ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

OBJECT-ORIENTED PROGRAMMING

REPORT OF PROJECT

NaTruKi: An SVG Viewer Application

Nhóm trưởng: Nguyễn Phúc Khang - 24120068
Thành viên: Hoàng Trọng Nghĩa - 24120103
Thành viên: Phan Chí Cao - 24120026
Thành viên: Nguyễn Hoàng Nhật - 24120110

Thành phố Hồ Chí Minh, 12/2025

Mục lục

1 Introduction

Sau khi đã hoàn thành `milestone-2` thì nhóm đã hoàn tất được việc hiển thị các thẻ `SVGPath` và `SVGGroup`. Nối tiếp theo sự phát triển ấy, nhóm tiến hành phát triển thêm để ứng dụng có thể hỗ trợ hiển thị các thẻ `LinearGradient` và `RadialGradient`. Vì sản phẩm đã được xây dựng và phát triển theo một khuôn mẫu, và áp dụng được các `design pattern` nên là việc thêm các tính năng mới không còn quá phức tạp đối với nhóm.

2 GitHub Repository Link

Nhằm phục vụ cho việc quản lý dự án, cũng như là công khai các mã nguồn có trong dự án thì nhóm cũng đã đẩy toàn bộ dự án lên trên GitHub:

Link: See more

3 GitHub Commit List

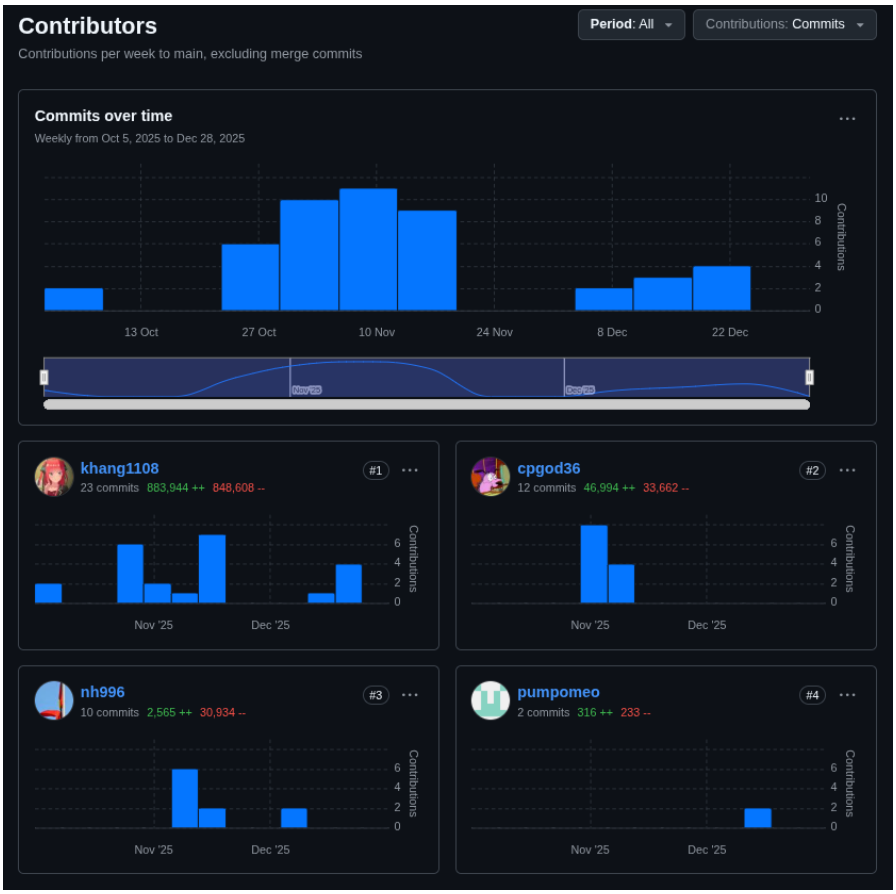
Để cung cấp một cái nhìn minh bạch và toàn diện về quá trình phát triển của dự án, dưới đây là toàn bộ lịch sử commit được trích xuất từ repository. Danh sách này được trình bày theo định dạng rút gọn, thể hiện các thay đổi, gộp nhánh và các cột mốc quan trọng từ khi bắt đầu cho đến khi hoàn thiện dự án.

```
c1939f5 add animation
576e6aa remove some unused files
e4e04c6 add gradient tag
```

