*. Phaidon Press Ltd., July 1995.

[3] V. Hlavac, M. Sonka, and R. Boyle, *Image Processing, Analysis, and Machine Vision*. Nelson Engineering, international ed of 4th revised ed edition ed., Jan. 2014.

[4] G. Sharma, W. Wu, and E. N. Dalal, "The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations," *Color Research & Application*, vol. 30, pp. 21–30, Feb. 2005.

[5] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.

[6] H. Steinhaus, "Sur la division des corp materiels en parties," *Bull. Acad. Polon. Sci*, vol. 1, pp. 801–804, 1956.

[7] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theor.*, vol. 28, pp. 129–137, Sept. 2006.

[8] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algo-*

rithms, SODA '07, (Philadelphia, PA, USA), pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[9] E. Reinhard and T. Pouli, "Colour spaces for colour transfer," in *Computational Color Imaging* (R. Schettini, S. Tominaga, and A. Trmeau, eds.), no. 6626 in Lecture Notes in Computer Science, pp. 1–15, Springer Berlin Heidelberg, Jan. 2011.

[10] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley, "Color transfer between images," *IEEE Comput. Graph. Appl.*, vol. 21, pp. 34–41, Sept. 2001.

[11] D. L. Ruderman, T. W. Cronin, and C.-C. Chiao, "Statistics of cone responses to natural images: implications for visual coding," *Journal of the Optical Society of America A*, vol. 15, pp. 2036–2045, Aug. 1998.

[12] G. Li, "Image fusion based on color transfer technique," in *Image Fusion and Its Applications* (Y. Zheng, ed.), InTech, June 2011.

[13] C. H. Nguyen, T. Ritschel, K. Myszkowski, E. Eisemann, and H.-P. Seidel, "3d material style transfer," *Computer Graphics Forum*, vol. 31, pp. 431–438, May 2012.

[14] B. T. Phong, "Illumination for computer generated pictures," *Communications of the ACM*, vol. 18, no. 6, pp. 311–317, 1975.

[15] Q. Yang, S. Wang, and N. Ahuja, "Real-time specular highlight removal using bilateral filtering," in *Computer Vision ECCV 2010* (K. Daniilidis, P. Maragos, and N. Paragios, eds.), no. 6314 in Lecture Notes in Computer Science, pp. 87–100, Springer Berlin Heidelberg, Jan. 2010.

[16] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, (New York, NY, USA), pp. 257–266, ACM, 2002.

[17] A. Lake, C. Marshall, M. Harris, and M. Blackstein, "Stylized rendering techniques for scalable real-time 3d animation," in *Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering*, NPAR '00, (New York, NY, USA), pp. 13–20, ACM, 2000.

[18] A. Hertzmann, "Introduction to 3d non-photorealistic rendering: Silhouettes and outlines," in *Non-Photorealistic Rendering, SIGGRAPH Course Notes*, 1999.

[19] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, and D. H. Salesin, "Computer-generated watercolor," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, (New York, NY, USA), pp. 421–430, ACM Press/Addison-Wesley Publishing Co., 1997.

[20] A. Bousseau, M. Kaplan, J. Thollot, and F. X. Sillion, "Interactive watercolor rendering with temporal coherence and abstraction," in *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, (New York, NY, USA), pp. 141–149, ACM, 2006.

[21] L. Shamir and J. A. Tarakhovsky, "Computer analysis of art," *J. Comput. Cult. Herit.*, vol. 5, pp. 7:1–7:11, Aug. 2012.

[22] L. Shamir, "Evaluation of face datasets as tools for assessing the performance of face recognition methods," *International Journal of Computer Vision*, vol. 79, pp. 225–230, Sept. 2008.

[23] J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1982.

[24] K. Perlin, "An image synthesizer," in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, (New York, NY, USA), pp. 287–296, ACM, 1985.

[25] X. Xie, F. Tian, and H. S. Seah, "Feature guided texture synthesis (FGTS) for artistic style transfer," in *Proceedings of the 2Nd International Conference on Digital Interactive Media in Entertainment and Arts*, DIMEA '07, (New York, NY, USA), pp. 44–49, ACM, 2007.

[26] C. Ma, H. Huang, A. Sheffer, E. Kalogerakis, and R. Wang, "Analogy-driven 3d style transfer," *Computer Graphics Forum*, vol. 33, pp. 175–184, May 2014.