PARTICLE-FLOW ANIMATION FOR PAINTING IMAGES

NUTCHAPHON REWIK

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF SCIENCE PROGRAM IN INFORMATION TECHNOLOGY

FACULTY OF INFORMATION TECNOLOGY

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2/2014

ใบรับรองปริญญานิพนธ์ ประจำปีการศึกษา 2557

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การจำลองภาพเคลื่อนไหวของภาพวาดศิลปะ

PARTICLE-FLOW ANIMATION FOR PAINTING IMAGES

ผู้จัดทำ

1. นายณัชพล เรวิก รหัสนักศึกษา 54070022

..... อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์ ดร.กันต์พงษ์ วรรัตน์ปัญญา)

..... อาจารย์ที่ปรึกษาร่วม

(ผู้ช่วยศาสตราจารย์ ดร. กิติ์สุชาต พสุภา)

Project Title	Particle-Flow Animation for Painting Images
Student	Mr. Nutchaphon Rewik Student ID 54070022
Degree	Bachelor of Science Program in Information Technology
Academic Year	2014
Advisor	Asst. Prof. Dr. Kuntpong Woraratpanya
Co – Advisor	Asst. Prof. Dr. Kitsuchart Pasupa

ABSTRACT

The purpose of this project is to develop a particle-flow animation application. This application creates the animation effect from painting images. Particle-flow animation simulates the colour-flow movement from motionless images. Interactive particles present the rhythm of the brushstrokes. This animation conveys lively image feelings; it beautifies attractiveness, and reminds audiences to original painting images. This project uses ideas of human visual perception that involves with colour. Colour segmentation is a focused technique to extract colour distinction from images. Numerous particles are rendered by modern OpenGL whose techniques are implemented to leverage rendering performance.

หัวข้อโครงงาน	การจำลองการเคลื่อนไหวของภาพวาดศิลปะ
นักศึกษา	นายณัชพล เรวิก รหัสนักศึกษา 54070022
ปริญญา	วิทยาศาสตรบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
ปีการศึกษา	2557
อาจารย์ที่ปรึกษา	ผู้ช่วยศาสตราจารย์ ดร. กันต์พงษ์ วรรัตน์ปัญญา
อาจารย์ที่ปรึกษาร่วม	ผู้ช่วยศาสตราจารย์ ดร. กิติ์สุชาต พสุภา

บทคัดย่อ

โครงงานนี้มีจุดประสงค์เพื่อพัฒนาแอพพลิเคชันที่แสดงภาพเคลื่อนไหวจากภาพวาดศิลปะ ภาพเคลื่อนไหวนี้เกิดจากการจำลองการไหลของสีซึ่งเกิดขึ้นภายในภาพวาดศิลปะที่เป็นภาพนิ่ง โดยการใช้อนุภาคจำนวนมากสร้างทิศทางการเคลื่อนไหวให้คล้ายกับลายพู่กันขณะที่ศิลปินกำลังวาดรูป ภาพเคลื่อนไหวนี้ทำให้ภาพนิ่งที่ไม่มีการเคลื่อนไหวกลายเป็นภาพเคลื่อนไหวที่มีชีวิตชีวา และทำให้ผู้ชมภาพวาดศิลปะสามารถนึกย้อนกลับไปถึงภาพวาดต้นฉบับได้ โดยโครงงานนี้ได้ใช้แนวคิดจากการรับรู้เกี่ยวกับสีของมนุษย์ การแบ่งแยกสีเป็นเทคนิคสำคัญที่ใช้ในการสกัดข้อมูลความแตกต่างของสีที่เกิดในภาพ อนุภาคจำนวนมากถูกแสดงด้วย Modern OpenGL ซึ่งเป็นเทคนิคที่จะช่วยให้การแสดงภาพเคลื่อนไหวมีความรวดเร็ว และมีประสิทธิภาพ

Acknowledgements

This project is a long venture at least half of a year. At first, this venture is just an imagination. I have no idea where the destination is. However, the kind supports and helps from many people assist me to the end of journey. This work would not have been done without these memorable assistants.

I would like to express my gratitude toward Professor Kuntpong Woraratpanya for his guidance and encouragement, which help me in completion of this project.

I sincerely thank all my friends and colleagues, who willingly helped me out with their ability.

I take this opportunity to express gratitude to all of the faculty members for their help and support. I also thank my parents for the unceasing encouragement, care and attention. I am also grateful to people who supported me through this venture.

I also place on record, my sense of gratitude to all, who directly or indirectly, have lent their hand in this work.

Nutchaphon Rewik

Table of Contents

Table of Contents (Continue)

Page

TABLE OF FIGURES

Page

Figure

TABLE OF FIGURES (Continue)

Page

Figure

TABLE OF TABLES

Page

Table

CHAPTER 1

INTRODUCTION

1.1 Research Motivation and Problem

Painting is one of the arts that artists convey emotions and ideas through images. When the artist paints an image, it creates the rhythm of the brushstrokes. However, once they finish their masterpieces, the result is just motionless images. People hardly feel the moment from painting steps.

Recently, Starry Night Interactive Animation is an application on App Store [1]. It makes Starry Night image alive by animating the colour flow. However, it presents only for the specific image; it cannot animate others. To animate arbitrary images, hard coding every image could achieve the task; however, it is a laborious work.

Therefore, we created the animation effect, which simulates the movement of colour from painting images. By using image processing and graphic rendering, we propose an artwork the so-called "Particle-flow Interactive Animation for Painting Image". The principal concept behind this work is a usage of colour segmentation technique. With this technique, the painting can be partitioned into multiple areas of colour shades. This technique is matched to human visual perception, because human recognizes and distinguishes images by colour with their eyes.

After colour areas are extracted, a graphic engine will render many particles to perform the smooth movement defining by steering behaviour, creating intuitive particle interaction.

1.2 Objective

1. To develop painting image particle-flow animation algorithm.

1.3 Proposed Framework

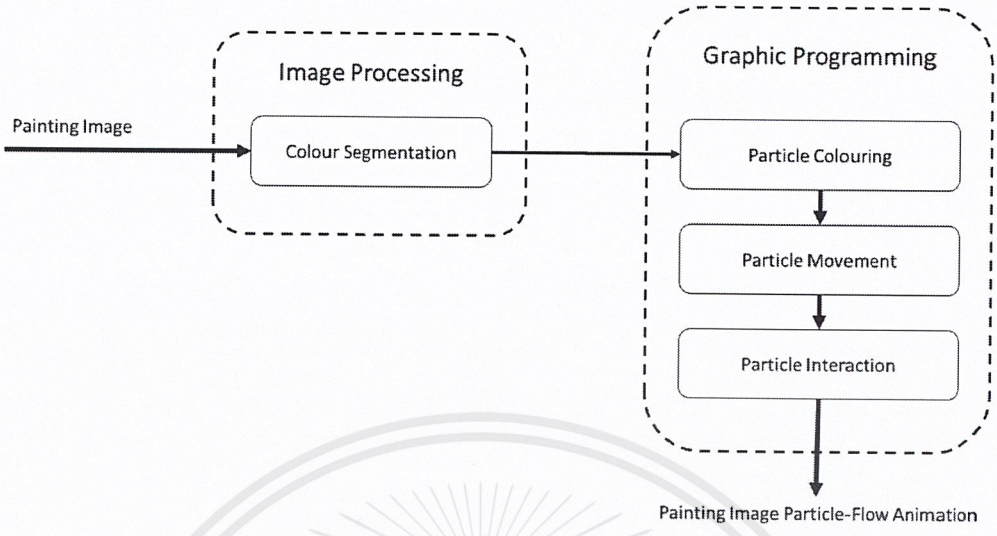


Figure 1.1 Particle-flow Animation Framework

The overview of proposed framework is shown in Figure 1.1. This framework has two main parts, image processing and graphic programming. In the former part, the image processing focuses on extracting information from painting images. Colour segmentation is an important technique, which gathers human visual perception related data. In the latter part, graphic programming creates colour movement. The result from the former part is used for animation rendering. Colour flow and movement techniques are proposed to beautify the animation. The result of the framework is particle-flow animation.

1.4 Scope

1.4.1 This project focuses on painting images. The image is in RGB colour format, and the resolution of the image is not over than 1280 x 720 pixels.

1.5 Benefits

1.5.1 Painting Image Particle-Flow Animation

1.5.2 Able to create the new aspect of painting admiration

CHAPTER 2

THEORITICAL BASIS AND LITERATURE REVIEW

This chapter describes the related theoretical basis composed of various colour spaces, k-means clustering and OpenGL.

2.1 Colour Space

2.1.1 RGB Colour Space

RGB colour space is additive colour space based on the RGB colour model. Normally, digital image uses this space to represent colour. One pixel can be described by three colour channels. The primary channels are red, green and blue. Almost possible colours could be made from the combination of these three channels. The value of each channel is intensity; it defines how intense the luminance of each channel will be shone as one pixel on the screen.

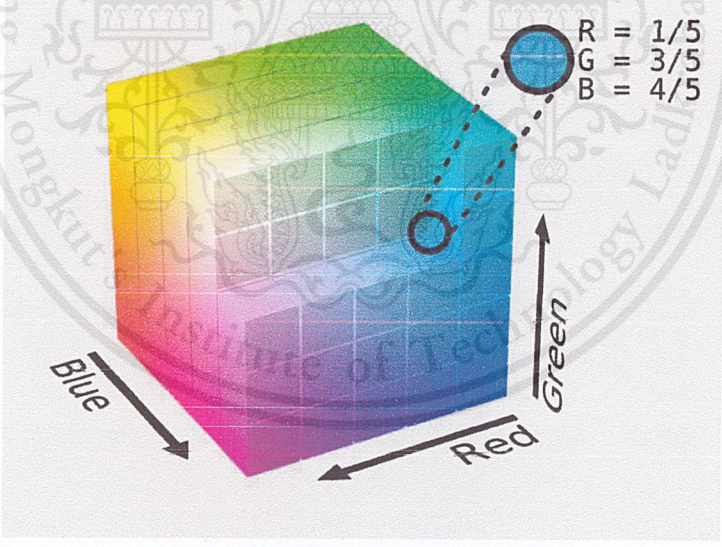


Figure 2.1 RGB Cube

2.1.2 HSV Colour Space

HSV colour space is cylindrical-coordinate representations of points in an RGB colour model. HSV colour space maps colour values into cylinder model. Hue, Saturation and Value are

three values that define colour in HSV. Hue is the angle around central vertical axis of cylinder; it ranges from 0 to 360 degrees. Saturation is the distance from vertical axis of cylinder; it defines colourfulness. The height corresponds to values. These three values give the three schemes the 'H', 'S' and 'V' in their names. HSV is used in colour pickers, image editor software, because it was developed to be perceptually relevant model.

