ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN CUỐI KÌ

Đồ họa với SVG, GDI+ và CImg

Họ và tên: Lâm Đức Tân

MSSV: 1512489

Lớp: 15CNTN

Giáo viên hướng dẫn: Đỗ Nguyên Kha, Nguyễn Vĩnh Tiệp

Thành phố Hồ Chí Minh, tháng 12 năm 2016

Mục lục

I. Hướng tiếp cận và giải quyết vấn đề:

1. Xây dựng hai lớp cơ sở là RGB_Color và Point

RGB_Color dùng để giải quyết các thuộc tính màu của đối tượng, gồm 3 biến red, green, blue có kiểu unsigned char. Khai báo và cài đặt phương thức `RGB_Color& setColor(char*)` để gọi khi đọc dữ liệu màu từ file .svg.

Point là lớp mô phỏng điểm trong không gian 2 chiều.

2. Xây dựng hai lớp Figure và Attribute

a. Attribute:

Attribute là một lớp Interface đại diện cho các thuộc tính của một hình (Figure). Lớp Attribute khai báo các hàm thuần ảo là `getAttrName(): const char*`, `setValue(char*): void` và hàm hủy ảo `~Attribute()`. Hai hàm thuần ảo trên là hai hàm bắt buộc phải có đối với các lớp kế thừa từ Attribute.

Các lớp Stroke, StrokeOpacity, StrokeWidth, Fill, FillOpacity, ListTransform kế thừa từ Attribute. Các lớp này ngoài các hàm khởi tạo, hủy và hai hàm bắt buộc trên thì cần phải khai báo và cài đặt thêm các phương thức `getValue()` để có thể dễ dàng truy xuất dữ liệu khi cần.

Transform: Là thuộc tính phức tạp nhất của một hình, nên sẽ được thiết kế phức tạp nhất. Transform là một lớp Interface cho các lớp phép biến hình. Lớp Transform khai báo các hàm thuần ảo `setValue(char*): void`, `getTransformName(): const char*`, `transform(myPoint&): myPoint` (hàm này nhận tham số đầu vào là một điểm và trả về một điểm có tọa độ là tọa độ của điểm đó sau phép biến hình), `getMatrix(): Matrix*`, `getReverseMatrix(): Matrix*` (dùng để hỗ trợ khi vẽ hình bằng GDI+), `clone() Transform*` (áp dụng mẫu thiết kế prototype). Translate, Rotate và Scale là các lớp kế thừa từ Transform. Tương tự như trên 3 lớp kế thừa này đều có các hàm `getValue()` (hoặc các getter thích hợp) để có thể truy xuất dữ liệu một cách thuận tiện.

ListTransform là lớp kế thừa trực tiếp từ Attribute. Lớp này chứa một thuộc tính là `vector<Transform*>` và các phương thức `setValue(char*): void`, `setValue(const vector<Transform*>&): void`, `addValue(const vector<Transform*>&): void`, `getMatrix(): Matrix*`, `getReverseMatrix(): Matrix*`, `getValue(): const vector<Transform*>&`.

b. Figure:

Có hai Interface là Figure (dùng cho các đối tượng hình không có các thuộc tính fill (như Line)) và FillFigure (dùng cho các đối tượng còn lại). Lớp Figure chứa các thuộc tính:

```
map<char*, Attribute*, cmp_str> m_attributes,
map<char*, float, cmp_str> m_ownAttributes;
```

`m_attributes` dùng để lưu các thuộc tính dùng chung cho các đối tượng hình, đó là Stroke, StrokeWidth, StrokeOpacity và Transform; m_ownAttributes dùng để chứa thuộc tính riêng của từng lớp (ví dụ như Line thì có x1, y1, x2, y2; còn Circle có cx, cy, r,…). Lớp FillFigure kế thừa trực tiếp từ Figure và bổ sung thêm các thuộc tính fill.