Listing 1: Lịch sử commit đầy đủ của dự án NaTruKi

4 GitHub Contribution

Bằng việc sử dụng GitHub để quản lý dự án, cũng như là tổ chức và xây dựng thì nhóm cũng đã có thể ghi lại những hoạt động và đóng góp của từng thành viên như sau:



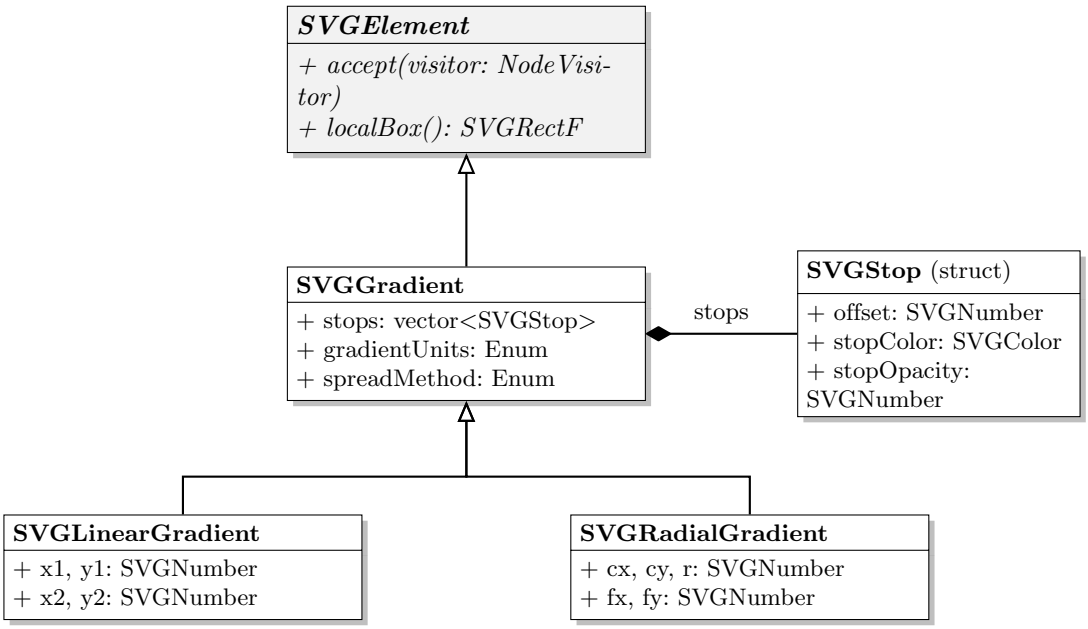
Hình 1: GitHub Contributions stats

Qua hình ảnh này ta có thể thấy được các sự đóng góp của các thành viên. Vì `milestone-3` không quá phức tạp nên số lượng commit tăng lên không quá nhiều.

5 Class Diagram

Để trực quan hóa được ý tưởng về các tổ chức và xây dựng dự án này của nhóm thì ta có thể xem qua *Class Diagram - Unified Model Language (UML)*, đây chính là sơ đồ tổng thể toàn bộ các mối quan hệ của các class có trong dự án, cũng như là các thuộc tính, các phương thức

được xây dựng cho mỗi class.



Hình 2: Chi tiết Diagram Class cho Linear và Radial Gradient

Để xem rõ hơn, nhóm cũng đã có để chi tiết `Class Diagram` ở trong `resources/plantuml.svg`.

5.1 Linear Gradient và Radial Gradient

Trong file `SVG`, tồn tại 2 thẻ chính được dùng để định nghĩa màu tô chuyển sắc (gradient) là `linearGradient` và `radialGradient`. Gradient cho phép tạo ra các hiệu ứng hình ảnh mượt mà, tăng tính thẩm mỹ và chiều sâu cho đồ họa vector vì chỉ sử dụng một màu tĩnh duy nhất, hay còn được gọi là `solid color`.

5.1.1 Linear Gradient

`LinearGradient` định nghĩa sự chuyển đổi màu sắc dọc theo một vectơ tuyến tính. Vectơ này được xác định bởi hai điểm: điểm bắt đầu $(x1, y1)$ và điểm kết thúc $(x2, y2)$.