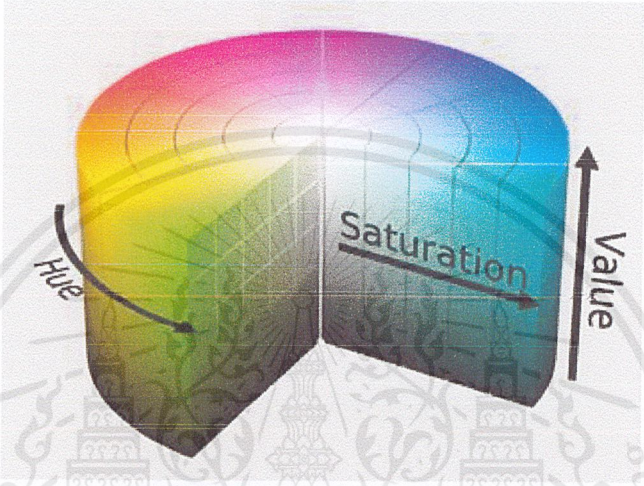


Figure 2.2 HSV Cylinder

2.2 K-Means Clustering

K-means clustering is a method for partitioning n samples into k clusters. It is applied to solve in many problem domains. In image processing, K-means is used in image segmentation. It is used for partitioning digital image into multiple segments. This technique aims to change representation of images into something more meaningful and easier to analyse. The standard algorithm is as follows:

1. Pick k random cluster centre points.
2. Assign each pixel in image to the cluster that minimizes the distance between the pixel and the cluster centre.
3. Re-compute cluster centre by finding an average point from all of pixels in the cluster.
4. Repeat step 2 and 3 until no pixels change clusters.

The distance is absolute difference between a pixel and the cluster centre; colour, intensity or location could be used to calculate this difference.

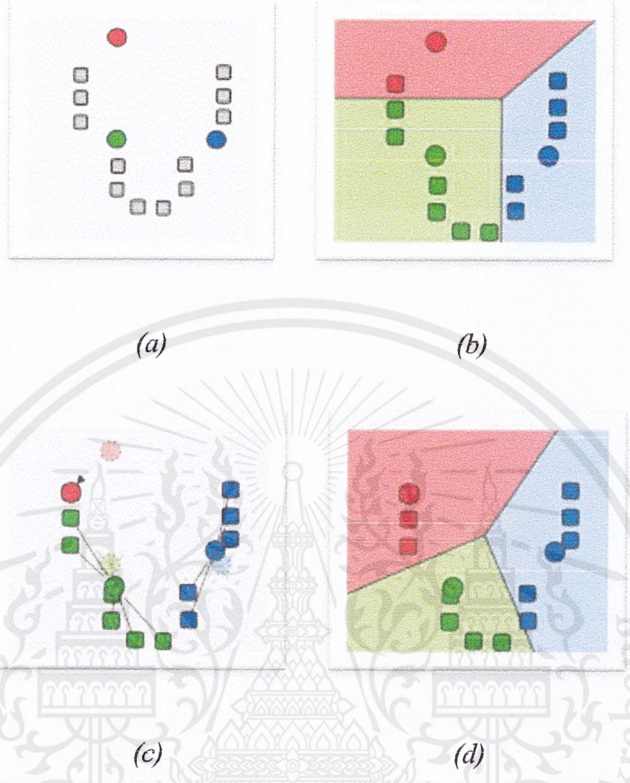


Figure 2.3 Demonstration of K-means standard algorithm

2.3 OpenGL

OpenGL is a graphic library. In this project, OpenGL is used for 2D particles rendering. Vertex Array Object and Vertex Buffer Object are modern OpenGL techniques, which are implemented to render a large number of particles.

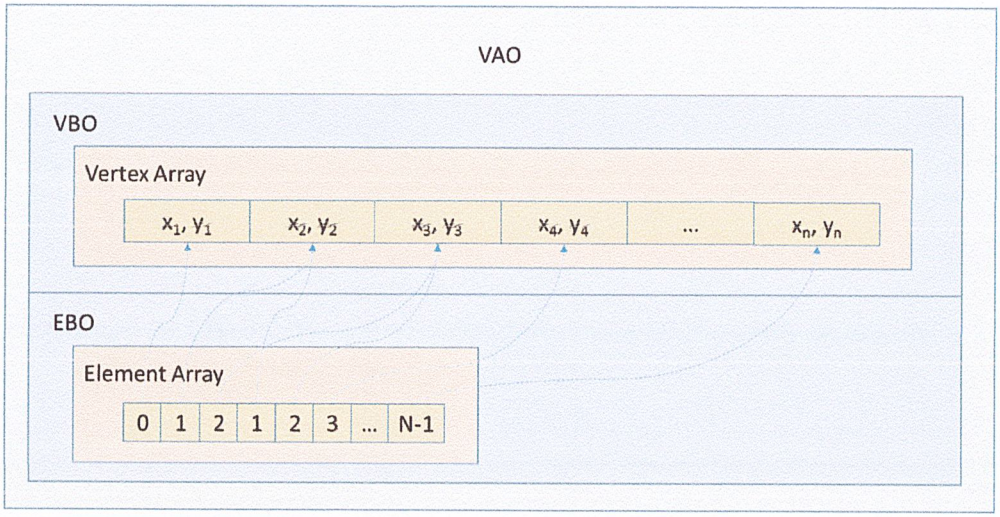


Figure 2.4 VAO, VBO and EBO structure

2.3.1 Vertex Array Object

A Vertex Array Object (VAO) is an OpenGL object. VAO stores all states of an object. In OpenGL, shader program is resided and run on GPU, it requires vertex attributes for rendering. Shader program fetches vertex attribute data from VAO; therefore, vertex attributes formats are stored in VAO. These formats are bound and correlated with vertex array buffer in VBO.

2.3.2 Vertex Buffer Object

A Vertex Buffer Object (VBO) is an OpenGL features for uploading vertex data (e.g. position, colour, etc.) to GPU. It is used for non-immediate-mode rendering. VBO gains higher performance than immediate mode rendering, because vertex data is in GPU buffer, thus it can be rendered directly by GPU.

VBO can be bound to many types of OpenGL buffer. In this project, "VBO" is used for VBO that binds its buffer to vertex array buffer.

2.3.3 Element Buffer Object

VBO binds its buffer data to vertex array buffer. Element Buffer Object (EBO) is a kind of VBO. EBO; however, binds its buffer data to element array buffer.

2.3.3 Vertex Array Buffer

Vertex Array Buffer stores vertex data. It is bound to VBO. Shader program reads vertex data from VAO, which binds vertex attribute to VBO.

2.3.4 Element Array Buffer

Instead of storing vertex data, element array buffer stores indices of elements. These indices refer to position in vertex array buffer binding to the same VAO. Element array buffer reduces the vertex array buffer's size, because rendering might share the same vertices as shown in Figure 2.4.

2.3.5 Shader

Shader is a program running on GPU. The purpose of shader program is to render graphics in programmable rendering pipeline architecture. In this project, OpenGL Shading Language (GLSL) is used to control OpenGL rendering pipeline.

2.3.5.1 OpenGL rendering pipeline

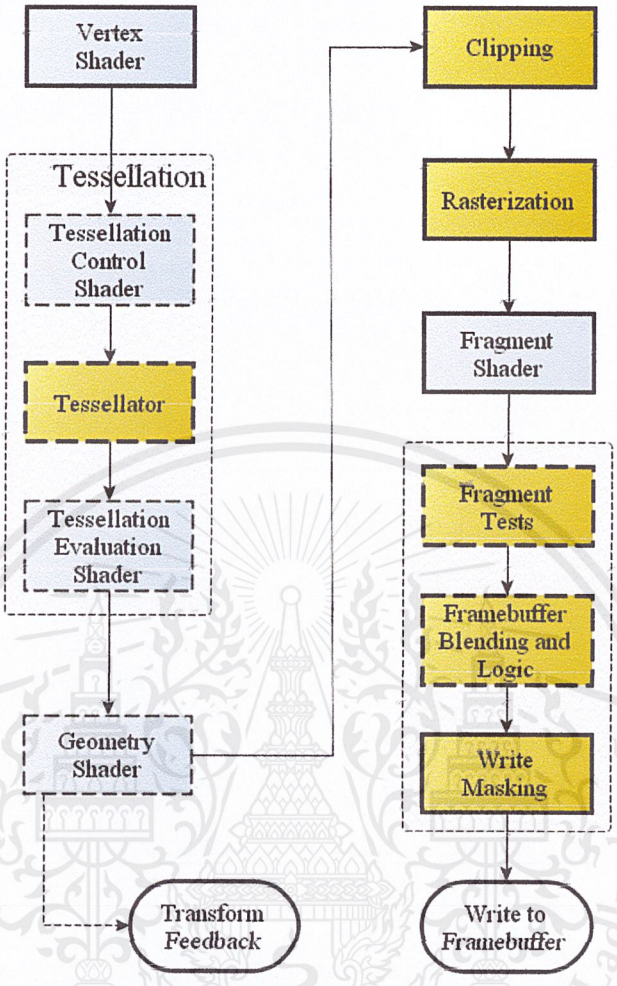


Figure 2.5 OpenGL rendering pipeline

The OpenGL pipeline consists of several stages as shown in Figure 2.5. In this project, Vertex Shader, Geometry Shader and Fragment Shader are two stages, which are used for particle-flow animation rendering.

2.3.5.1.1 Vertex Shader

Vertex Shader processes data per vertex. It receives the attributes input from vertex source (i.e. VBO) and produce output to next stages. This shader stage must produce a position value as an output in order to emit a valid vertex.

2.3.5.1.2 Fragment Shader

Fragment Shader takes a single fragment as an input and produces a single fragment as an output. Colour and depth value can be set by this stage.

2.4 Literature Review

[1] Starry Night Interactive Animation is an application on App Store. This application creates the animation of "Starry Night" painting, which was painted by "Vincent van Gogh". However, this application is specified for Starry Night; it cannot create the same animation effect for arbitrary images.

[2] [3] Tse-Wei C. et al. (2008) studied about image segmentation based on k-means clustering algorithm on HSV colour space. This research converts images from RGB to HSV. It uses k-means to cluster the converted image into multiple colour segments. According to perceptual perspective, this research concluded that the colour segmentation results on HSV colour space are much better than RGB and other colour spaces. This is because values based on HSV are close to human visual perception.

[4] Berg J. V. (2000: 44-45) explained the steps to build a particle system such as fire, smoke and snow. This could be the foundation to design another advanced particle system. It makes particle design to be more dynamic, flexible and maintainable.

[5] [6] Ed A. et al. (2012) explained modern OpenGL programming. This course teaches the overview of OpenGL's structures and architecture. Today, fixed rendering pipeline is deprecated, and it is replaced with programmable rendering pipeline. CPU and GPU efficiently communicate together, reducing bottleneck problem. This is an efficient way to implement particles rendering, these techniques increase real-time rendering performance.

[7] Michael B.D. et al. (1992) published "A general approach to connected-component labelling for arbitrary image representations". This citation proposed labelling techniques using UION-FIND algorithm, which runs in linear time. This algorithm is demonstrated with pixels array scanned method. This algorithm can be applied to digital binary images, since binary images are 2D pixels array.

