銘　傳　大　學

資　訊　工　程　學　系

專 題 研 究 終 審 文 件

本校一一四學年度　資訊工程學系

組員：周孝倫

所提專題研究：點構式向量繪圖系統

指 導 教 授 ：王豐緒

中華民國　一一四　年　十一　月　十四　日

致謝

在本專題研究過程中，雖然未有固定指導老師，但仍獲得許多寶貴的建議與指導。在此特別感謝曾抽空審閱我的報告、提供意見的教授們，你們的指導與建議對我研究思路的完善與報告撰寫的順利完成均有重要幫助。在此謹表達誠摯的謝意。

摘要

目前市面上雖已存在各式各樣的向量圖繪圖程式，但幾乎是以點線面架構運作，例如 Adobe Illustrator 便是透過這種架構發展，這種架構將無法支持非典型的形狀，缺乏彈性，難以支持全新結構之物件。且如今大多繪圖工具的複雜性越來越高，獨自學習的門檻也越來越大。為提高初學電繪之學生學習之效率，並提供更加彈性化的開發概念，本研究試圖建立不同於目前主流之點線面架構之向量圖格式，並以使用者操作順暢為宗旨，設計一個架構更加簡單，彈性更加廣泛的物件模型，並使用物件導向技術降低未來增加各式物件種類的成本。

關鍵字：向量圖、可擴充性、人機互動

Summary

Although a variety of vector graphics software are currently available on the market, most still operate based on the traditional “point-line-surface” architecture. For example, Adobe Illustrator is developed using this framework. However, such an architecture has limited support for atypical shapes, lacks flexibility, and struggles to accommodate objects with entirely new structures. In addition, the increasing complexity of existing drawing tools raises the learning threshold for beginners. To enhance the learning efficiency of students new to digital drawing and to provide a more flexible development concept, this study aims to establish a vector graphics format different from the mainstream point-line-surface architecture. The design focuses on smooth user interaction, creating an object model that is simpler in structure and more flexible in application. Moreover, object-oriented techniques are employed to reduce the future costs associated with expanding the variety of object types.

Key words: vector graphics, extendibility, HCI

目錄

圖表目錄

第一章 緒論

此章將敘述促使本研究產生的動機，以及本研究欲解決之問題。

第一節 研究背景與動機

隨著各類繪圖軟體的持續演進，其功能設計逐漸朝向高度整合與多元複雜的方向發展。以目前市面上廣泛使用的向量繪圖工具 Adobe Illustrator 為例，本校商業設計系在標誌設計、視覺傳達等相關課程中均需依賴此軟體進行創作。然而，由於 Adobe Illustrator 具備大量且精細的專業功能，其操作界面與流程對初學者而言存在相當程度的複雜性，往往必須在專業指導之下方能逐步熟悉並有效應用。造成其學習門檻偏高的原因，除功能架構龐大外，亦包括部分操作邏輯並非直覺。例如，使用者需透過按下 Alt 鍵並操作滑鼠滾輪以進行畫面縮放，此方式與多數通用軟體僅透過滑鼠滾輪控制縮放的常見模式不同，致使初次操作者常出現明顯的適應困難。

此外，現行多數繪圖軟體對物件的定義仍侷限於點、線、面等基本幾何構成。當開發者需於既有向量架構中加入全新的物件概念時，通常必須額外建立新的根類別以支撐其功能，這不僅降低了系統架構的彈性，也使擴充與維護成本顯著提高，進而限制創新物件模型的導入與延展。

鑑於上述問題，本研究旨在提出一套以「點」為核心概念、並以精簡化形狀定義為設計原則之圖形結構模型。此架構期望能有效降低初學者的操作門檻，提供更加友善且易於掌握的繪圖環境；同時提升系統於物件擴充、功能延展與效能優化上的彈性，使開發者能以較低成本進行後續維護與創新。透過此研究之建構與驗證，期望能在使用者體驗與系統開發面向上，兼顧易用性與可擴充性，提供現行繪圖軟體架構之外的可行替代方案。

第二節 研究問題

本研究透過日常觀察發現，初次接觸 Adobe Illustrator 的學生普遍出現明顯的適應不良現象，這不僅造成操作上的困難，亦對學習者的自信形成負面影響。本研究推測，造成 Adobe Illustrator 難以上手的主要因素包括：其常規操作方式與一般通用軟體的操作邏輯不完全一致，以及其所具備的功能高度複雜。針對此複雜性，本研究將從使用者與開發者兩個層面分別探討其問題來源與可行的解決方式。

對使用者而言，為確保創作品質，仍需依賴多樣且完整的進階功能；然而，若操作介面與 API 設計未能精簡，將增加學習負擔，降低使用流暢度。因此，對使用端而言，理想的工具設計應能以單一工具支援多種任務，以減少操作切換並提升易用性。反之，從開發者角度來看，若單一工具承載過多職責，將導致系統耦合度過高，不利於後續維護。因此，開發端更傾向令每個工具僅負責單一職能，以確保擴充與維護成本之可控性。

本研究進一步發現，現行主流繪圖軟體普遍無法跳脫「點、線、面」的物件模型框架。此架構雖具備傳統向量圖形處理的優勢，但在面對新的物件概念或功能需求時，往往必須增設額外的根類別，導致整體結構愈趨複雜並降低擴充彈性。為解決此問題，本研究提出一種由「點」為核心、並省略形狀定義的圖形資料架構，以根本性地簡化程式結構、提升物件彈性，使開發者得以在同一物件基礎上建立多樣化的子類功能。藉此，期望能提供一種更輕量、高效率且易於維護的軟體設計模式，並以此架構為基礎開發全新的向量式繪圖應用程式。