Thuộc tính	Giá trị
`x1, y1`	Tọa độ điểm bắt đầu của vectơ gradient.
`x2, y2`	Tọa độ điểm kết thúc của vectơ gradient.

Bảng 1: Thuộc tính của `LinearGradient`

Màu sắc tại điểm bắt đầu được xác định bởi `stop` đầu tiên, và màu sắc tại điểm kết thúc được xác định bởi `stop` cuối cùng. Sự pha trộn màu sắc được nội suy tuyến tính giữa các điểm dừng dọc theo vectơ này.

```
<linearGradient id="c" x1="49%" y1="40%" x2="49%" y2="40%">
```

5.1.2 Radial Gradient

`RadialGradient` định nghĩa sự chuyển đổi màu sắc tỏa ra từ một tiêu điểm (focal point) đến mép của một hình tròn bao quanh.

Thuộc tính	Giá trị
`cx, cy`	Tọa độ tâm của vòng tròn gradient lớn nhất (nơi gradient kết thúc).
`r`	Bán kính của vòng tròn gradient.
`fx, fy`	Tọa độ của tiêu điểm (focal point), nơi gradient bắt đầu (offset 0%). Nếu không được khai báo, tiêu điểm sẽ trùng với tâm (cx, cy).

Bảng 2: Thuộc tính của `RadialGradient`

Dưới đây là ví dụ của `RadialGradient`:

```
<radialGradient id="c" cx="49%" cy="40%" r="128%" gradientTransform="matrix(.82 0 0
1 .088 0)">
```

5.1.3 Gradient Stop

Cả hai loại gradient đều sử dụng danh sách các thẻ con `<stop>` để định nghĩa các mốc màu. Mỗi điểm dừng bao gồm:

Thuộc tính	Giá trị
`offset`	Vị trí của điểm dừng trên vectơ gradient, thường có giá trị từ 0% đến 100% (hoặc 0.0 đến 1.0).
`stop-color`	Màu sắc tại vị trí điểm dừng đó.
`stop-opacity`	Độ trong suốt của màu tại điểm dừng (tùy chọn).

Bảng 3: Thuộc tính của `Gradient Stop`

Dưới đây là một ví dụ của thẻ `Gradient Stop`:

```
<stop offset=".53" stop-color="#ff3750" stop-opacity="0"/>
```

5.1.4 Hệ tọa độ và Spread Method

Một thuộc tính quan trọng khác của Gradient là `gradientUnits`, quyết định cách hiểu các giá trị tọa độ:

Thuộc tính	**Giá trị**
`objectBoundingBox` (mặc định)	Các tọa độ được tính theo tỉ lệ tương đối (0 đến 1) so với hộp bao (bounding box) của đối tượng được tô màu.
`userSpaceOnUse`	Các tọa độ được tính theo hệ tọa độ hiện hành (user coordinate system) tại thời điểm đối tượng được vẽ.

Bảng 4: Thuộc tính của `gradientUnits`

Ngoài ra, thuộc tính `spreadMethod` quy định cách tô màu cho các vùng nằm ngoài phạm vi vectơ gradient (trước điểm bắt đầu hoặc sau điểm kết thúc):

Thuộc tính	**Giá trị**
`pad` (mặc định)	Lấy màu của điểm dừng biên tương ứng để tô tràn ra ngoài (mặc định).
`reflect`	Gradient được phản chiếu (đảo ngược) liên tục.
`repeat`	Gradient được lặp lại liên tục.

Bảng 5: Thuộc tính của `spreadMethod`

5.2 Triển khai Gradient

Dựa vào các tính chất và cấu trúc của hai thẻ `LinearGradient` và `RadialGradient`, nhóm sẽ triển khai như sau:

Enum Class : Đối với hai thẻ Gradient nói trên, nó sở hữu hai thuộc tính quan trọng đó chính là `Hệ tọa độ` và `Spread Method`, nó quyết định tới cách hiểu các giá trị tọa độ có trong thẻ, cũng như là cách tô màu các vùng nằm ngoài phạm vi của vector gradient. Để lưu được các thuộc tính ấy, nhóm tiến hành khởi tạo 2 `enum class` có tên là `SVGGradientUnits` và