Khai báo và cài đặt các phương thức getter/setter cho các đối tượng Figure, vì dữ liệu được lưu trong các map và có kiểu con trỏ nên nếu không viết hàm sẵn thì mỗi lần truy xuất rất phiền phức. Các phương thức getter/setter này tận dụng lại các getter/setter của các phương thức getter/setter của các Attribute đã được cài đặt rồi. Ngoài ra cón có các hàm `virtual void draw(CImg<unsigned char>&) = 0; virtual void draw(HDC&) = 0;` để thực hiện thao tác vẽ hình lên màn hình bằng các thư viện CImg và GDI+. Khai báo và cài đặt hàm `virtual void setByNode(xml_node<>*) = 0` và đặc biệt là `virtual void setByNodeAndAddTransform(xml_node<>*) = 0` (dùng để xử lý đối tượng Group, vì các thành phần của Group vừa kế thừa các transform của Group và vừa có các transform của riêng nó.)

Lớp Figure còn có thêm thuộc tính static là `static vector<Figure*> sampleObjects`, phương thức `static void addSample(Figure*)` và phương thức `static Figure* createObject(char*), virtual char* className() = 0, virtual Figure* clone() = 0` dùng để tạo đối tượng Figure bằng tham số chuỗi tên lớp, giúp tăng tính hiệu quả cho chương trình.

Phương thức virtual init() dùng để khởi tạo m_attributes cho Figure (cài đặt các hàm insert() cho map).

II. Cài đặt cụ thể các lớp đối tượng hình

Ta chia thành 7 nhóm cơ bản:

1. Line
2. Polyline, Polygon, Rectangle
3. Ellipse, Circle
4. Text
5. Path
6. Group
7. ListFigure

1. Line

ownAttribute của Line gồm có tọa độ của 2 đầu mút: x1, y1, x2, y2, đều có kiểu float. Phương thức `void myLine::setAttribute(char* attr_name, char* attr_value)` có thể được cài đặt như sau:

```
void myLine::setAttribute(char* attr_name, char* attr_value)
{
	map<char*, Attribute*>::iterator it1;
	map<char*, float>::iterator	it2;

	it1 = this->m_attributes.find(attr_name);
	if (it1 != this->m_attributes.end())
	{
		it1->second->setValue(attr_value);
	}
	else
	{
		it2 = this->m_ownAttributes.find(attr_name);
```

```
                if (it2 != this->m_ownAttributes.end())
                {
                        it2->second = atof(attr_value);
                }
        }
}
```

2. Polyline, Polygon, Rectangle

Polyline có một thuộc tính riêng là m_pointData. Cài đặt thêm hàm `setPointData(char*)` để đọc giá trị các điểm của Polyline từ một chuỗi cho trước.

Polygon kế thừa từ Polyline và chỉ override các hàm vẽ.

Rectangle có các thuộc tính: tọa độ đỉnh trái trên, chiều dài, chiều rộng. Do các thuộc tính này đều có kiểu float nên việc xử lí tương tự như Line. Hàm vẽ của Rectangle được viết bằng cách tạo một polygon tương ứng và vẽ polygon đó.

3. Ellipse, Circle

Ellipse có các thuộc tính cx, cy, rx, ry. Circle có cx, cy và r. Vấn đề quan trọng ở đây là khi scale một Circle theo Ox và Oy với hai tỷ lệ khác nhau thì sẽ tạo ra hình Ellipse nên hàm vẽ của Circle phải tạo một đối tượng Ellipse và vẽ đối tượng này. Có thể giải quyết vấn đề này bằng cách cho Circle kế thừa Ellipse, tuy nhiên việc kế thừa này cũng có thể dẫn đến những vấn đề khác.

4. Text

Text gồm có các thuộc tính text (giá trị của chuỗi text), x, y, font-family, font-size, text-anchor. Việc vẽ text gặp nhiều khó khăn do CImg không hỗ trợ chuyển đổi font chữ mà chỉ có một font chữ mặc định, còn GDI+ gặp vấn đề ở font-size vì không tương thích với font-size của svg. Cần viết một hàm chuyển từ char* hoặc string sang WCHAR để có thể gọi hàm DrawString trong thư viện GDI+.

5. Path

Dữ liệu đường đi của Path được lưu trong một `vector<pair<char, vector<float>>>`, với biến char lưu kí tự đầu tiên của một lệnh vẽ (C, H, V, Z, L, M), vector<float> lưu giá trị các số nằm sau kí tự đó trong file svg.