[8] [9] Fisher, R. explained connected-component labelling by using flood fill algorithm. Four ways neighbouring and eight ways neighbouring are techniques to group connected pixels. Recursive and stack methods are two approaches to implement these techniques. After connected-components are labelled, colour map images are presented to visualize distinct groups of pixels.

[10] Reynolds, C. W. (1999) presented solution for autonomous character in animation and games. Natural movement behaviours are explained by using simple force model. Complex movement patterns can be produced from the combination of simple behaviours, for example, seeking, fleeing and pursuit. This technique yields results in real-time. Today, it has been widely used in industries such animation and game.

CHAPTER 3

RESEARCH

3.1 Proposed Framework

This section will explain the particle-flow animation framework. Figure 3.1 illustrates the overview of proposed framework, which is composed of two main parts; (i) image processing and (ii) graphic programming, which will be explained in the following sections.

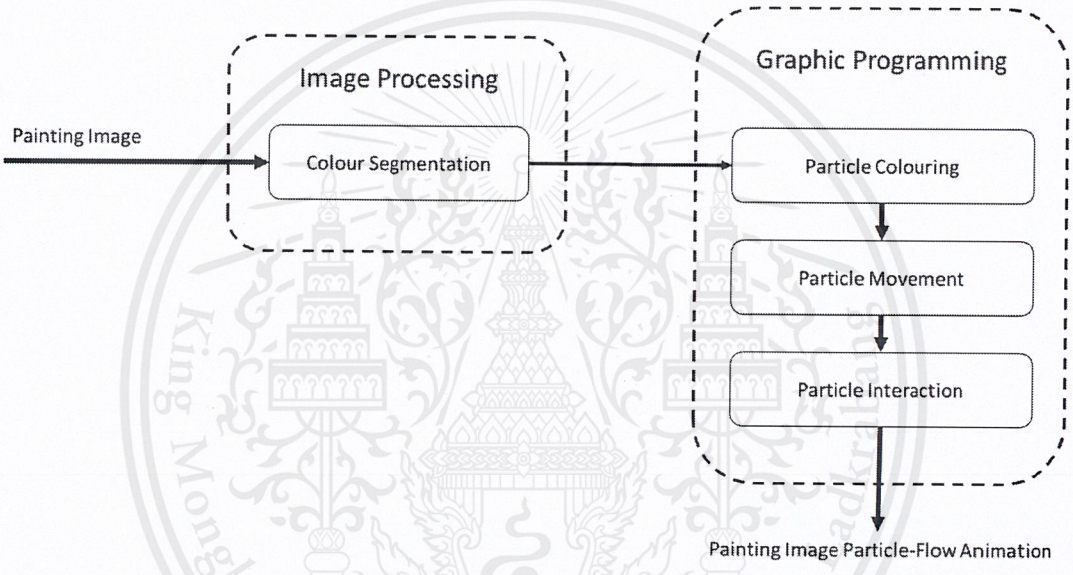


Figure 3.1 Framework of particle-flow animation for painting image

3.2 Image Processing

The purpose of this part is to extract colour area information from painting images. This information will be used in graphic programming part subsequently. Image processing techniques that are used in this part will be described as follows.

3.2.1 Colour Segmentation

Partitioning image by colour is the goal of this technique. The animation needs the colour area information to colour and position particles. Many clustering techniques

could achieve the task; however, k-means is chosen over others, because it is simple and suitable for this project.

By using k-means, contiguous pixels that has similar colour will be clustered into a group. Steps of colour segmentation are briefly stated as follows.

1. **Step A:** Convert a painting image from RGB to HSV colour space
2. **Step B:** Cluster HSV image by k-means clustering algorithm
3. **Step C:** Apply closing morphological transformation to the image. (Optional)
4. **Step D:** Assign colour areas by labelling the contiguous pixels in the same cluster.

(A) RGB-to-HSV Images Conversion

This step converts RGB to HSV images, whose values are more perceptually relevant to human perception. This step gives better colour segmentation results by using k-means clustering [3].

RGB image is in red, green and blue format. These three values range from 0 to 255. In contrast, HSV is in Hue, Saturation and Value format. Hue ranges from 0 to 360. Saturation and Value range from 0 to 1. RGB-to-HSV conversion is defined as.

$$R' = \frac{R}{255} \tag{2.1}$$

$$G' = \frac{G}{255} \tag{2.2}$$

$$B' = \frac{B}{255} \tag{2.3}$$

$$Cmax = \max(R', G', B') \tag{2.4}$$

$$Cmin = \min(R', G', B') \tag{2.5}$$

$$\Delta = Cmax - Cmin \tag{2.6}$$

$$H = \begin{cases} 60° \times \left(\frac{G' - B'}{\Delta} \; mod \; 6\right), Cmax = R' \\ 60° \times \left(\frac{B' - R'}{\Delta} + 2\right), Cmax = G' \\ 60° \times \left(\frac{R' - G'}{\Delta} + 4\right), Cmax = B' \end{cases} \tag{2.7}$$

$$S = \begin{cases} 0 & , \Delta = 0 \\ \frac{\Delta}{Cmax} & , otherwise \end{cases} \tag{2.8}$$

$$V = Cmax \tag{2.9}$$

Unfortunately, OpenCV defines H in the range of 0 to 179, instead of 0-to-255 value; this causes the loss of colour resolution. Therefore, the colour in converted image will be slightly different from the original image.

(B) K-Means Clustering

K-means clustering is chosen, because it is a simple clustering technique and easy to implement. In this step, HSV images are partitioned by k-means. The value of k is set to 2, thus image will be partitioned into two clusters.

In fact, any numbers of partitions could be set; however, a large number of partitions will take a long time to compute in further steps. According to the image

sizes, painting images typically have a large size. Moreover, the greater value of k, the more unwanted tiny colour areas would have at step D. Therefore, k=2 is a reasonable value.

Figure 3.2 Original painting image

Figure 3.3 Clustered image

This step results the clustered image as shown in Figure 3.3. This image contains either 0 or 1; it defines the group of the pixels.

(C) Closing Morphological Transformation

Closing morphological operator smoothens clustered images. This makes adjacency areas to have more similar colour. While k-means algorithm partitions colour areas in step D, multiple groups of small colour areas will be grouped into a large colour area. Consequently, it could significantly reduce complexity of computation, because the animation will have less colour areas to be calculated.

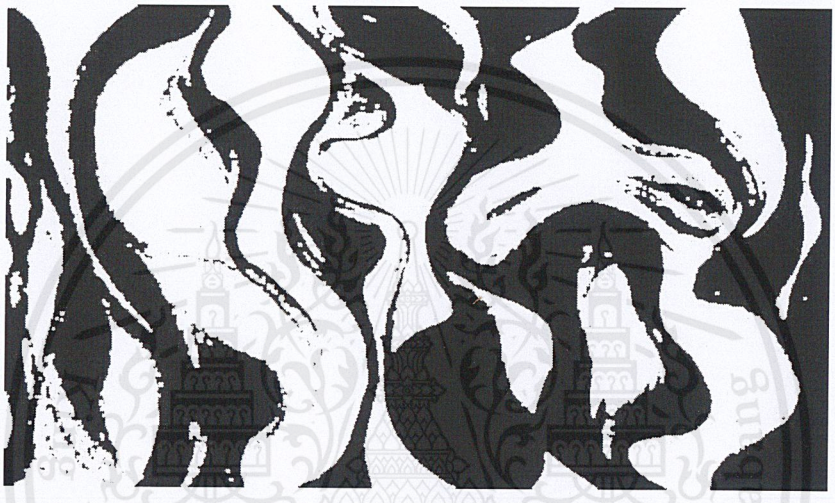


Figure 3. 1 Clustered image after applying closing operation

(D) Colour Area Labelling

The final step is to label colour areas from the clustered images. To label colour area, contiguous pixels in the same cluster will be labelled as the same colour area. The result of this step is the colour-segmented image, which has multiple colour areas. Figure 3.4 illustrates the result of this step.

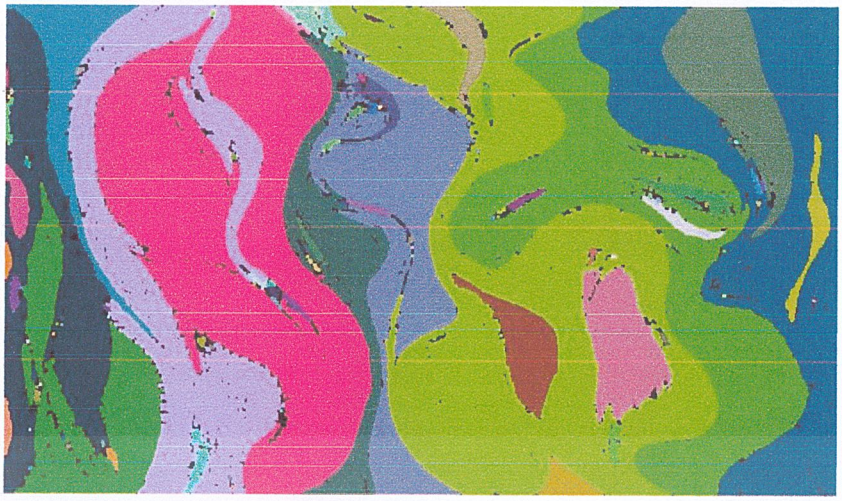


Figure 3.4 Colour Area Image

3.3 Graphic Programming

After colour areas are extracted from the image, graphic engine will be used for animation rendering. Moving particles perform the animation acting. Therefore, this section explains two parts; the former is the particles movement behaviour, and the latter is OpenGL rendering.

3.3.1 Particle

A particle is a small rectangular shape, which flows to perform the animation. A particle could be in any position, colour or speed. Many particles could be vary, thus a particle attributes are defined. These attributes tell graphic engine how a particle should be rendered on the screen. In particle-flow animation, a particle has attributes as follows.

```
Class Particle
{
      //basic attributes
      Vector3 position;
      Vector3 colour;
      float speed;
      float size;
      float rotation;

       //for movement logic
      Float interpolatedPosition
      Emiiter* firstEmitter
      Emitter* targetEmitter

      //for OpenGL rendering
      Matrix4 TransformationMatrix;
}
```

Basic attributes (e.g. position, colour, size, and speed) are obvious in their name, but other attributes will be explained later is this chapter.

3.3.2 Particle Emitter

Emitter is a vertical line in colour area; it emits particles periodically. The emitters will be created after colour areas are extracted; they are separated equally in horizontal space. The emitters fill out entire animation by generating particles repeatedly; therefore, the animation will have less motionless empty gaps, this makes the animation smoother.