`SVGSpreadMethod`. Việc sử dụng Enum Class giúp cho nhóm có thể tránh được sự xung đột tên (Scoped Access), an toàn cho kiểu dữ liệu (Type Safety) và làm cho code của dự án trở nên tường minh hơn.

```
enum class SVGGradientUnits
{
    UserSpaceOnUse,
    ObjectBoundingBox
};

enum class SVGSpreadMethod
{
    Pad,
    Reflect,
    Repeat
};
```

Bên cạnh việc khởi tạo các `enum class` cho hai thuộc tính nói trên, thì nhóm cũng khởi tạo một `struct SVGStop` bởi lẽ các thẻ gradient sẽ được cấu trúc bởi nhiều thẻ `<stop>` khác nhau có trong thẻ đó, chính vì thế việc khởi tạo một `SVGStop` cũng rất là quan trọng. Cấu trúc này cần phải chứa đầy đủ các thông tin của một điểm dừng màu bao gồm vị trí (offset), màu sắc (stop-color) và độ trong suốt (stop-opacity). Vì đây chỉ là một đối tượng chứa dữ liệu đơn thuần (Data Transfer Object), không có hành vi phức tạp, nên việc sử dụng `struct` là lựa chọn tối ưu, giúp code gọn gàng và truy cập dữ liệu trực tiếp dễ dàng hơn.

```
struct SVGStop
{
    SVGNumber offset;
    SVGColor stopColor;
    SVGNumber stopOpacity;
};
```

Sau khi đã khởi tạo được các `class` và `struct` cần thiết để có thể lưu những thuộc tính cơ bản của một thẻ Gradient, thì nhóm sẽ tiến hành khởi tạo một class cha `SVGGradient`. Bởi lẽ `LinearGradient` và `RadialGradient` đều có những tính chất cơ bản nhất mà một thẻ Gradient cần có đó chính là danh sách các thẻ `<stop>`, thuộc tính `gradientUnits` và thuộc

tính `spreadMethod`. Bên cạnh đó, dựa vào cách triển khai dự án từ ban đầu, thì nhóm đã áp dụng `Visitor Design Pattern` nhằm phục vụ cho việc duyệt qua các phần tử và hiển thị chúng lên màn hình. Chính vì lẽ đó mà một phương thức `accpect()` là cực kỳ quan trọng, vì nó quyết định xem thẻ Gradient có được chấp nhận và hiển thị lên màn hình hay không.

Tiếp theo đó, nhóm triển khai hai class con chính của milestone lần này đó chính là `SVGLineraGradient` và `SVGRadialGradient` kế thừa `SVGGradient`. Bởi vì cách hiển thị và cấu trúc của hai thẻ này khác nhau về cách hiển thị tọa độ, nên là bên trong mỗi class này, nhóm sẽ tiến hành khai báo các thuộc tính riêng cho từng class. Đối với `Linear Gradient` nhóm sẽ khai báo tọa độ `x1, y1, x2, y2` còn đối với `RadialGradient` nhóm sẽ tiến hành khai báo `cx, cy, r, fx, fy`.

```
class SVGGradient : public SVGElement
{
public:
    SVGGradient() : gradientUnits(SVGGradientUnits::ObjectBoundingBox),
    spreadMethod(SVGSpreadMethod::Pad) {}
    ~SVGGradient() override = default;

    // Gradient attributes
    std::vector<SVGStop> stops;
    SVGGradientUnits gradientUnits;
    SVGSpreadMethod spreadMethod;

    // Implement pure virtual methods from SVGElement
    // Gradients do not draw themselves directly when visited
    void accept(NodeVisitor& visitor) override {}

    // Gradients do not have a bounding box
    SVGRectF localBox() const override { return {0, 0, 0, 0}; }
};

class SVGLinearGradient : public SVGGradient
{
public:
    SVGLinearGradient() : x1(0.0), y1(0.0), x2(1.0), y2(0.0) {} // Default 0% to 100%
    ~SVGLinearGradient() override = default;

```

```
    // Linear Gradient attributes (can be numbers or percentages, storing as numbers for now)
    // Note: Parsing logic will need to handle % vs numbers.
    // Usually standardizing to 0..1 bounding box or absolute values.
    SVGNumber x1, y1, x2, y2;
};

class SVGRadialGradient : public SVGGradient
{
public:
    SVGRadialGradient() : cx(0.5), cy(0.5), r(0.5), fx(0.5), fy(0.5) {}
    ~SVGRadialGradient() override = default;

    SVGNumber cx, cy, r, fx, fy;
};
```