本研究所開發之應用程式以 Java Swing 為基礎構建，具備可讀寫自訂檔案格式的功能，並支援符合使用者一般習慣的常見操作，包括全選、存檔、複製、重作、刪除以及放大與縮小等快捷鍵操作，以提升使用者上手速度並降低學習摩擦。為驗證本研究所提出架構在效能、可維護性及擴充彈性方面的實質效益，後續將透過架構理解度分析、模組擴充測試與相容性評估等方法進行全面性檢驗，以確立本研究方法的有效性與可行性。

第二章 概念與文獻探討

本章說明本研究所涉及之理論與文獻。

第一節 向量圖理論

本節說明當今向量圖之概念，介紹向量圖與點陣圖比較之優缺點，和市面上常見之向量圖應用程式，並介紹向量圖於當今的實際應用與功能。

向量圖結構

向量圖（Vector Graphics）是一種以數學函數或向量資料為基礎所構成的圖形形式。相較於點陣圖（Raster Graphics），常見的向量圖檔案格式（如 .svg）僅需儲存結構化的幾何與屬性資訊，因而具有資料量輕量、可無損縮放等特性，特別適合作為大量 3D 物件或複雜場景的描述方式。目前廣泛使用的商用向量圖繪製軟體，例如 Adobe Illustrator，採用付費訂閱模式，並將向量圖形解讀為線段的集合，而線段則進一步由若干控制點構成。控制點依其性質可分為角點（Corner Point）與平滑點（Smooth Point），分別用以形成直線與曲線結構。

在曲線表示方面，多數向量繪圖軟體使用三次貝茲曲線（Cubic Bézier Curve）作為其主要曲線模型，其數學定義如下：

$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$ ，其中$t \in [0,1]$。（Wikipedia, 2025）

此曲線模型以四個控制點P_0, P_1, P_2, P_3描述曲線形狀，具備良好的可控性與平滑度，因而成為圖形處理領域的標準方法。

目前常見的向量圖檔案格式為 Scalable Vector Graphics（SVG），其檔案結構遵循 XML（eXtensible Markup Language），以標記（Tag）的方式逐段描述場景中各物件之屬性。以下提供一個 SVG 檔案片段作為示例：

```
<circle cx="50" cy="50" r="40" stroke="black" fill="red" />
```

此段標記描述一個圓心座標為 (50, 50)、半徑為 40、外框為黑色、填滿為紅色的圓形，語意清晰且具高度可讀性，使使用者能直接從文字結構理解物件內容。

實際應用

向量圖目前廣泛應用於三維（3D）應用程式之中，例如虛擬引擎 Unity3D，其物件外觀通常以頂點向量所組成的集合加以呈現。相較於點陣圖方式，向量式描述能以較少的資訊表達幾何結構，並具備可無損縮放與視覺平滑度高等優勢。此外，向量圖亦常用於商標設計領域。由於商標在各式媒介上需要以不同尺寸呈現，設計者與企業通常不希望放大後產生鋸齒等解析度瑕疵，而商標本身亦較少以寫實效果為主要訴求，因此向量圖特性更能符合商標設計的需求。

然而，向量圖亦存在其限制，特別是在模擬複雜、寫實情境方面較為不利。因此，在多數 3D 應用程式與遊戲引擎中，常採用「向量建構物件形狀」與「點陣圖貼圖呈現細節」的混合模式。於 Unity3D 中，此類附著於向量形狀（Mesh）上的點陣圖被稱為「材質」（Material），其透過紋理貼圖（Texture）補充物件表面質感與細節，以達成兼具結構彈性與視覺寫實度的效果。

第二節 自創之點構式繪圖邏輯概念

本節說明本研究所提出之點構式向量架構的基本組成及基本概念。

物件的組成

不同於傳統向量繪圖中以「點、線、面」作為基本構成要素的概念，點構式向量繪圖邏輯以兩項核心元素組成物件：繪製邏輯、點集合。其中，繪製邏輯負責決定物件之最終外觀與行為，包含拖拽方式、中心點定義、縮放與移動邏輯、各點之間的關係描述、以及物件能否成立的合法條件等功能設計。點集合則作為物件的基礎結構資料，而顏色則提供視覺屬性。

透過此設計，系統不僅相較於既有向量圖模型減少了「線、面」的抽象定義，亦使物件的彈性大幅提升，能更直接地與外部互動。此結構的簡化有助於降低開發複雜度，同時提升物件行為可擴充性，使其能在更高層次上進行功能延展與互動設計。

點的概念

在點構式向量繪圖架構中，點可區分為「動態點」與「靜態點」兩種類型。靜態點的屬性無法由使用者直接修改，其位置與行為由繪製邏輯決定，並相對於物件呈現固定或可預期的狀態，因此稱為「靜態」。靜態點常用於表示物件之中心點、凸包之快取座

標、選取判定依據等功能性資料，可視為輔助物件運作的結構性資訊。
相對地，動態點則為圖形外觀的核心參考，其屬性（如位置）可由使用者直接操作。動態點構成經典幾何形狀（如線段、曲線或其他自訂圖形）之基本框架，因此稱為「動態」。透過動態點的調整，使用者能直接改變圖形外觀，使之成為物件可交互之主要界面。
靜態點與動態點兩項概念的引入，使物件不再侷限於幾何形狀的描述，而能進一步具備與其他物件交互的能力。例如，群組物件（Group Object）即是一種典型應用：群組可透過四個靜態點維護其邊界，並於繪製邏輯中覆寫其操作行為，以達成統一化的功能表現。此機制亦使群組物件與一般圖形在邏輯層面具備一致性，進而提升整體架構之彈性與擴充性。
本研究的系統中，中心點初始定義為所有點集合之重心（Centroid）。若開發者欲新增如橢圓形之物件類別，可藉由配置兩個動態點來控制其長軸與短軸，並搭配兩個靜態點進行輔助運算：其中一靜態點可作為圓心的快取位置，另一則可作為邊界凸包（Convex Hull）處理的快取座標。如此一來，若需使相機或其他物件對準該形狀，只需對齊凸包資訊即可，無須重新計算邊界範圍，從而提升效能並降低運算成本。