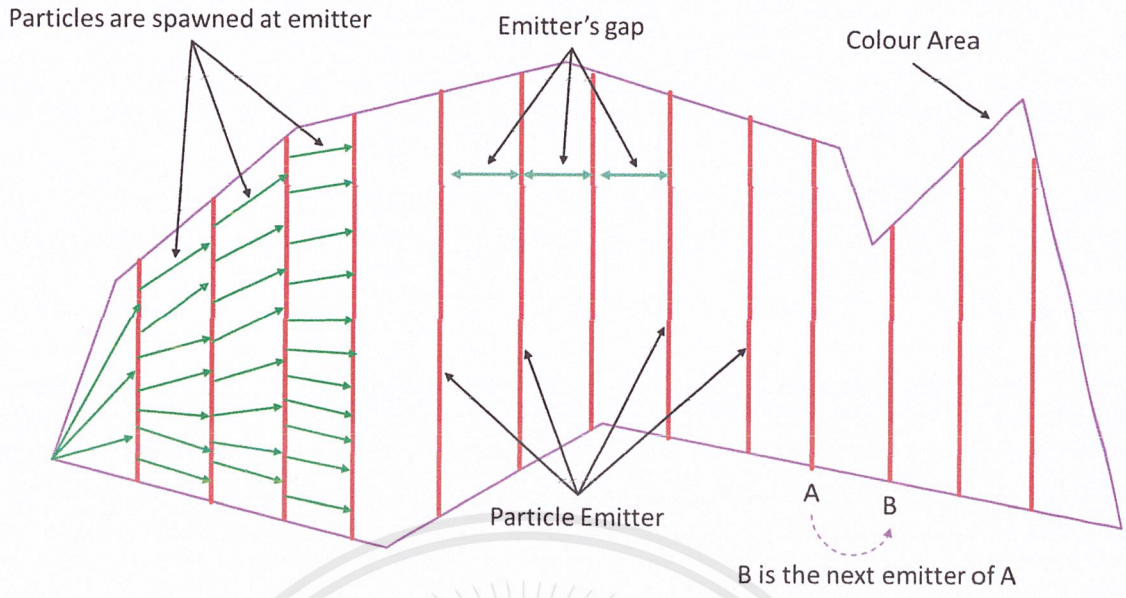


Figure 3.5 Illustration of particle emitters

A Particle has the first emitter and the target emitter attributes. When particle's age is expired, particle will be re-spawned at the first emitter. Target emitter tells particles where they should move next. Particle will move to the target emitter, until there is no target emitter. In this case, particle is outside of colour area, and it will be re-spawned.

3.3.3 Particle Movement

Particle-Flow animation performs by flowing particles. Particle flow means moving particle to follow a path. A particle will move following each point from the starting point to the ending point along a path.

Figure 3.6 A particle flows along a path

To move a particle to somewhere, given a target position where a particle will be moved. Direction vector is a unit vector of target position subtracts by particle position. The next

position of particle is calculated by multiplying the direction vector, particle speed and delta time (i.e. time since last frame) together. The following pseudo code describes the movement of particle to a target position.

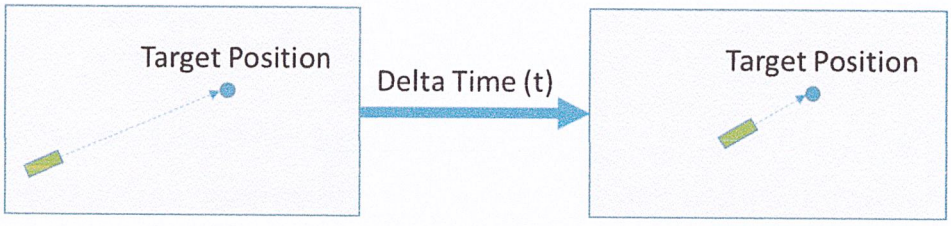


Figure 3.7 A particle movement to target position

```
MoveParticle( Position targetPosition)
{
      Vector2 DirectionVector = (targetPosition -
particle.position).normalized;
      particle.position = DirectionVector * particle.speed * deltaTime;
}
```

A particle needs its path information to flow inside colour area. Since many particles are rendered in the animation, and each particle needs its own path information; creating all their paths is endless work. Therefore, this project implements interpolated position, which will be explain as follows.

Actually, path is a virtual concept to represent the flow line of particles. Instead of creating paths, interpolated position stores and generates virtual path in real-rime. Interpolated position is kept as a particle's attribute. The following figure demonstrates the concept of interpolated position.

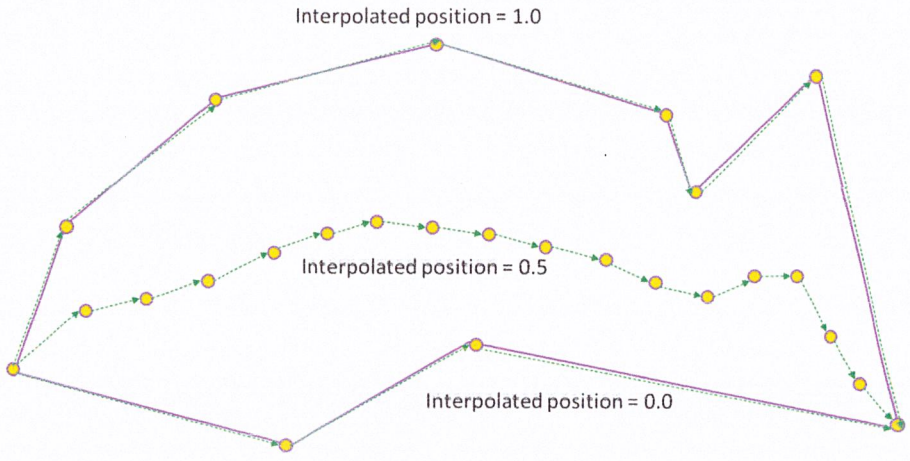


Figure 3.8 Demonstration of interpolated position

Interpolated position is a floating-point value ranged from 0.0 to 1.0. It describes where particle's position should be when they are moving inside colour area. For example, interpolated position equals 0.5; that means particle will be moved horizontally along the centre of colour area, because 0.5 value stores information of centre path. Accordingly, zero is bottom path, and one is top path. Figure 3.8 shows interpolated position in various values.

Particles can be animated by initially setting interpolated position value. After each update call, the actual target position can be calculated from interpolated position and emitter's information. After actual target position is calculated, particle will be moved to that target in turn, the distance depend on particle's speed attribute.

Particle's target will be changed conditionally, when particle moves close to target emitter, target emitter will be changed to next emitter. This results in continuous movement, since particles will be moved from an emitter to another emitter. This condition will occur occasionally, unless there is no next emitter, in this case particle will be re-spawned instead

Particles are short-lived. They will be re-spawned periodically by two conditions. The first condition is age, and the second is position. When particle's age is expired, or particle moves outside its colour area; Particle will be re-spawned immediately. The following pseudo code explains a particle's life cycle and particle movement.

```
//call only once at start
Start(){
	particle.interpolatedPosition = randomValueBetween(0.00,1.00);
}

//call frequently
Update( float deltaTime)
{
	Vec3 targetPosition = getActualPosition(
particle.interpolatedPosition ,
	particle.targetEmitter)
	float distance = Distance( particle.position, targetPosition)
	if( distance < distanceToChangeTarget )
	{
		If( particle.targetEmitter.nextEmitter != null ){
			particle.targetEmitter =
particle.targetEmitter.nextEmitter;
		}Else{
			RepawnParticle( particle )
		}
	}Else{
		moveParticle();
	}
}

RepawnParticle(Particle particle)
{
	particle.position = getActualPosition( particle.interpolatedPosition
					,particle.firstEmitter)
	particle.targetEmitter = particle.firstEmitter.nextEmitter
}
```

3.3.4 Particle System

Particle-Flow animation contains many particles. Depending on images, it could be thousands or even hundreds of thousands particles. Since the number of particles is large, it is difficult to manage each particle directly. Therefore, particle system is implemented to manage particles in the animation. Particle system is shown as the following pseudo-code.

```
int maxParticles
Particle particleBuffer[maxParticles];

Update( deltaTime ){
	for i = 0 to maxParticles{
		particleBuffer[i].Update( deltaTime );
	}
}
```

```
Render(){
	Render_all_particles_in_particleBuffer();
}
```

Following to pseudo-code, particles will be updated by simply iterate all particles, and then update each particle directly. This work is done by CPU; On the other hand, render call is not simple. Rendering involves with vertex data copying and draw call sending to GPU. Since the animation has a large number of particles, render call thus need to manage the number of data copying and draw call sending to GPU. The following pseudo-code differentiates between bad and good particle rendering.

```
//bad render
Render(
{
	//sending drawcall each time for drawing each particle
	for i = 0 to maxParticles{
		Draw( particleBuffer[i] );
	}
}

//good render
Render(
{
	//sending drawcall one time for drawing all particles
	Draw(particleBuffer);
}
```

Accordingly, modern OpenGL features are used in this project. Vertex array object (VAO) and vertex buffer object (VBO) are implemented to minimize draw call. This rendering part will be explained in the following section.

3.3.5 A Particle Model

A particle is a rectangular shape. To draw a rectangle, four vertices data are required as shown in Figure 3.9. In modern OpenGL, a rectangle can be drawn by two triangles as shown in Figure 3.10. The first triangle uses vertex number (0, 1, 2), and the second uses (1, 2, 3).

Figure 3.9 Four vertices rectangle

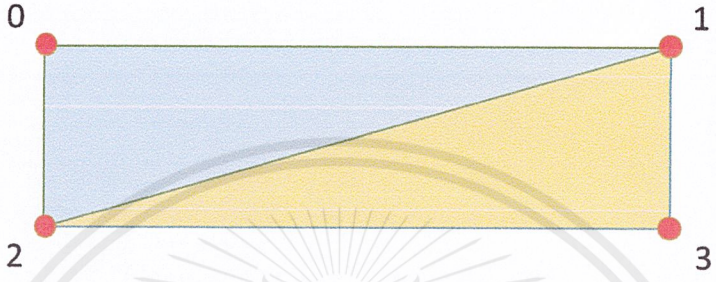


Figure 3.10 Two triangles represent a rectangle in OpenGL

A particle is rendered by six vertices data. This data will be stored in VBO. After sending the draw call of two triangles, a rectangle is drawn on the screen as shown in Figure 3.12.