5.3 Cơ chế Xử lý và Hiển thị Gradient

Để đảm bảo khả năng hiển thị chính xác các hiệu ứng chuyển sắc phức tạp trên các thiết bị khác nhau, nhóm phát triển đã thiết kế một quy trình xử lý dữ liệu đồ họa chi tiết, kỹ lưỡng và được chia tách rõ ràng giữa tầng dữ liệu và tầng hiển thị. Quy trình này không chỉ tuân thủ chuẩn SVG 1.1 mà còn tối ưu hóa cho framework Qt.

5.3.1 Phân tách dữ liệu

Quá trình chuyển đổi từ văn bản XML sang các cấu trúc dữ liệu C++ được thực hiện tại lớp `SVGFactoryImpl`. Thay vì xử lý tuyến tính đơn giản, nhóm áp dụng cơ chế phân tích ngữ nghĩa để chuẩn hóa dữ liệu đầu vào.

Chuẩn hóa Tọa độ và Đơn vị đo Trong chuẩn SVG, các tọa độ gradient (‘x1‘, ‘y1‘, ‘cx‘, ‘cy‘...) có thể tồn tại dưới dạng đơn vị người dùng (user units) hoặc phần trăm. Thuật toán parsing tại hàm `parseNumber` được xây dựng để nhận diện hậu tố phần trăm (%). Khi phát hiện giá trị phần trăm, hệ thống tự động chuẩn hóa về miền giá trị thực $[0.0, 1.0]$. Điều này cực kỳ quan trọng vì nó cho phép các thuật toán render phía sau hoạt động độc lập với kích thước thực tế của ảnh, đảm bảo tính chất "scalable" của đồ họa vector.

Xử lý dữ liệu Color Stops Dữ liệu về màu sắc trong gradient không nằm trực tiếp trên thẻ cha mà phân tán trong các thẻ con `<stop>`. Nhóm triển khai một thuật toán duyệt cây cục bộ để thu thập toàn bộ các thẻ `stop`. Tại mỗi điểm, màu sắc được phân tách thông qua `parseColor` - hỗ trợ đa dạng định dạng từ tên màu chuẩn (W3C named colors), mã Hex 6 ký tự, mã Hex 3 ký tự, cho đến hàm `rgb()`. Giá trị opacity cuối cùng của một điểm dừng là tích hợp giữa thuộc tính `stop-opacity` và kênh Alpha của màu gốc.

Bảng 6: Ánh xạ thuộc tính Gradient từ XML sang C++

Thuộc tính SVG	**Kiểu dữ liệu C++**	**Cơ chế Xử lý đặc biệt**
`x1, y1, x2, y2`	`SVGNumber (double)`	Chuyển đổi % sang $[0, 1]$, mặc định 0.0/1.0
`gradientUnits`	`Enum Class`	Map string sang `UserSpaceOnUse/ObjectBoundingBox`
`stop-color`	`SVGColor (struct)`	Parse Hex/RGB/Name, tách kênh Alpha
`offset`	`SVGNumber`	Clamp giá trị vào đoạn $[0.0, 1.0]$

5.3.2 Rendering và Biến đổi Ma trận

Sau khi dữ liệu gradient đã được phân tích và lưu trữ trong cây DOM, quá trình hiển thị diễn ra trong lớp `QtRenderer`. Đây là nơi dữ liệu trừu tượng chuyển hóa thành pixel trên màn hình. Quy trình này được chia thành 4 bước kỹ thuật chính:

Bước 1: Truy xuất thông tin Khi gặp một thuộc tính ‘fill‘ hoặc ‘stroke‘ có định dạng ‘url(#id)‘, hệ thống thực hiện hai tác vụ:

1. **Lookup**: Tìm kiếm đối tượng ‘SVGGradient‘ trong bảng ‘SVGDocument‘ thông qua ID.
2. **Type Resolution**: Xác định loại gradient (Linear hay Radial) thông qua cơ chế `dynamic_cast`. Nếu thất bại (con trỏ null), hệ thống sẽ revert về màu mặc định (thường là trong suốt).