繪製邏輯

繪製邏輯可設計成具備交互能力之物件，並且彈性極大，可覆蓋各式行為，這種繪製邏輯更進一步擴張了本邏輯系統之物件概念的彈性，僅需物件類別，即可產生各式物件。

本研究之創新性在於以「點為核心」重構向量繪圖架構，透過靜態點與動態點的分層設計，省略傳統形狀定義所需的中介層，進而提升物件通用性與系統擴充性。

第三節 概念實現之應用程式設計

本節說明基於點構式架構所研製之專案程式，並講解其操作設計、檔案格式協定及其他功能。

操作設計

本研究所開發之應用程式以 Java Swing 為基礎，並採用點構式向量繪圖邏輯進行架構設計，程式命名為 Painter。Painter 提供兩種操作模式：編輯模式與閱覽模式。為提升使用者體驗並降低初次

使用者之學習門檻，本系統整合多項符合一般通用軟體操作習慣的快捷工具，使使用者能以較低的負擔快速熟悉其操作方式。其可用之快捷工具如下所列：

ctrl+C	對選取物件複製
up/right	放大所選取之物件
down/left	縮小所選取之物件
delete	刪除所選取之物件，若無物件選取則刪除最上層圖層
ctrl+Z	重作操作
ctrl+Y	復原操作
ctrl+S	存檔
ctrl+A	全部選取
ctrl+滾輪	視角拉近拉遠
ctrl+滑鼠點擊	選取多重物件
滑鼠左鍵	選取當前最上層之物件或點，或是點空白處取消選取全部物件
滑鼠右鍵	改變所選物件之顏色
檔案托拽	讀取自訂義的檔案格式（.vecf）
視窗左邊	圖層管理器，可滑鼠托拽更蓋物件的圖層順序，可滑鼠托拽邊界改變布局
視窗上面	工具列，可生成出預設圖案，以及群組、解散群組、生成多邊形、多段線、貝茲曲線等，可滑鼠托拽邊界改變布局

Table 1 操作方式列表

自定義的檔案格式協定

本研究針對系統之檔案讀寫需求，設計了一套獨立的專用檔案格式，其副檔名為 .vecf，意指 vector format。在此格式中，檔案第一行用以記錄系統靈敏度參數與相機偏移量，共包含三個浮點數；自第二行起，每一行皆代表一個物件之描述資訊。

對於一般（經典）圖形物件，其資料格式定義如下：

$$[Type_{name}]\ [x_1]\ [y_1]\ [x_2]\ [y_2]\ \dots\ C\ [R]\ [G]\ [B]$$

各項資料以空白字元分隔。其中，$[Type_{name}]$為物件類型之代號，例如圓形以 Cr 表示；$[x_1]\ [y_1]$ ...為構成該物件之點集合；

$C\ [R]\ [G]\ [B]$則以字母 C 作為顏色標示，後街 RGB 三個整數作為色彩資訊。

至於群組物件，其資料格式定義為 $\mathrm{G}{:}\langle \mathrm{Object1}\rangle, \langle \mathrm{Object2}\rangle, \dots$，群組

以 G: 作為開頭，每個子物件以逗號分隔；系統將依據各子物件於格式中宣告的型態進行解析，並依序建立物件後加入群組。
為確保系統之擴充性，本格式要求所有新增物件類別皆需覆寫其專屬之檔案儲存格式、繪製邏輯與合法存在條件，使新物件能正確參與整體架構之輸入、輸出與渲染流程。

其他功能

本專案支援在未預先安裝 Java 環境的 Ubuntu 作業系統、Windows 10 及以上版本作業系統，以及任一具備 Java 8 的作業系統上運行。各平台若使用相同版本之程式，即可共用同一套程式碼，因而大幅提升不同系統間的更新同步性與維護效率。本專案提供兩種程式啟動入口：閱覽模式與一般編輯模式，以滿足不同使用需求。

第四節 文獻回顧

本節將現有之相似向量圖形式與點構式向量結構比對，並分析兩者差異及現有架構之優缺點。

Algorithmic art 介紹

演算法藝術（Algorithmic Art）是一種以演算法生成視覺藝術的創作形式。演算法藝術家有時亦被稱為演算法學家，其作品形式包括數位繪畫、數位雕塑、互動裝置及音樂作品（Wikipedia, 2025）。
不同於傳統向量圖的結構，演算法藝術的基本單元為演算法本身，類似於點構式架構的繪製邏輯。此類方法的優點在於運行效能較高，但缺點則為可讀性與維護性不足，對人類設計者而言，算法生成的結果難以逐點理解或手動調整。
點構式向量架構則藉由「靜態點」與「動態點」的概念，改善了演算法藝術在維護性上的限制。每個點皆為獨立物件，具備明確座標與繪製方法，使設計者能直接查看並理解各點的屬性，進而方便手動調整與管理。
綜上所述，本研究以演算法藝術與向量繪圖理論為基礎，提出點構式繪圖架構。以下章節將進一步說明其設計原理與實作細節。