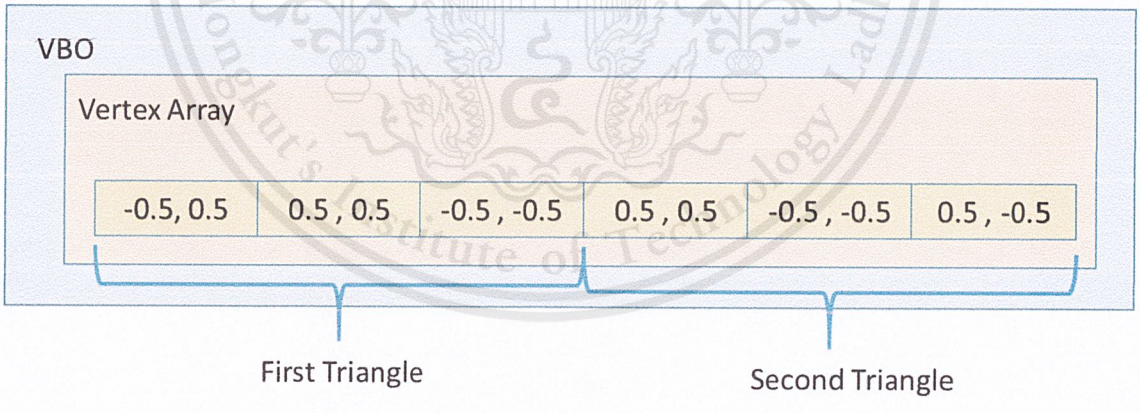


Figure 3.11 Six vertices VBO of a rectangle

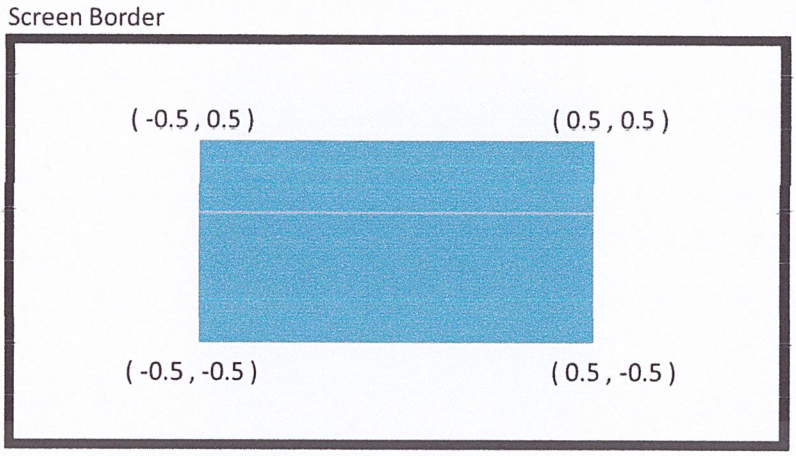


Figure 3.12 A rectangle on screen

A particle needs six vertices data to be drawn, however, the number of vertices data still can be reduced. Since two triangles share the same vertices, which are vertex number 1 and 2; therefore, EBO is used as shared vertices indexing.

EBO reduces the number of shared vertices in vertex array. In this case, EBO for a rectangle contains six indices, and it reduces VBO's vertices data to contain only four instead of six. EBO's structure is shown in Figure 3.13.

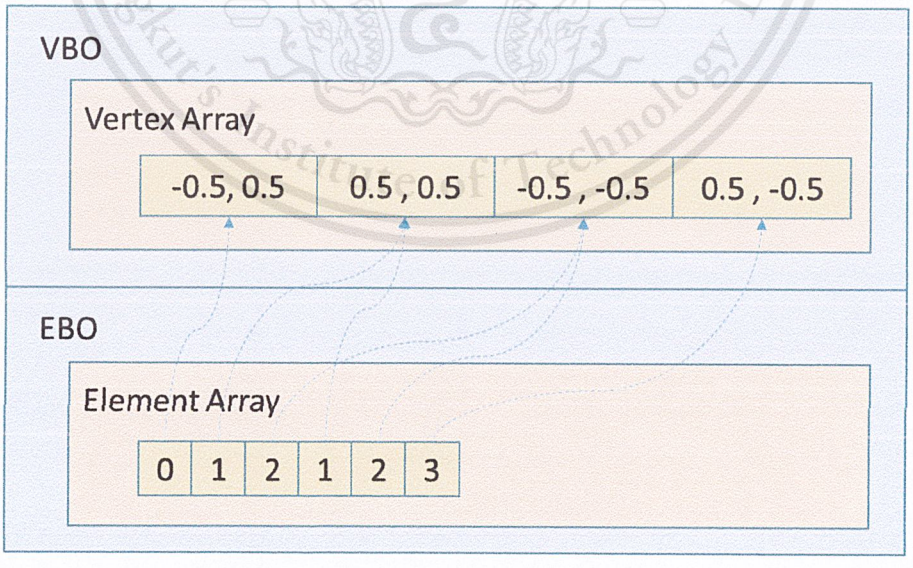


Figure 3.13 VBO's and EBO's data

3.3.6 Particle's transformation matrix

Vertices data represents a particle model. It is, however, a still object. Since only vertices data are given, a particle object will be shown in fixed positions. In animation, a particle needs to be transformed according to its property. To transform a particle, each vertex data could be calculated directly, nevertheless, it is a tired work. Since every vertices data need to be updated each frame, it is a heavy task on CPU. Therefore, transformation matrix is introduced to simplify particle transformation.

$$\begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

Figure 3.14 4x4 Transformation Matrix

Transformation matrix is a 4x4 matrix as shown in Figure 3.14. This matrix is a particle's attribute; it stores three properties, which are position, angle, and size of particle. Since these three properties can be represented in linear transformation. Translation, rotation and scaling represent position, angle and size of particle respectively.

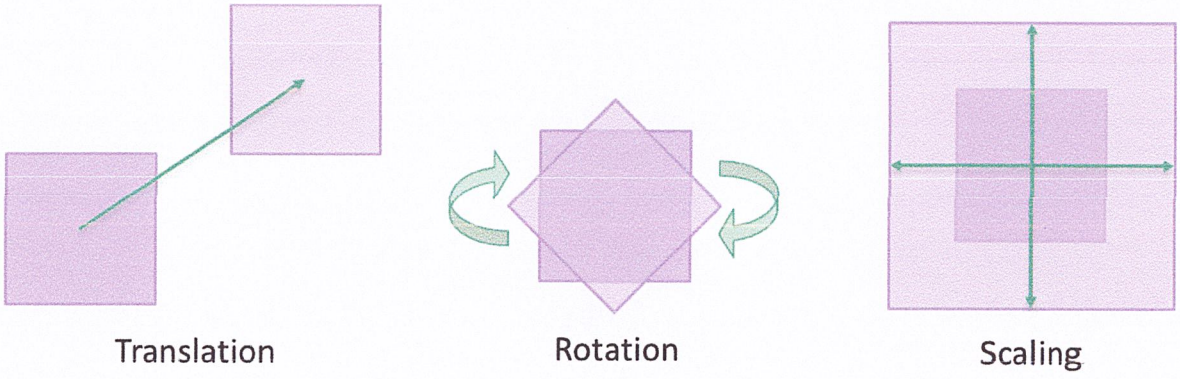


Figure 3.15 Linear transformation

Particle transformation is applied via transformation matrix attribute; it will be manipulated during transformations. After this is done, particle's transformation matrix will be used to calculate actual vertices positions, which will be rendered on screen. This operation is done by multiplying two matrices, which are transformation matrix and particles vertices data. In this operation, particle vertices data is represented in homogenous coordinate, which is a 4-vector (x, y, z, 1).

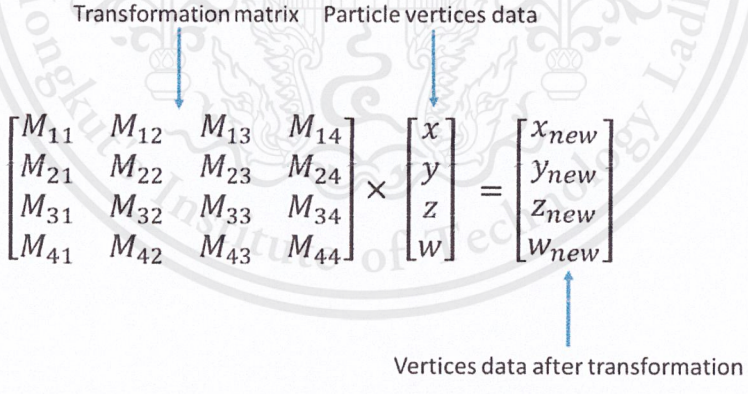


Figure 3.16 Vertices data after applying transformation

However, transformation matrix is calculated on CPU side, it increases rendering performance. Each vertex does not need to be calculated on CPU directly, because transformation matrix dispatches this work to GPU. Moreover, particle vertices data is static; it is an object

model, which it resides on GPU forever. Therefore, only particles' transformation matrix will be copied to GPU every frame, instead of a million vertices position.

3.3.7 Rendering the animation

Previously, a single particle can be rendered by particle model and transformation matrix. This section will explain N particles rendering. There are 3 phases, initialization, update and render, which OpenGL requires these steps to render the animation. The following pseudo code demonstrates all phases, and each phase will be explained further.

```
Init()
{
	//1 vertex and element array memory allocation
	int maxParticle = N;

	float particleVertices[8] = { x1,y1, x2,y1, x2,y2, x1,y2 }; //model
	int particleElements[6] = { 0, 1, 2, 0, 2, 3 };
	Mat4 transformMatrix[N];

	//2 VAO, VBO and EBO initialization
	uint VAO,VBO,EBO,MatrixBuffer;
	GLGenerateBuffers( &VAO );
	GLGenerateBuffers( &VBO );
	GLGenerateBuffers( &EBO );
	GLGenerateBuffers( &MatrixBuffer );


	//3 enable instancing attributes
	GLEnableInstancing(MatrixBuffer);

	//4 VBO, EBO and Transformation Matrix buffer data allocation
	GLCreateNewBuffer( VBO , sizeof(particleVertices) );
	GLCopyDataToGPUBuffer(particleVertices, VBO);

	GLCreateNewBuffer( EBO , sizeof(particleElements) );
	GLCopyDataToGPUBuffer(particleElements, EBO);

	GLCreateNewBuffer(MatrixBuffer, sizeof(transformMatrix) );
	GLCopyDataToGPUBuffer(transformMatrix, MatrixBuffer);
}

Update( deltaTime )
{
	…
	update particles attributes
	…

	For( i=0 to maxParticles)
	{
```

```
            Particle particle = particleBuffer[i];
            transformMatrix[i] = particle.transformationMatrix;
        }
}

Render()
{
        //copy new transformation matrix to GPU and render particles
        GLCopyDataToGPUBuffer(transformMatrix, MatrixBuffer);
        GLDrawElementsInstanced( EBO , maxParticle );
}
```

3.3.7.1 Initialization

```
//1 vertex and element array memory allocation
int maxParticle = N;

float particleVertices[8] = { x1,y1, x2,y1, x2,y2, x1,y2 }; //model
int particleElements[6] = { 0, 1, 2, 0, 2, 3 };
Mat4 transformMatrix[N];
```

First, vertex array and element array memory allocation. Vertex array contains vertices data of a particle model. A particle requires four vertices (rectangle shape), thus the size of vertex array is eight. Element array contains indices of particles. A particle requires six indices, thus the size of element array is six. This model's structure was shown in Figure 3.13

```
//2 VAO, VBO and EBO initialization
uint VAO,VBO,EBO,MatrixBuffer;

GLGenerateVAO( &VAO );
GLGenerateVBO( &VBO );
GLGenerateEBO( &EBO );
GLGenerateBuffers( &MatrixBuffer );
```

The second is OpenGL's objects initialization. This step calls OpenGL function to generate OpenGL's buffer, it returns id of the object that bound to GPU. These IDs are kept, because it will be used for OpenGL's object communication later.

```
//3 enable instancing attributes
GLEnableInstancing(MatrixBuffer);
```

The third is instancing enabling. Instancing is an OpenGL's feature. Instancing allows OpenGL to render multiple copies from the same object. Since every particles share the same single model, instancing improves rendering performance significantly. Instancing is illustrated in the following figure.