6. Group

Là đối tượng phức tạp nhất trong các đối tượng Figure, vì cấu trúc của Group là cấu trúc đệ quy (do có thể có các Group lồng nhau). Vấn đề cần giải quyết là tìm cách đọc và lưu lại các thành phần của Group. Nhờ sử dụng đa hình, chúng ta có thể giải quyết bằng cách lưu vào một `vector<Figure*>`. Tiếp theo là giải quyết vấn đề transform cho. Do các thành phần của Group được transform theo thứ tự là transform của riêng nó rồi đến transform của group bao hàm nó nên cần xây dựng phương thức `void myGroup::setByNodeAndAddTransform(xml_node<>* node)` để thêm transform vào đối tượng. Cài đặt của phương thức này có thể được viết như sau:

```
void myGroup::setByNodeAndAddTransform(xml_node<>* node)
{
	for (xml_attribute<> *attr = node->first_attribute();
		attr; attr = attr->next_attribute())
	{
		if (strcmp(attr->name(), "transform") != 0)
			this->setAttribute(attr->name(), attr->value());
		else
		{
			Figure* pFig = Figure::createObject("line");
			pFig->setAttribute(attr->name(), attr->value());
			this->addTransform(pFig->getTransform());
			delete pFig;
		}
	}

	xml_node<>* cnode = node->first_node();
	while (cnode)
	{
		Figure* pFig = Figure::createObject(cnode->name());
		pFig->setAttribute(this);
		pFig->setByNodeAndAddTransform(cnode);
		this->addElement(pFig);
		cnode = cnode->next_sibling();
	}
}
```

Hàm `void myGroup::setByNode(xml_node<>* node)` chỉ đơn giản là gọi hàm `void myGroup::setByNodeAndAddTransform(xml_node<>* node)`.

7. ListFigure

Lớp này thực chất là một `vector<Figure*>`. Có thể viết phương thức `INT ListFigure::readXML(char* fileName)` để đọc dữ liệu từ một file .svg. Hàm vẽ được cài đặt rất đơn giản bằng cách dùng một vòng for duyệt qua các phần tử của `vector<Figure*>` và gọi hàm vẽ của phần tử đó.

III. Nhận xét và Tổng kết

1. Ưu điểm:

Thiết kế tận dụng tính đa hình và kế thừa, cũng như cấu trúc dữ liệu `std::map` giúp cho việc thêm đối tượng mới cũng như thuộc tính mới trở nên dễ dàng, tránh được sự trùng lắp các mã nguồn khi chuyển đổi chuỗi sang dữ liệu, sự trùng lặp của các đoạn lệnh if.

Sử dụng phương pháp khởi tạo đối tượng bằng cách sao chép, giúp tránh được sự trùng lắp các mã nguồn tạo đối tượng và đọc thuộc tính từ tập tin, sự trùng lắp các đoạn lệnh if và

mã hóa cứng tên của lớp trong mệnh đề if, cũng như hạn chế việc sửa đổi mã nguồn của lớp ListFigure khi bổ sung thêm một đối tượng mới.

2. Khuyết điểm

Mã nguồn viết chưa gọn và rõ ràng, nhiều chỗ còn lặp lại.

Chưa tận dụng được nhiều ở các thư viện đồ họa GDI+ và CImg. Nếu tận dụng tốt được các thư viện này sẽ giúp giảm đi sự phức tạp khi thiết kế và cài đặt.

Cách thiết kế lớp Group chưa hiệu quả khi phải copy các transform của Group vào các thành phần, làm tốn bộ nhớ không cần thiết.

3. Tổng kết

Đồ án "Đồ họa với SVG, GDI+ và CImg" giúp cho sinh viên hiểu rõ và vận dụng được Phương pháp Lập trình hướng đối tượng vào việc thiết kế và cài đặt một chương trình cụ thể. Tính đóng gói, các mẫu thiết kế (Singleton, Prototype,…) và đặc biệt là tính Đa hình và Kế thừa là những yêu cầu bắt buộc và cũng gần như là giải pháp tốt nhất để thực hiện Đồ án. Ngoài ra, Đồ án này cũng giúp cho sinh viên làm quen với các đồ án phần mềm ngày càng phức tạp trong thực tế.