第三章 可行性分析

本章分析此研究的經濟可行性與技術可行性。

第一節 經濟可行性

本研究之專案開發以 Java 語言為基礎，並充分運用網路上免費取得的工具，以開發出基於點構式向量繪圖系統之應用程式。研究過程中所使用的主要工具包括 Eclipse、VirtualBox、GitHub、HackMD 及 Inno Setup 等，皆為免費資源，無需額外經濟支出。

此外，本研究在統計分析、論證及資料收集方面所採用的方法亦無須任何費用。專案開發環境需使用 Windows 作業系統之電腦，本研究所使用之電腦皆為既有設備，因此無額外經濟負擔。研究中所引用之文獻均為公開可取得之資源，亦無需支付費用。基於上述考量，本研究在經濟可行性方面完全可行。

下文將依序介紹本研究專案開發所需之各項工具及其用途。

eclipse

eclipse 是由 Eclipse Foundation 所經營，是一個常見的 Java 的免費開發集成環境，具備介面化的操作，可省略一些步驟的 cmd 指令操作，例如 jar 檔案輸出，並支援快速除錯功能與外部專案導入編輯。

Virtual box

Virtual box 是由 Oracle 公司經營的虛擬機系統，可透過.iso 檔案模擬不同作業系統的運行，可以用來提供一個安全的隔離操作實驗環境。Virtual box 提供了一個方便的多系統支援管道。由於 macOS 的 EULA（使用者合約）規定不可使用非 mac 裝置使用 macOS，因此本研究不支援 macOS 系統。

Github

Github 是一個專案版本控制的系統，以專案櫃為根基，並支援多人上傳更新，以及隨時恢復舊版的功能，可高效率支援跨平台設計的資源同步，以及過去版本歷史的監控，加以確保程式與過去版本兼容。

第二節 技術可行性

本研究開發所使用的程式語言僅為 Java。Java 語言可在本校資訊工程學系的課程中學習，學生於大二必修的「物件導向技術」及

其他相關課程中即可掌握相關技能。因此，從技術可行性角度來看，本研究完全可行。

第四章　系統分析

此章針對基於研究所開發之專案的架構與效能進行分析，並評估其是否解決本研究提出之問題。其中架構將會使用系統流程圖、樹狀圖與布局圖示之。

第一節 功能分析

此節將針對應用程式系統的整體架構分析。首先，本研究之專案以使用布局圖表示此系統的佈局規劃，然後使用樹狀圖，從類別的角度出發，分析類別與類別之間的關係。再利用系統流程圖，個別展示一般應用程式模式與閱覽模式的執行流程。

使用分布圖

本研究之專案使用 java 自帶的 BorderLayout 對所有主動操作的工具做管理，本研究之專案設計的可主動操作功能說明如下：

1. 工具欄：

工具欄可以增加各種功能之元件，如新增特定形狀、群組、導入點陣圖檔案，吸管吸取顏色等等。

2. 圖層管理器：

實時將畫布中物件的圖層呈現在管理器內，並且可手動拖動元件更改物件的圖層順序。

3. 畫布：

繪製物件的區域，偵測各種輸入事件，提供直覺化的操作介面。

0-1 painter 布局圖

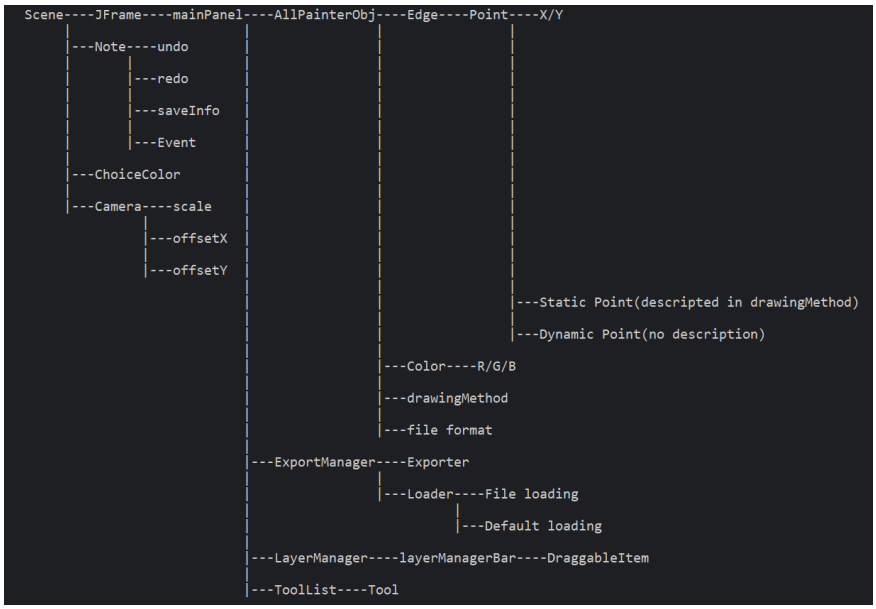
使用樹狀圖

本研究專案的核心環境為 Scene 類別，提供整體操作功能之 API。其內部包含 Note 與 ChoiceColor 類別，以及代表相機狀態的三個變數：scale、offsetX 與 offsetY，分別對應縮放比例及座標偏移。Note 類別提供三個主要 API：saveInfo、redo 與 undo，分別用於儲存狀態、恢復操作與撤銷操作。