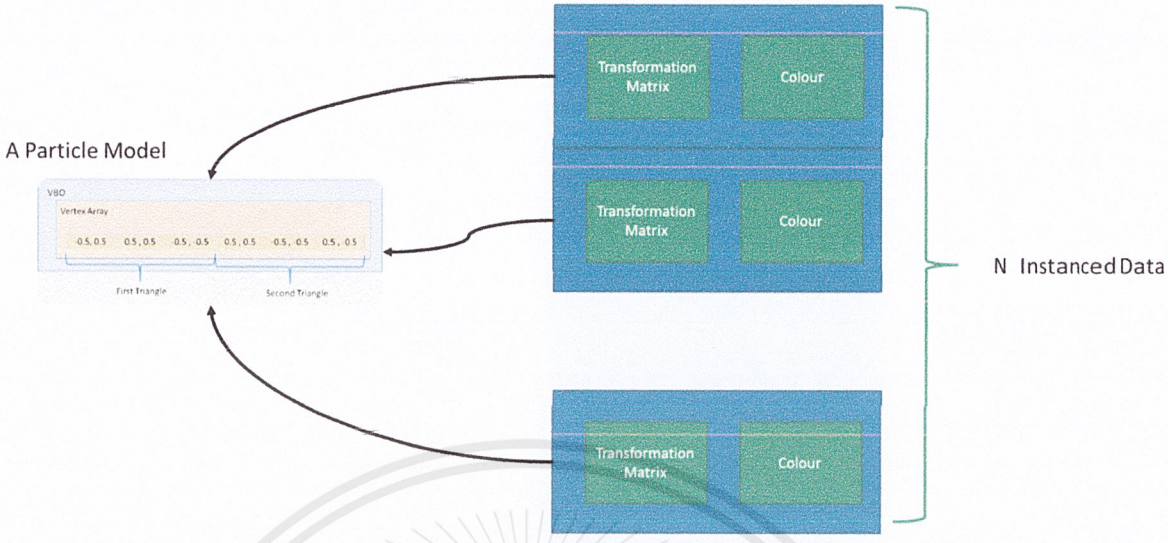


Figure 3.17 Particles rendering with OpenGL's instancing features

However, transformation matrix is a particle's unique attribute; each particle has its own transformation matrix, and it is not shared. Therefore, this attribute needs to be set as the instanced attribute; this allows OpenGL to know the structure of allocated buffer data.

```
//4 VBO, EBO and Transformation Matrix buffer data allocation
GLCreateNewBuffer( VBO , sizeof(particleVertices) );
GLCopyDataToGPUBuffer(particleVertices, VBO);

GLCreateNewBuffer( EBO , sizeof(particleElements) );
GLCopyDataToGPUBuffer(particleElements, EBO);

GLCreateNewBuffer(MatrixBuffer, sizeof(transformMatrix) );
GLCopyDataToGPUBuffer(transformMatrix, MatrixBuffer);
```

Finally, OpenGL's buffers allocation, OpenGL's buffer is a GPU memory buffer; hence, its space needs to be allocated. The buffers' sizes equal to their actual sizes. After these buffers are allocated, the data will reside on GPU forever. However, it is necessary to update some data periodically, and in this case, transformation matrix buffer will be updated before rendering.

3.3.7.2 Update

Update function is called as often as possible, because this function does not involve with GPU. Particles attributes will be recalculated from delta time by particle system. After particles attributes updating has been done, the particle's transformation matrix will be updated in turn.

```
Update( deltaTime )
{
	…
	update particles attributes
	…

	For( i=0 to maxParticles)
	{
		Particle particle = particleBuffer[i];
		transformMatrix[i] = particle.transformationMatrix;
	}
}
```

3.3.7.3 Render

Render function is called when the animation needs to be redrawn, it has two steps

1. Update transformation matrix's buffer by copy newer buffer to GPU.
2. Send the draw call to redraw particles.

```
Render()
{
	//copy new transformation matrix to GPU and render particles
	GLCopyDataToGPUBuffer(transformMatrix, MatrixBuffer);
	GLDrawElementsInstanced( EBO , maxParticle );
}
```

This render function takes only one data copying and one draw call. Once the draw call begins, GPU will render multiple elements from EBO. This technique reduces bottleneck problem between CPU and GPU when the animation has a large number of particles.

CHAPTER 4

RESULTS AND DISCUSSION

Many image processing techniques are used in this project. Most of them mainly focus on colour segmentation. This process contains many steps. Therefore, this chapter will show the experimental results of each step from the original image to the colour-segmented image.

4.1 Colour Segmentation

Colour area is the extracted information. This information controls particle attributes and behaviours. The animation uses colour area as an input information; therefore, the animation quality depends on the colour area information.

In the animation, audiences want to feel the colour movement inside colour areas. If the animation has many tiny colour areas, particles will be moved for a short distance. This does not give the feeling of the brush strokes.

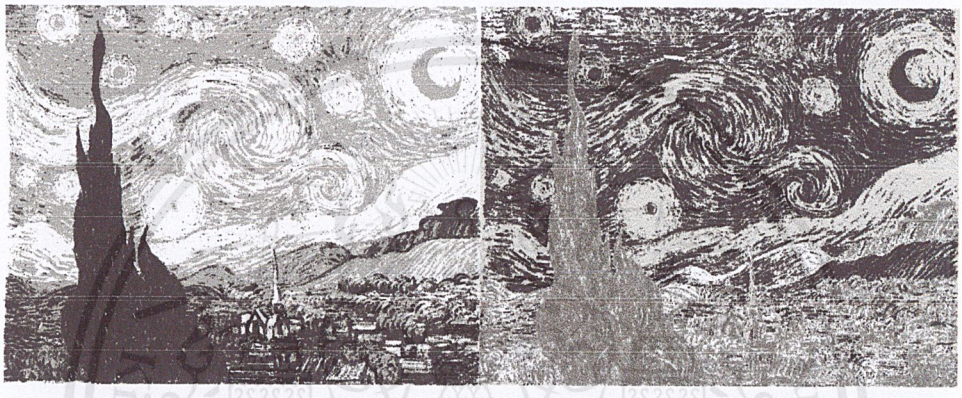
On the other hand, if the animation has only one large colour area, the particle will not be moved like painting image. Therefore, this section shows the comparative result of colour area extracting, and explains the chosen techniques.

4.1.1 k-means Image Clustering Results

By using k-means clustering, the original image will be partitioned into k clusters. In fact, k could be any value greater than 1. However, k=2 is chosen over others. The reason is that it is the minimum value; the original image will be partitioned into two large clusters. This partitioned image will be used next in the colour area extraction, and it will result in large colour areas later.

On the other hand, k>2 will result in multiple small clusters, which they need to be eliminated; if this partitioned image is used in the next process, the colour area will become many unwanted tiny fragments. The following figures show the result of using k-means clustering in various k values to demonstrate the difference between each partitioned image.

(a) Original image *(b) k = 2*



(c) k = 3 *(d) k = 4*

(e) k = 5 *(f) k = 6*

(g) k = 7 *(h) k = 8*

Figure 4.1 k-means results

4.1.2 Morphological Operation Results

Morphological operation processes binary images based on shape. In this project, closing operations are used as image smoothing. After the partitioned image is done, it will consist of 0 and 1 binary number. Closing operation is then applied; the same clusters, which are in adjacent area, will be grouped into a same big cluster.

Closing operation takes two parameters. The first is the partitioned image, and the second is structuring element. Structuring element is a built shape, used to interact with an input image. In this project, adaptive sized ellipse is used as structuring element; the size depends on process iteration. A 5x5 ellipse-structuring element is shown in the following figure.

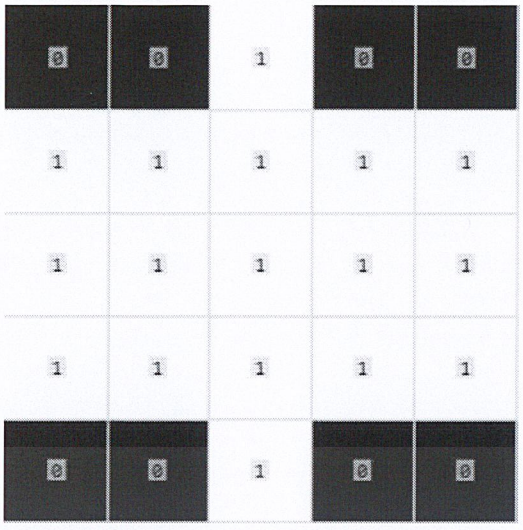
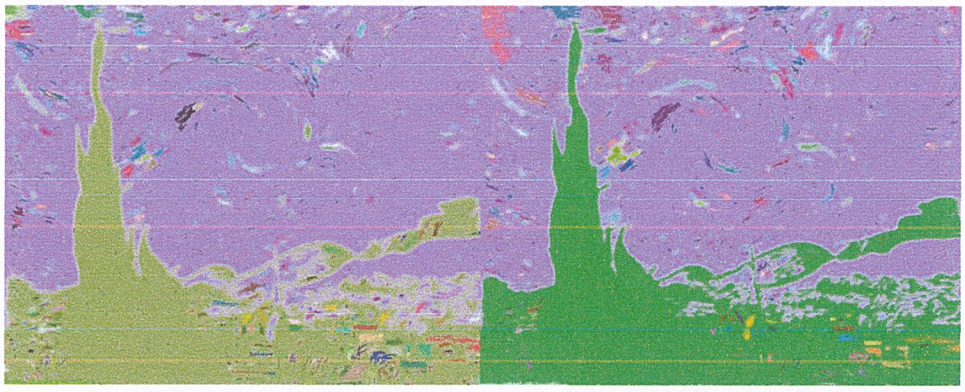


Figure 4.2 5x5 Ellipse-structuring element

Closing operation could be iterated process. This operation could be repeated many times, and it will give a smoother image as a result. Excessive closing operation, however, can blur the image and takes longer time to process. Therefore, the experiment compares closing operation applying in various iterations. The acceptable result will be chosen as a standard iteration of closing morphological operation.

Figure 4.3 Closing Operation

The following figure shows the result of colour area detection after applying closing operation on the starry night image. This result demonstrates the compactness of clusters, after applying closing operation in various iterations.



(a) t=0 *(b) t=1*

(c) t=2 *(d) t=3*

(e) t=4 *(f) t=5*

Figure 4.4 Colour area extraction after applying closing operation result

Following Figure 4.4, (a) t=0 means no closing operation is applied. Many tiny colour areas occur in the colour area image, this results in many artefacts when the animation is played. On the other hand, t=5 means closing operation is applied for five iterations, the colour area

image loses many details of small colour areas. Therefore, the animation will not give the feeling of colour movement in lost details.

To balance the trade-off between least artefacts and least detail loss, one iteration value is chosen to be a standard iteration, because it results in good-clustered compactness, and it preserves the details of small colour areas.