Bước 2: Khởi tạo Đối tượng Dựa trên loại gradient đã xác định, một đối tượng ‘QGradient‘ (của Qt) được khởi tạo.

- **Với Linear**: Khởi tạo ‘QLinearGradient(x1, y1, x2, y2)‘.
- **Với Radial**: Khởi tạo ‘QRadialGradient(cx, cy, radius, fx, fy)‘.
- **Color Stops**: Duyệt qua danh sách ‘std::vector<SVGStop>‘. Mỗi điểm dừng được thêm vào gradient thông qua hàm ‘setColorAt(position, color)‘. Tại bước này, màu sắc được tiền xử lý để nhân (pre-multiply) với độ trong suốt (opacity) tổng thể của đối tượng.

- **Spread Method**: Cấu hình chế độ lặp lại màu sắc khi vượt quá biên gradient (Pad, Reflect, Repeat) tương ứng với enum ‘QGradient::Spread‘.

Bước 3: Ánh xạ Không gian Tọa độ Đây là bước phức tạp toán học nhất. Hệ thống phải quyết định ma trận biến đổi T_{brush} dựa trên thuộc tính ‘gradientUnits‘.

Trường hợp 1: ‘userSpaceOnUse‘ Gradient sử dụng chung hệ tọa độ với đối tượng. Ma trận biến đổi chỉ đơn thuần là ‘gradientTransform‘ do người dùng định nghĩa.

$$T_{brush} = T_{user\_transform}$$

Trường hợp 2: ‘objectBoundingBox‘ (Mặc định) Gradient được định nghĩa trong không gian chuẩn hóa $[0,1] \times [0,1]$. Cần một phép tính Affine để "kéo dãn" không gian này khớp với hình chữ nhật bao (Bounding Box) của đối tượng (x, y, w, h). Nhóm áp dụng công thức biến đổi Affine tiêu chuẩn [1] để xây dựng ma trận T_{bbox}:

$$T_{bbox} = \underbrace{\begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Translate}(x,y)} \times \underbrace{\begin{bmatrix} w & 0 & 0 \\ 0 & h & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Scale}(w,h)} = \begin{bmatrix} w & 0 & x \\ 0 & h & y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Từ đó suy ra hệ phương trình ánh xạ điểm (u, v) bất kỳ trong gradient sang tọa độ màn hình (x', y'):

$$\begin{cases} x' = w \cdot u + x \\ y' = h \cdot v + y \end{cases} \tag{2}$$

Ma trận này sau đó được nhân với ma trận nội tại của gradient (T_{grad}) để tạo ra phép biến đổi cuối cùng:

$$T_{final} = T_{grad} \times T_{bbox}$$

5.3.2.1 Bước 4: Áp dụng Brush (Brush Synthesis)

Cuối cùng, một đối tượng ‘QBrush‘ được tạo ra từ ‘QGradient‘ đã cấu hình. Ma trận T_{final} được nạp vào Brush thông qua ‘setTransform‘. Brush này được gán cho ‘QPainter‘ để thực hiện thao tác tô màu (rasterization) lên các path hoặc shape tương ứng.

`QtRenderer.cpp` minh họa việc cài đặt logic này:

```
if (userSpace) {
    brush.setTransform(gradTransform);
} else {
    // Map Unit Square (0..1) to BBox (x,y, w,h)
    QTransform bboxTransform(bbox.width, 0, 0, bbox.height, bbox.x, bbox.y);
    brush.setTransform(gradTransform * bboxTransform);
}
```

6 Các tính năng Mở rộng

Bên cạnh việc hoàn thiện các yêu cầu cốt lõi của đồ án, nhóm phát triển đã tích hợp thêm một bộ công cụ tương tác mạnh mẽ (Interactive Graphics Tools). Các tính năng này không đơn thuần là giao diện người dùng mà được xây dựng dựa trên nền tảng toán học về biến đổi hình học 2D.