介面層 mainPanel 與 ExportLoadSystem、LayerManager 及 ToolList 類別相互關聯。其中，ExportLoadSystem 含有兩個內部類別 Loader 與 Exporter，分別負責資料的輸入與輸出；LayerManager 則包含內部類別 DraggableItem，並以 DraggableItem 列表管理畫布上元件的顯示順序與所屬物件；ToolList 的內部類別 Tool 繼承自 JButton，為本程式原生提供的操作 API，可儲存對應操作功能，並在介面布局中呈現對應圖片或文字。

PainterObj 為所有圖形物件（如群組、圓形等）的根類別。其可儲存點陣列、繪圖流程、操作邏輯、合法存在條件、物件檔案格式以及是否可被拖動等屬性。所有方法之預設流程遵循經典多點控制式形狀設置，同時提供必要的 API 以便擴充。PainterObj 由 Scene 類別管理，依據物件是否可拖動加入相應的物件列表。

此外，PainterObj 擁有歸屬的 Point 與 Color 類別。Color 儲存 RGB 值；Point 則儲存 XY 座標及其所屬物件資訊，使每個點均可精確管理其屬性與繪製邏輯。



0-2 類別關係樹狀圖

使用流程圖

本流程圖將呈現本程式兩個開啟方式：編輯模式以及閱覽模式的流程。

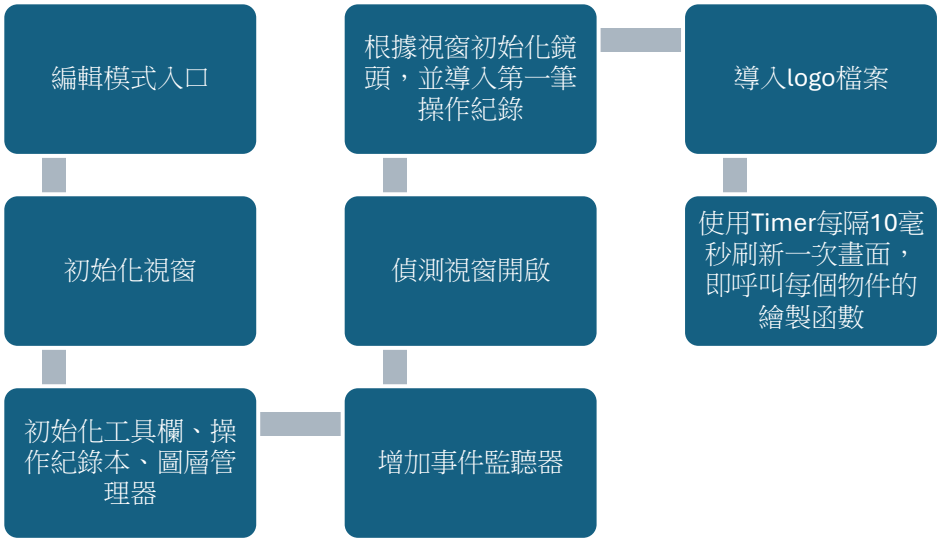


Table 2 編輯模式流程圖

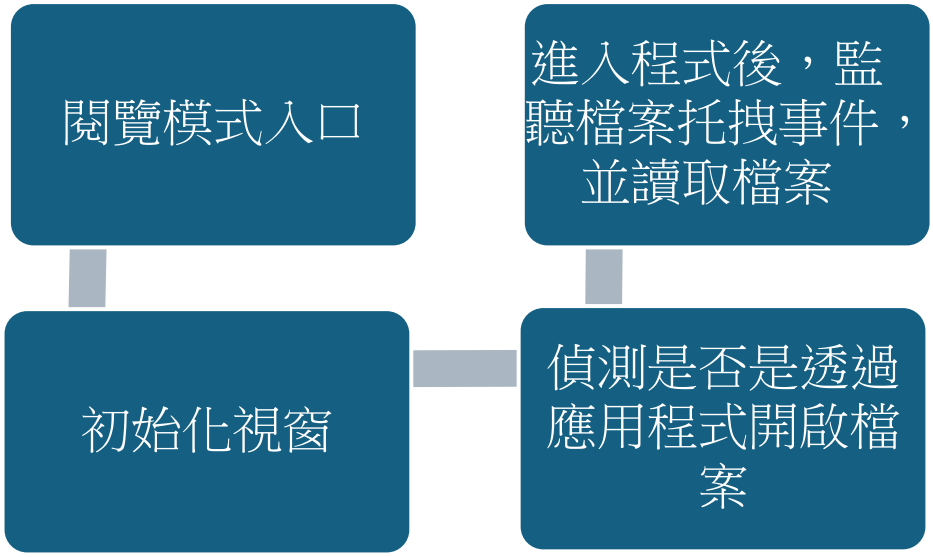


Table 3 閱覽模式流程圖

第二節 專案分析

此節將使用樹狀圖來表示專案資料夾的結構，有助於開發者接手以及統一資料夾功能分類。

資料夾樹狀圖

```
Painter:
    bin:
        javafile:
            -compiled Class File
        testing:
            -painterBrowser.class
            -Main.class
    src:
        javafile:
            -ExportManager.java
            -LayerManager.java
            -Scene.java
            -PainterObj.java
            -ToolList.java
        testing:
            -painterBrowser.java
            -Main.java
    resource:
        -painter_logo.ico
        -painter_logo.png
        -triangle.png
        -circle.png
        -Nedge_BL.png
        -Nedge_BS.png
        -Nedge_SL.png
        -Nedge_SS.png
        -quad.png
        -file.txt
```