As a result, this operation groups multiple small clusters into a whole cluster. The partitioned image after applying closing operation will be useful later in the next process, because a separated whole cluster will be detected as a single colour area. This operation will directly result in large colour areas extraction.

4.2 Feedback

The feedback was collected by a questionnaire. This questionnaire focuses on aesthetics of the artwork. We displayed this artwork on a 55-inch LCD touchscreen PC during the senior project exhibition held at faculty of information technology, King Mongkut's Institute of Technology Ladkrabang in Thailand. Many visitors played with this artwork; twenty of them agreed to participate in the questionnaire.

The questionnaire consists of five questions; it is adapted from [9] and [10]. The 1-5 Likert -scale is used in this questionnaire, with 1 being strongly disagree, and 5 being strongly agree.

Volunteers were tested on the questionnaire after using the application. The following table shows the result of the test.

Table 4.1 Results of the questionnaire

Questions	Mean ($\bar{x}$)	SD
1. The animation ran smoothly.	3.93	0.70
2. It reminded me to original image.	4.33	0.82
3. I liked the interaction.	3.80	0.77
4. The animation was like a painting image being painted.	3.27	0.88
5. I liked the aesthetics.	3.73	0.88

According to Table 4.1, it can be concluded that Particle-Flow animation reminds people to the original image, because the second question is the most outstanding item with the highest mean($\bar{x} = 4.33$). However, painting moment likeness needs to be improved, because the fourth question has the lowest score ($\bar{x} = 3.27$); moreover, volunteers also made comments as further suggestions, for instances:

- "Particle movement is harsh. It should be softer."
- "Particles movement should be adjusted depending on the direction in images"

CHAPTER 5

CONCLUSIONS

5.1 Research Conclusion

Particle-flow animation is created by image-processing and graphic programming techniques. The result of this project is particle-flow animation. It turns painting images into an interactive animation. This is different from the previous related work [1], which it ties to a particular image.

The particle-flow algorithm, however, significantly relies on input images data. If the image has distinct colour areas, the animation will look fine. In contrast, if the image data has vague colour areas, some artefacts will be occurred in the animation. These artefacts are from colour area detection, because it cannot predict one-hundred-percent precision of artist's intent.

This project is presented as an interactive artwork. The animation awakens motionless images, making more aesthetics; it reminds audiences to the original painting. Particle interaction rejuvenates audiences' concentration. It attracts audiences to admire painting images in the new aspect; Audiences can play with it rather than just watching it. This creates the new dimension of aesthetics, by changing audiences' role to become active participants, instead of passive observers. We hope that this artwork could aid people in Art studying by tempting them with colour-flow movement and interaction.

5.2 Recommendations for Further Research

The particle-flow algorithm heavily relies on image data. This algorithm can be improved by human input. For example, K-means clustering could be adaptive process, which it depends on colour area distinction. In distinct colour areas, k=2 can be used as an default value. However, k>2 could be adapted to vague colour areas. This work could be done by human, for example, specify interested region. The result from human involvement will make a better version, because human perception is used as processing data.

In this project, particle's speed, size, shape are manually set. To make a better animation, the author recommends to change these values depends on the image emotion. For example, colour temperature could be calculated; this could vary the animation feeling.

Particle's movement depends on the colour area information, thus the flow movement is fixed. Sometimes particles have unnatural movement. Defining new movement behaviour could be the good idea, and it will make the animation more aesthetics.

This project implements graphic programming part with OpenGL, the author uses modern OpenGL techniques for particle rendering. The performance is acceptable on the tested machine. However, if more performance needs to be improved, advanced particle rendering techniques could be used. For example, transform feedback in OpenGL. This technique will result better performance, because it dispatches most works to GPU.

Bibliography

[1] artof01 **"Starry Night Interactive Animation"** [online] Available: https://itunes.apple.com/us/app/starry-night-interactive-animation/id511943282. 1994.

[2] T.-W. Chen, Y.-L. Chen and S.-Y. Chien "Fast image segmentation based on K-Means clustering with histograms in HSV color space" **Multimedia Signal Processing, IEEE 10th Workshop,** Cairns, 2008.

[3] A. Irani and B. Belaton "A K-means Based Generic Segmentation System" **Computer Graphics, Imaging and Visualization, 2009. CGIV '09. Sixth International Conference,** Tianjin, 2009.

[4] B. J. V. **"Building an Advanced Particle System"** Reading: Game Developer Magazine, pp. 44-50, 2000.

[5] A. Edward and S. Dave "Introduction to modern OpenGL programming" **SIGGRAPH '12 ACM SIGGRAPH 2012 Courses,** 2012.

[6] A. Edward and S. Dave "An introduction to shader-based OpenGL programming" **SIGGRAPH '09 ACM SIGGRAPH 2009 Courses,** 2009.

[7] M. B. Dillencourt, H. Samet and M. Tamminen "A general approach to connected-component labeling for arbitrary image representations" **Journal of the ACM,** vol. 39, no. 2, pp. 253-280, 1992.

[8] L. Vandevenne **"Lode's Computer Graphics Tutorial"** Available: http://lodev.org/cgtutor/floodfill.html. 2014.

[9] R. Fisher, S. Perkins, A. Walker and E. Wolfart. **"Connected Components Labeling"** Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm. 2014.

[10] C. W. Reynolds "Steering Behaviors For Autonomous Characters" **Game Developers Conference,** California, 1999.

Bibliography (Continue)

[11] S. Tirakoat "Application of 3D Cartoon Animation to Thai Plays for Thai Youth: Case study of North Eastern Region of Thailand" **International Journal of Innovation, Management and Technology,** vol. 2, 2011.

[12] Surveymonkey **"Animation Questionnaire"** Available: http://www.surveymonkey.com/s/PDYQFJX. 2015.

BIOGRAPHY

Name : Mr. Nutchaphon Rewik

Project Title : Particle-Flow Animation for Painting Images

Major Field : Software Engineering

Biography

Mr. Nutchaphon Rewik was born on August 10, 1992 in Uthaithani, Thailand. He received Bachelor's Degree of Sciences (First-Class Honour Degree) from Faculty of Information Technology, King Mongkut's Institute of Technology Ladkrabang.

After he graduated, he worked as a software engineer. He plans to continue his career in software engineering.

PARTICLE-FLOW ANIMATION FOR PAINTING IMAGES

Nutchaphon Rewik

Faculty of Information Technology, King Mongkut's Institute of Technology Ladkrabang, Bangkok

Email: nrewik@outlook.com

ABSTRACT

The purpose of this project is to develop a particle-flow animation application. This application creates the animation effect from painting images. Particle-flow animation simulates the colour-flow movement from motionless images. Interactive particle present the rhythm of the brushstrokes. This animation conveys lively image feelings; it beautifies attractiveness, and reminds audiences to original painting images. This project uses ideas of human visual perception that involves with colour. Colour segmentation is a focused technique to extract colour distinction from images. Numerous particles are rendered by modern OpenGL whose techniques are implemented to leverage rendering performance.

Index Terms – particle; animation; image segmentation; OpenGL; interactive multimedia artwork; fine arts

1. INTRODUCTION

Painting is one of the arts that artists convey emotions and ideas through images. When the artist paints an image, it creates the rhythm of the brushstrokes. However, once they finish their masterpieces, the result is just motionless images. People hardly feel the moment from painting steps.

Recently, Starry Night Interactive Animation is an application on App Store [1]. It makes Starry Night image alive by animating the colour flow. However, it presents only for the specific image; it cannot animate others. To animate arbitrary images, hard coding every image could achieve the task; however, it is a laborious work.

Therefore, we created the animation effect, which simulates the movement of colour from painting images. By using image processing and graphic rendering, we propose an artwork the so-called "Particle-flow Interactive Animation for Painting Image". The principal concept behind this work is a usage of colour segmentation technique. With this technique, the painting can be partitioned into multiple areas of colour shades. This technique is matched to human visual perception, because human recognizes and distinguishes images by colour with their eyes.

After colour areas are extracted, a graphic engine will render many particles to perform the smooth movement defining by steering behaviour, creating intuitive particle interaction.

2. LITERATURE REVIEW

[1] Starry Night Interactive Animation is an application on App Store. This application creates the animation of "Starry Night" painting, which was painted by "Vincent van Gogh". However, this application is specified for Starry Night; it cannot create the same animation effect for arbitrary images.

[2] [3] Tse-Wei C. et al. (2008) studied about image segmentation based on k-means clustering algorithm on HSV colour space. This research converts images from RGB to HSV. It uses k-means to cluster the converted image into multiple colour segments. According to perceptual perspective, this research concluded that the colour segmentation results on HSV colour space are much better than

RGB and other colour spaces. This is because values based on HSV are close to human visual perception.

[4] Berg J. V. (2000: 44-45) explained the steps to build a particle system such as fire, smoke and snow. This could be the foundation to design another advanced particle system. It makes particle design to be more dynamic, flexible and maintainable.

[5] [6] Ed A. et al. (2012) explained modern OpenGL programming. This course teaches the overview of OpenGL's structures and architecture. Today, fixed rendering pipeline is deprecated, and it is replaced with programmable rendering pipeline. CPU and GPU efficiently communicate together, reducing bottleneck problem. This is an efficient way to implement particles rendering, these techniques increase real-time rendering performance.

[7] Michael B.D. et al. (1992) published "A general approach to connected-component labelling for arbitrary image representations". This citation proposed labelling techniques using UION-FIND algorithm, which runs in linear time. This algorithm is demonstrated with pixels array scanned method. This algorithm can be applied to digital binary images, since binary images are 2D pixels array.

[8] [9] Fisher, R. explained connected-component labelling by using flood fill algorithm. Four ways neighbouring and eight ways neighbouring are techniques to group connected pixels. Recursive and stack methods are two approaches to implement these techniques. After connected-components are labelled, colour map images are presented to visualize distinct groups of pixels.

[10] Reynolds, C. W. (1999) presented solution for autonomous character in animation and games. Natural movement behaviours are explained by using simple force model. Complex movement patterns can be produced from the combination of simple behaviours, for example, seeking, fleeing and pursuit. This technique yields results in real-time. Today, it has been widely used in industries such animation and game.

3. PROPOSED FRAMEWORK

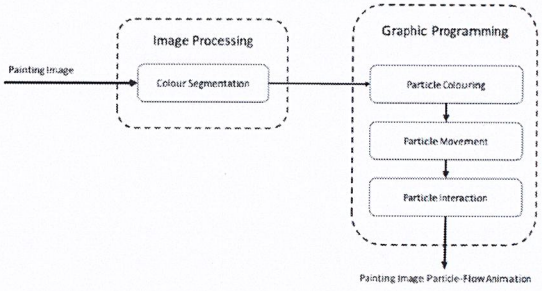


Figure 1. Framework of particle-flow animation for painting image

The proposed framework consists of two main parts; (i) image processing and (ii) graphic programming, which will be explained in the following sections.

3.1. Colour Segmentation

The purpose of this part is to extract colour area information from painting images. This information will be used in graphic programming part subsequently.