6.1 Cơ chế Thay đổi Góc nhìn (View Transformation)

Để hiện thực hóa các chức năng tương tác phức tạp, hệ thống xây dựng một đường ống biến đổi (Transformation Pipeline) dựa trên tích của các ma trận biến đổi Affine. Ma trận tổng hợp T_{view} được tính toán để ánh xạ một điểm P_{world} từ không gian SVG sang điểm P_{screen} trên màn hình theo công thức:

$$P_{screen} = P_{world} \times \underbrace{T_{origin} \times T_{scale} \times T_{flip} \times T_{rotate} \times T_{pan} \times T_{center}}_{M_{view}} \tag{3}$$

Dưới đây là chi tiết kỹ thuật cho từng phép biến đổi thành phần:

1. Move (Pan) Chức năng di chuyển hình vẽ được thực hiện thông qua ma trận tịnh tiến T_{pan}. Vector dịch chuyển $(\Delta x, \Delta y)$ được tính toán từ sự thay đổi vị trí con trỏ chuột trong sự kiện `mouseMoveEvent`.

- **Logic**: Cho phép người dùng "kéo" không gian nhìn (viewport) để xem các phần khác nhau của bản vẽ.

- **Công thức**: $T_{pan} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{bmatrix}$
- **Lưu ý**: Phép Pan được áp dụng sau khi đã Scale và Rotate để đảm bảo hướng di chuyển luôn khớp với hướng kéo chuột của người dùng, bất kể hình đang xoay hay phóng to như thế nào.

2. Zoom (Scale) Chức năng phóng to/thu nhỏ bao gồm hai hệ số: hệ số tự động (S_{fit}) để hình vừa màn hình và hệ số người dùng (S_{user}).

- **Auto-Fit**: Khi tải file, S_{fit} được tính toán: $S_{fit} = 0.98 \times \min(\frac{W_{view}}{W_{svg}}, \frac{H_{view}}{H_{svg}})$.
- **User Zoom**: S_{user} thay đổi theo hàm mũ khi cuộn chuột: $S_{user} = S_{old} \cdot (1.0015)^{\Delta_{scroll}}$.
- **Tổng hợp**: Ma trận T_{scale} thực hiện phóng to đồng dạng $S = S_{fit} \cdot S_{user}$ quanh gốc tọa độ.

3. Rotate (Xoay) Chức năng xoay cho phép quay toàn bộ bản vẽ quanh tâm của chính nó.

- **Cơ chế**: Góc xoay θ được cập nhật (cộng thêm $90°$ mỗi lần) và giới hạn trong $[0, 360°)$.
- **Đảm bảo tâm xoay**: Để hình xoay quanh tâm (chứ không phải góc trái trên), trước khi xoay, ta nhân với ma trận T_{origin} để dời tâm hình về gốc $(0, 0)$. Sau khi xoay xong, ma trận T_{center} cuối chuỗi sẽ đưa hình về lại giữa màn hình.
- **Ma trận**: $T_{rotate} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

4. Flip (Lật) Chức năng lật (Mirror) tạo ra ảnh phản chiếu của bản vẽ qua trục dọc.

- **Thực hiện**: Được cài đặt đơn giản bằng một phép Scale với hệ số âm trên trục X: $Scale(-1, 1)$.
- **Kết hợp**: Do tính chất không giao hoán của ma trận, phép Flip được đặt trước phép Rotate trong chuỗi xử lý để đảm bảo hướng lật luôn cố định theo trục của đối tượng gốc, tránh gây bối rối cho người dùng khi hình đang xoay.

6.2 Thuật toán Zoom tại vị trí con trỏ

Một trong những thách thức lớn nhất là tính năng "Zoom into Cursor" (phóng to tại chỗ trỏ chuột), tương tự như các phần mềm đồ họa chuyên nghiệp (Photoshop, AutoCAD). Nhóm đã cài đặt thuật toán **Invariant Point** (Điểm bất biến) trong hàm `applyZoom`.

Nguyên lý của thuật toán là giữ cho điểm trong không gian (P_{world}) nằm dưới con trỏ chuột không thay đổi vị trí trên màn hình sau phép biến đổi Zoom. Quy trình thực hiện qua 4 bước:

Bước 1 - Inverse Mapping : Tính tọa độ thế giới của điểm dưới con trỏ chuột trước khi zoom: $P_{world} = T_{old}^{-1} \times P_{cursor\_screen}$.

Bước 2 - Update Scale : Cập nhật hệ số scale mới $S_{new} = S_{old} \cdot factor$.

Bước 3 - Forward Mapping : Tính vị trí màn hình dự kiến của P_{world} với scale mới: $P'_{screen} = T_{new\_temp} \times P_{world}$.