0-3 專案架構圖

```
    Assets:
        v1.7.1~v1.10:
            jar files:
                -painter.jar
                -painterBrowser.jar
            Windows:
                -painter.exe
                -painterBrowser.exe
            Ubuntu:
                -painter:
                    bin:
                        -painter
                    lib:
                        app:
                        runtime:
                        libapplauncher.so
                        painter.png
                -painterBrowser:
                    bin:
                        painterBrowser
                    lib:
                        app:
                        runtime:
                        libapplauncher.so
                        painter.png
    -report.docx
    -readme
```

0-4 版本控制

第三節 研究方法有效性評估

本節評估本研究對於開發者擴充功能的成本，分別以架構理解、模組擴充以及相容延展性做評估。

架構理解

本研究以「是否能在一張 A4 圖中清楚呈現資料從點 → 關係 → 形狀 → 最終繪製流程」作為判斷架構易讀性與理解程度的標準。若在流程中需要過多註解、線條交錯、標示多個暫存變數、中間類別或例外處理，即代表架構過於複雜，不利於使用者或開發者理解。

以下將以使用貝茲曲線構成面的程式碼範例，分析在現有架構中新增一個新形狀的流程，藉此評估該架構的易理解性與可擴展性。

```
class BezierSurface extends PainterObj {
    public BezierSurface(Scene scene) {
        super(scene);
    }
    public static BezierSurface INSTANCE(Scene scene) {
        BezierSurface s=new BezierSurface(scene);
        s.addPoint(0,0);
        s.addPoint(0,1);
        s.addPoint(1,1);
        s.addPoint(1,0);
        return s;
    }
    @Override
    public void draw(Graphics g,double scale,double offsetX,double offsetY) {
        if(!islegalObj()) {
        setDrawingColor(g,this);
        Graphics2D g2=(Graphics2D)g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setColor(g.getColor());
        Path2D path=new Path2D.Double();
        path.moveTo(this.getEdge()[0].getX()*scale+offsetX,this.getEdge()[0].getY()*scale+offsetY);
        if(this.getEdge().length>3) {
            for(int i=1;i<this.getEdge().length-2;i++)
            path.curveTo(this.getEdge()[i].getX()*scale+offsetX,this.getEdge()[i].getY()*scale+offsetY,this.getEdge()[i+1].getX()*scale+offsetX,this.getEdge()[i+1].getY()*scale+offsetY,this.
        path.curveTo(this.getEdge()[this.getEdge().length-2].getX()*scale+offsetX,this.getEdge()[this.getEdge().length-2].getY()*scale+offsetY,this.getEdge()[this.getEdge().length-1].getX()*
        }
        else {
            path.quadTo(this.getEdge()[1].getX()*scale+offsetX,this.getEdge()[1].getY()*scale+offsetY,this.getEdge()[2].getX()*scale+offsetX,this.getEdge()[2].getY()*scale+offsetY);
            path.quadTo(this.getEdge()[2].getX()*scale+offsetX,this.getEdge()[2].getY()*scale+offsetY,this.getEdge()[0].getX()*scale+offsetX,this.getEdge()[0].getY()*scale+offsetY);
            path.quadTo(this.getEdge()[0].getX()*scale+offsetX,this.getEdge()[0].getY()*scale+offsetY,this.getEdge()[1].getX()*scale+offsetX,this.getEdge()[1].getY()*scale+offsetY);
        }
        path.closePath();
        g2.fill(path);
        }
    }
    @Override
    public String toString() {
        return "BS "+this.DescriptPoint();
    }
}
```

0-5 根類別 API 描述圖

本程式碼主要包含以下功能：建構子、預設實例生成、繪畫方法及檔案輸出方式。

預設實例生成流程：

流程依序為：宣告物件實例 → 新增點位置 → 回傳完整實例。此流程可快速建立初始形狀並提供後續操作基礎。

繪畫方法流程：

於繪製時，程式首先判斷當前結構是否合法（預設條件為至少三個點）。若合法，系統將取得顏色，並於第一個點位置開始繪製。接著遍歷各個點：若僅有三個點，則以兩段貝茲曲線完成繪製；若點數超過三個，則使用三次貝茲曲線函數生成完整路徑。最終，程式將封閉路徑並填充顏色，完成形狀呈現。

模組擴充

本項引用 Taghreed Riyad 等人於 2021 年發表之論文 Measure extendibility/extensibility quality attribute using object oriented design metric（使用物件導向設計指標來衡量可擴展性品質屬性）所提出之架構擴充性量化方法，將點構式向量架構與點線面架構比較。

該論文所提出之可擴展性公式為 *Extendibility=0.5×ANA-0.5×DCC+0.5×MFA+0.5×NOP*，其中 *ANA* 為抽象指標，計算方式為 $ANA = \frac{\sum_{i=1}^{N} Anc(C_i)}{N}$，其中$Anc(C_i)$代表類別$C_i$的祖先數量，*N* 代表類別總數；*DCC* 為耦合度指標，計算方式為$DDC = \frac{\sum_{i=1}^{n}(A_i+P_i)}{n}$，其中$n$為系統中類別的總數、$A_i$為第 i 個類別中，屬性所引用的其他類別數量、P_i為第 i 個類別中，方法參數所引用的其他類別數量；*MFA* 為繼承度指標，計算方式為$MFA(C_i) = \frac{N_{inh}(C_i)}{N_{inh}(C_i)+N_{own}(C_i)}$，其中$N_{inh}(C_i)$為從父類別繼承的方法數，$N_{own}(C_i)$為該類別自己定義的方法數量，$n$為該類別所有可使用的方法；NOP 為多型指標，計算方式為 $NOP = \frac{\sum_{i=1}^{C} PolyMorphicMethods_i}{C}$，其中$PolyMorphicMethods_i$為第 i 個類別中具有多態行為的方法數，$C$為類別總數。