Partitioning image by colour is the goal of this technique. The animation needs the colour area information to colour and position particles. Many clustering techniques could achieve the task; however, *k*-means is chosen over others, because it is simple and suitable for this project.

By using *k*-means, contiguous pixels that has similar colour will be clustered into a group. Steps of colour segmentation are briefly stated as follows.

Step 1: Convert a painting image from RGB to HSV colour space.

Step 2: Cluster HSV image by *k*-means clustering algorithm.

Step 3: Apply closing morphological transformation to the image. (Optional)

Step 4: Assign colour areas by labelling the contiguous pixels in the same cluster.

3.1.1 RGB-to-HSV Images Conversion

This step converts RGB to HSV images, since values in HSV are more perceptually relevant to human perception.

3.1.2 k-means Clustering

k-means clustering is chosen, because it is a simple clustering technique and easy to implement. In this step, the value of k is set to 2, the HSV-converted image will be partitioned to a clustered image, which contains two clusters whose values are either 0 or 1. Figure 2.(b) illustrates a clustered image.

3.1.3 Closing Morphological Transformation

Closing morphological operator smoothens clustered images. This makes adjacency areas to have more similar colour. While *k*-means algorithm partitions colour areas in step D, multiple groups of small colour areas will be grouped into a large colour area. Consequently, it could significantly reduce complexity of computation, because the animation will have less colour areas to be calculated.

3.1.4 Colour Area Labelling

The final step is to label colour areas from the clustered images. To label colour area, contiguous pixels in the same cluster will be labelled as the same colour area. The result of this step is the colour-map image, which has multiple colour areas. Figure 2.(c) illustrates the result of this step.

3.2 Graphic Programming

After colour areas are extracted from the image, graphic engine will be used for animation rendering. Moving particles perform the animation acting. Therefore, this section explains about particle movement and structure.

3.2.1 Particle

A particle is a small rectangular shape. The animation contains numerous particles depending on the size of image, and many particles could be vary in their attributes; therefore, we implement particle system to manage particles' states.

(a) Original Image

(b) Clustered Image

(c) Colour Map Image

Figure 2. The illustration of colour segmentation

3.2.2 Emitter

Emitter is a vertical line in colour area; it emits particles periodically. The emitters will be created after colour areas are extracted; they are separated equally in horizontal space. The emitters fill out entire animation by generating particles repeatedly; therefore, the animation will have less motionless empty gaps, this makes the animation smoother.

A Particle has the first emitter and the target emitter attributes. When particle's age is

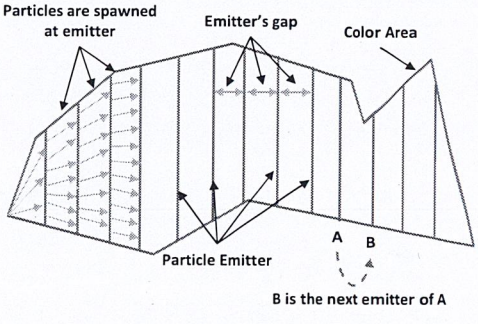


Figure 3. The illustration of particle emitters

expired, particle will be re-spawned at the first emitter. Target emitter tells particles where they should move next. Particle will move to the target emitter, until there is no target emitter. In this case, particle is outside of colour area, and it will be re-spawned.

3.2.3 Particle Movement

Particle-Flow animation performs by flowing particles. Particle-flow means moving particle to follow a path. A particle will move following each point from the starting point to the ending point along a path.

Figure 4. A particle flows along a path

3.2.4 Interpolated Position

Actually, path is a virtual concept to represent the flow line of particles. Instead of creating paths, interpolated position stores and generates virtual path in real-rime.

Interpolated position is kept as a particle's attribute; it is a floating-point value ranged from 0.0 to 1.0. It describes where particle's position should be when they are moving inside colour area. For example, interpolated position equals 0.5; that means particle will be moved horizontally along the centre of colour area, because 0.5 value stores information of centre path. Accordingly, zero is bottom path, and one is top path. Figure 5 shows interpolated position in various values.

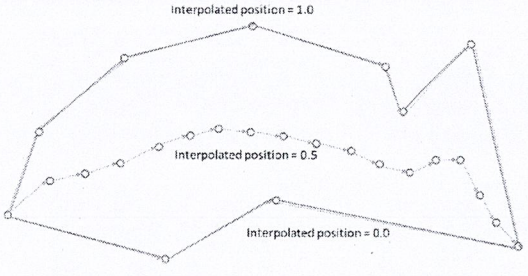


Figure 5. Demonstration of interpolated position

3.2.5 Rendering the animation

The animation will be rendered using OpenGL. A large number of particles will be managed by the particle system. Multiple copies of a particle model will be rendered using OpenGL instancing. Modern OpenGL's buffer memory management is used to reduce the bottleneck problem between CPU and GPU. These techniques increase rendering performance of the animation.

4. RESULTS

4.1 *k*-means Image Clustering Results

By using *k*-means, the original image will be partitioned into k clustered. In fact, k could be any value greater than 1. However, k=2 is chosen over others. The reason is that it is the minimum value. The result from k=2 yields two large clusters in a clustered image, which it will be used next in the colour area extraction, and it will result in large colour areas later.

4.2 Morphological Operation Results

Closing operation could be iterated process, it could be repeated many times, and it will give a smoother image as a result. Excessive closing operation, however, can blur the image and takes longer time to process. Therefore, this experiment compares closing operation applying in various iterations.

When no closing operation is applied, too many unwanted colour areas occur in the colour area image, it results in many artefacts.

On the other hand, applying five iterations of closing operation, the colour area image loses many details of small colour areas. Therefore, the animation will not give the feeling of colour movement in lost details.

To balance the trade-off between least artefacts and least detail loss, one iteration value is chosen to be a standard iteration, because it results in good-clustered compactness, and it preserves the details of small colour areas.

Questions	Mean ($\bar{x}$)	SD
1. The animation ran smoothly.	3.93	0.70
2. It reminded me to original image.	4.33	0.82
3. I liked the interaction.	3.80	0.77
4. The animation was like a painting image being painted.	3.27	0.88
5. I liked the aesthetics.	3.73	0.88

Table 1. Result of the questionnaire score

5. FEEDBACK

The feedback was collected by a questionnaire. This questionnaire focuses on aesthetics of the artwork. We displayed this artwork on a 55-inch LCD touchscreen PC during the senior project exhibition held at faculty of information technology, King Mongkut's Institute of Technology Ladkrabang in Thailand. Many visitors played with this artwork; however, twenty of them agreed to participate in the questionnaire.

The questionnaire consists of five questions; it is adapted from [9] and [10]. The 1-5 Likert-scale is used in this questionnaire, with 1 being strongly disagree, and 5 being strongly agree. Twenty volunteers were tested on the questionnaire after using the application.

According to Table 1, it can be concluded that Particle-Flow animation reminds people to the original image, because the second question is the most outstanding item with the highest mean ($\bar{x} = 4.33$). However, painting moment likeness needs to be improved, because the fourth question has the lowest score ($\bar{x} = 3.27$); moreover, volunteers also made comments as further suggestions, for instances:

- "Particle movement is harsh. It should be softer."
- "Particles movement should be adjusted depending on the direction in images."

5. CONCLUSIONS

Particle-flow animation is created by image-processing and graphic programming techniques. The result of this project is particle-flow animation algorithm. It turns painting images into an interactive animation. This is different from the previous related work [1], which it ties to a particular image.

The particle-flow algorithm, however, significantly relies on input images data. If the image has distinct colour areas, the animation will look fine. In contrast, if the image data has vague colour areas, some artefacts will be occurred in the animation. These artefacts are from colour area detection, because it cannot predict one-hundred-percent precision of artist's intent.

This project is presented as an interactive artwork. The animation awakens motionless images, making more aesthetics; it reminds audiences to the original painting. Particle interaction rejuvenates audiences' concentration. It attracts audiences to admire painting images in the new aspect; Audiences can play with it rather than just watching it. This creates the new dimension of aesthetics, by changing audiences' role to become active participants, instead of passive observers. We hope that this artwork could aid people in Art studying by tempting them with colour-flow movement and interaction.

6. RECOMMENDATIONS FOR FURTHER RESEARCH

The particle-flow algorithm heavily relies on image data. This algorithm can be improved by human input. For example, k-means clustering could be adaptive process, which it depends on colour area distinction. In distinct colour areas, k=2 can be used as an default value. However, k>2 could be adapted to vague colour areas. This work could be done by

human, for example, specify interested region. The result from human involvement will make a better version, because human perception is used as processing data.

The author use modern OpenGL techniques for graphic rendering part. The performance is acceptable on the tested machine. However, if more performance needs to be improved, advanced particle rendering techniques, for instance, transform feedback in OpenGL could be used. This technique dispatches most works to GPU, and it will give better performances.

REFERENCES

[1] artof01, "Starry Night Interactive Animation," artof01, 18 May 2014. [Online]. Available: https://itunes.apple.com/us/app/starry-night-interactive-animation/id511943282. [Accessed 18 May 2014].

[2] T.-W. Chen, Y.-L. Chen and S.-Y. Chien, "Fast image segmentation based on K-Means clustering with histograms in HSV color space," in Multimedia Signal Processing, IEEE 10th Workshop, Cairns, 2008.

[3] A. Irani and B. Belaton, "A K-means Based Generic Segmentation System," in Computer Graphics, Imaging and Visualization, 2009. CGIV '09. Sixth International Conference, Tianjin, 2009.

[4] B. J. V., "Building an Advanced Particle System," Game Developer Magazine, pp. 44-50, 2000.

[5] A. Edward and S. Dave, "Introduction to modern OpenGL programming," in SIGGRAPH '12 ACM SIGGRAPH 2012 Courses, 2012.

[6] A. Edward and S. Dave, "An introduction to shader-based OpenGL programming," in SIGGRAPH '09 ACM SIGGRAPH 2009 Courses, 2009.

[7] M. B. Dillencourt, H. Samet and M. Tamminen, "A general approach to connected-component labeling for arbitrary image representations," Journal of the ACM, vol. 39, no. 2, pp. 253-280, 1992.

[8] L. Vandevenne, "Lode's Computer Graphics Tutorial," 2004. [Online]. Available: http://lodev.org/cgtutor/floodfill.html. [Accessed 3 5 2015].

[9] R. Fisher, S. Perkins, A. Walker and E. Wolfart., "Connected Components Labeling," Hypermedia Image Processing Reference, 2003. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm. [Accessed 03 05 2015].

[10] C. W. Reynolds, "Steering Behaviors For Autonomous Characters," in Game Developers Conference, California, 1999.

[11] S. Tirakoat, "Application of 3D Cartoon Animation to Thai Plays for Thai Youth: Case study of North Eastern Region of Thailand," International Journal of Innovation, Management and Technology, vol. 2, 2011.

[12] "Animation Questionnaire," surveymonkey, 30 April 2015. [Online]. Available: http://www.surveymonkey.com/s/PDYQFJX. [Accessed 30 April 2015].