Bước 4 - Adjust Pan : Điều chỉnh độ dịch chuyển (Pan) để bù đắp sai lệch: $\Delta_{pan} = \Delta_{pan} + (P_{cursor\_screen} - P'_{screen})$.

6.3 Render Off-screen và Xuất ảnh

Tính năng **Save as PNG** được thực hiện thông qua kỹ thuật Off-screen Rendering. Thay vì chụp màn hình (screenshot) vốn phụ thuộc vào độ phân giải hiển thị, nhóm khởi tạo một `QImage` độc lập với kênh màu Alpha (ARGB32 Premultiplied) để hỗ trợ nền trong suốt.

Quy trình xuất ảnh tận dụng lại mẫu thiết kế Visitor:

```
QImage CanvasWidget::renderToImage(const QSize& size) {
    QImage image(size, QImage::Format_ARGB32_Premultiplied);
    image.fill(Qt::transparent); // Nen trong suot

    QPainter painter(&image);
    // Tai su dung QtRenderer de ve len QImage thay vi QWidget
    renderDocument(painter, size);

    return image;
}
```

Nhờ kiến trúc tách biệt giữa `SVGDocument` (Models) và `QtRenderer` (Views), việc chuyển đổi mục tiêu vẽ từ màn hình sang file ảnh diễn ra hoàn toàn trong suốt và không yêu cầu viết lại logic vẽ hình.

6.4 Trải nghiệm người dùng (Dynamic Loading)

Hệ thống quản lý đối tượng `SVGDocument` thông qua `std::unique_ptr`, cho phép thay thế (hot-swap) tài liệu SVG đang hiển thị một cách an toàn và tức thì. Khi người dùng tải file mới, con trỏ cũ tự động được giải phóng, bộ nhớ được thu hồi, và toàn bộ trạng thái View (Zoom, Pan) được reset về mặc định, mang lại trải nghiệm mượt mà không cần khởi động lại ứng dụng.

7 Results

Qua quá trình phát triển dự án, nhóm đã hoàn thành các tính năng sau đây trong milestone thứ ba. Dưới đây là kết quả render của các test cases khác nhau, thể hiện khả năng của ứng dụng trong việc xử lý các file SVG với độ phức tạp và cấu trúc khác nhau cùng với video demo.

Xem thêm tại đây: Google Drive

8 Kết luận (Conclusion)

Dự án xây dựng ứng dụng đọc và hiển thị file SVG (Scalable Vector Graphics) đã hoàn thành các mục tiêu đề ra, đánh dấu một bước tiến quan trọng trong việc vận dụng kiến thức Hướng đối tượng (OOP) và Đồ họa Máy tính vào thực tế.

Thông qua quá trình phát triển, nhóm đã xây dựng thành công một **SVG Rendering Engine** mạnh mẽ, vượt qua các thách thức kỹ thuật từ việc phân tích cú pháp XML phức tạp đến việc hiển thị chính xác các cấu trúc hình học và hiệu ứng màu sắc nâng cao (Gradient). Không dừng lại ở việc hiển thị tĩnh, ứng dụng đã được tích hợp bộ công cụ tương tác hoàn chỉnh bao gồm Zoom (phóng to tại trỏ chuột), Pan (di chuyển), Rotate (xoay) và Flip (lật), cùng khả năng xuất ảnh chất lượng cao, mang lại trải nghiệm người dùng hiện đại và chuyên nghiệp.

Về mặt kỹ thuật, đồ án này là minh chứng cho hiệu quả của việc áp dụng triệt để các Mẫu thiết kế (Design Patterns). Việc kết hợp linh hoạt giữa Composite cho cấu trúc cây DOM, Factory cho quá trình khởi tạo đối tượng, và Visitor cho quy trình Render đã tạo nên một kiến trúc phần mềm mạch lạc, tách biệt rõ ràng giữa dữ liệu và giao diện. Nhóm phát triển cũng đã tích lũy được những kinh nghiệm quý báu về C++ Modern, quản lý bộ nhớ tự động, và đặc biệt là việc

ứng dụng Đại số Tuyến tính (ma trận Affine) để giải quyết các bài toán biến đổi hình học trong không gian 2D.

Tài liệu tham khảo

[1] J. D. Foley, F. Van, A. Van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*. Addison-Wesley Professional, 1996.