根據此公式，另$ANA_{pointBased}$為點構式架構之 *ANA* 值、$ANA_{original}$為點線面架構之 *ANA* 值，其餘代數同理，本研究之專案所設計之架構將使系統中除了根類別外，所有類別皆僅繼承根類別，

$ANA_{pointBased}$為$\frac{n-1}{n}$，其中 n 為總類別數，當總類別數越多，

$ANA_{pointBased}$將趨近於 1；本研究之專案，除群組外所有類別之屬性與方法僅在繪製函數時引用一個外部類別（即點類別），當總類別數量越多時，$DCC_{pointBased}$將趨近於 1；本研究專案可延伸出無數新屬性之類別，皆固定繼承 21 個不同的方法，當總數為 n 且平均新屬性數量為 m 時，$MFA_{pointBased}$為$\frac{21}{21+\mathrm{m}}$；本研究之專案每一類別可延伸至多 23 種覆寫方法，且至少須覆寫 3 種方法，可知 $3 \leq NOP_{pointBased} < 23$。

經上述分析，本研究之專案可擴張性為

$\frac{10.5}{21+m} + 0.5 \times NOP_{pointBased}$，其中$3 \leq NOP_{pointBased} < 23$。

點線面架構向量圖系統具備點、線、面三個系統基本類別，其中線引用了一個外部類別（即點類別），面引用了一個外部類別（即線類別），根類別引用了這些基本屬性，延伸的類別皆繼承自根類別，可得$ANA_{pointBased} = ANA_{original}$；若點線面架構之類別所引用之外部類別數量為$K$，則$1 \leq K \leq 3$，可得$1 \leq DDC_{original} \leq 3$；當總數為 n 且平均新屬性數量為 m 時，$MFA_{original}$為$\frac{n}{n+m}$，另點線面架構之根類別實現方式與點構式架構盡可能相似可得出
$MFA_{pointBased} \cong MFA_{original}$ 且 $NOP_{pointBased} \cong NOP_{original}$

從以上計算可得出

$0 \leq Extendibility_{pointBased} - Extendibility_{original} \leq 2 \times \beta$，$\beta = weight\ of\ DCC$。

綜上所述，相同向量圖系統之設計之下，點構式架構相較於點線面架構具備更高的可擴張性。

相容延展

本項評估方式為分析由舊程式版本所做出的檔案是否能被新版本的程式所開啟，這裡將使用應用程式的 logo 檔案作評估。

```
1 33.0 244.51515151515184 125.6060606060606
2 SS 6.128935731861057 -2.5106339882841806 1.0035757318610798 -1.22133398828419 14 8.021623347726925 4.124424462148825  C 0.0 0.5 1.0
3 SS 1.2684296205899734 -1.1793479142478946 1.8803716946834435 1.794850637607544 7.536032221930512 4.290072847679361 8.021623347726925 4.123775021453318  C 0.0 0.4 1.0
4 SS 0.8311074000301133 -1.2572590193012378 1.2653362181 4988 -1.1792613540656007 6.129568342133806 5 -2.5106755136701495 5.516267595499416 -2.492252350538816  C 0.0 0.6 1.0
5 SS 0.8335721686842064 -1.259394275432347 1.2761336729022759 -1.17926581372368 1.89138022459937 1.797789522775228 1.38608454360355 1.5794263223491414  C 0.0 0.4 1.0
6 SS 5.795195003243078 -2.427188272093893 8 7.319915388688472 3.072538856833515 1.7177467076449169 -1.300507260024197  C 0.0 0.5 0.0
7 SS 4.324754136525414 0.7262774217752579 7.325768849422001 3.0823353846709645 5.865792659072715 0.23403269294567197  C 0.0 0.3 0.0
8 SS 5.798792071714871 -2.4335966612955304 5.861700931069423 0.2591929261352574 6 7.335434545860375 3.0994147896738102  C 0.0 0.4 0.0
```

0-6 logo 檔案

此檔案建立於版本1.3，已經歷超過8次的更新，因此可得知本程式對於其舊版具備足夠的兼容性。

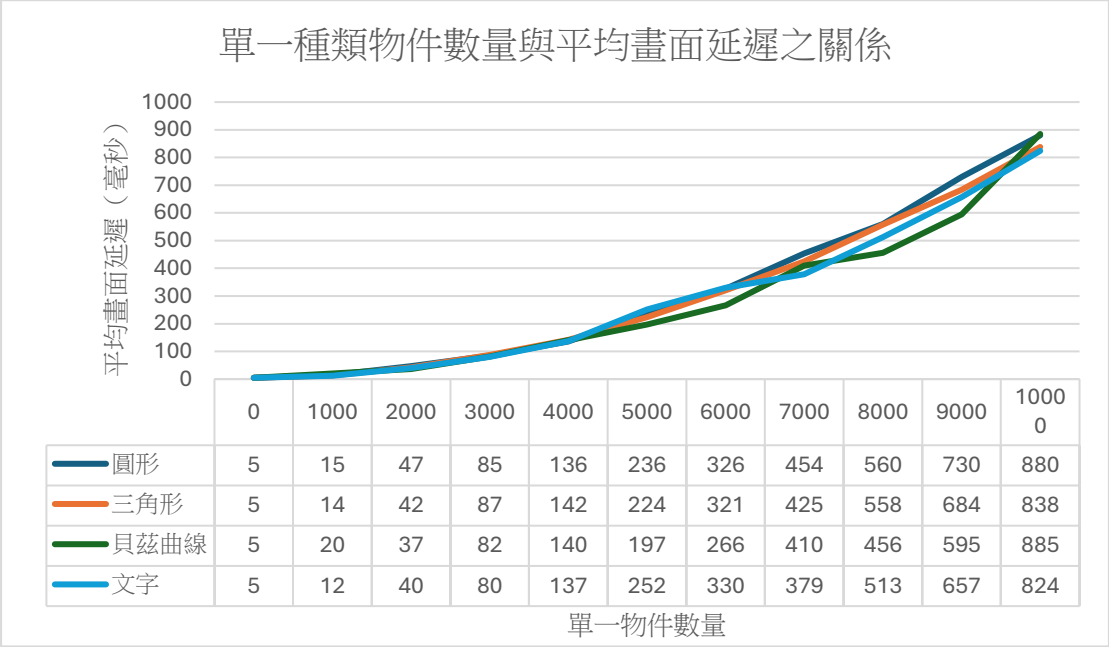
第四節 效能分析

本節針對點構式向量結構對效能的影響做全面的分析，首先將會製作數種相同物件作為樣本，再針對所有樣本做刷新率分析製作折線圖。

運行效能

本項的實驗方法是使用畫面刷新的迴圈內增加當前時間偵測並

計算時間差的方式，以奈秒級別的時間精度計算單一種類物件數量對於畫面延遲速率的影響，並分別以三種不同的經典圖形物件以及一種非經典圖形物件測試做成折線圖，其中非經典圖形物件類別為測試擴張性能之擴張物件（即文字物件）。本實驗中文字物件會與經典物件比較延遲。下面將會以折線圖方式示之。



	0	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
圓形	5	15	47	85	136	236	326	454	560	730	880
三角形	5	14	42	87	142	224	321	425	558	684	838
貝茲曲線	5	20	37	82	140	197	266	410	456	595	885
文字	5	12	40	80	137	252	330	379	513	657	824

Table 4 單一種類物件數量與平均畫面延遲之關係折線圖

本實驗所使用的硬體規格為 intel® Core™ Ultra 9 275HX、RAM 32.0GB、Intel® Graphics、NVIDIA GeForce RTX5060 Laptop GPU，所使用之 Java 執行環境為 JDK-25。 由本實驗紀錄之數據可發現系統中單一經典種類物件數量與平均畫面延遲關係近似於函數$y = 1.16 \times 10^{-10}x^3 + 6.86 \times 10^{-6}x^2 + 0.0066x + 2.5$，其中$y$為延遲時間，$x$為單一種類物件數量，經過預測分析顯示，當$x = 50000$時，$y$應約為32000毫秒，實際上當$x = 50000$時，平均刷新速率約為36000毫秒，符合預測函數模型，且非經典圖形與經典圖形之延遲並無太大差異。

第五章 結論

本章總結本研究的結果與對問題的影響，並做出回饋統計。

第一節 研究成果與總結

本研究提出一全新向量圖之架構與概念，稱為點構式向量繪圖系統，並設計基於點構式向量繪圖系統之應用程式，再加以考量現今通用前端操作，以解決當今向量圖之點線面架構較難以支持全新概念之物件以及當今繪圖軟體難以上手之問題。

第二節 研究對問題的影響

本研究透過突破性的點構式架構之概念，解決當今市面上之向量圖型繪圖軟體難以擴張全新功能之問題，成功的抽象化不同性質之物件，使新性質的物件可被更順利的兼容。

第三節 使用者測試與回饋統計

為了驗證本系統在實際操作上的可用性與使用者滿意度，本研究邀請 5 位具有專業繪圖經驗之使用者與 5 位不具專業繪圖經驗之使用者進行測試。測試項目包含介面操作流暢度、功能完整性、學習曲線以及系統穩定性四個面向，測試結果顯示，多數使用者認為本系統介面設計直覺，學習時間短，功能配置明確。特別是在「物件拖曳」與「群組管理」功能方面，獲得平均 4.6 分（滿分 5 分）的評價。然而，部分使用者指出精準托拽可更直覺化，與快捷鍵提示可更明顯。

整體而言，使用者對系統的操作體驗與效能表現皆持正面態度，顯示點構式向量繪圖系統能有效降低初學者使用門檻，並具備良好的擴充潛力。

第四節 研究限制與未來展望

本專案未來將會持續降低使用者的使用成本同時擴張功能，以及滿足與當前商用向量圖原理相互轉換的功能，並逐漸提升計算速度，以下的甘特圖表示當前成果與未來方向。

	Ver 1.11	Ver 1.12	Ver 1.13	Ver 2.0	Ver 2.1
優化垂直水平移動機制	已完成				
增加精準輸入操作	製作中				
增加文字物件	已完成				
支援 SVG 檔案					
增加矩陣轉換功能					
增加漸層					
增加自動吸附演算法					
測試 Java 8向下兼容性	已完成				
優化演算法提升計算能力					

Table 5 甘特圖

參考文獻

一、 中文部分

1. 維基百科（Wikipedia）。（2025年11月8日）。〈貝茲曲線〉。取自：https://zh.wikipedia.org/zh-tw/%E8%B2%9D%E8%8C%B2%E6%9B%B2%E7%B7%9A

二、 英文部分

1. 維基百科（Wikipedia）。（2025年11月8日）。〈Algorithmic art〉。取自：https://en.wikipedia.org/wiki/Algorithmic_art
2. Taghreed R. Alreffaee, et al. (2021). *Measure extendibility/extensibility quality attribute using object-oriented design metric.* TELKOMNIKA, Vol. 19, No. 5.