



FreeFrame

nl nn

→ è è → è ffi



Jeremy Meissner

Genève le 10 Juin 2022

1. Documentation	7
1.1 Page de garde	7
1.2 Abstract	7
1.3 Avant-propos	7
1.4 Remerciements	7
1.5 Introduction	7
1.6 Motivation	8
1.7 Coordonnées	8
1.8 Environnement	8
1.9 Charte graphique	9
1.9.1 Maquette	9
1.9.2 Affiches	10
1.10 Cahier des charges	10
1.10.1 Sujet	10
1.10.2 Fonctionnalités	10
1.10.3 Interface	12
1.10.4 Informations techniques	12
1.10.5 Environnement	14
1.10.6 Entretenant	14
1.10.7 Livrables	14
1.10.8 Structure	15
1.11 Étude d'opportunité	15
1.11.1 Analyse de l'existant	15
1.11.2 Critique de l'existant	17

1.12 Technologies	18
1.12.1 OpenGL	18
1.12.2 C	18
1.13 Outils	19
1.13.1 GitHub Projects	19
1.13.2 Zotero	20
1.14 Librairies	20
1.14.1 OpenTK	20
1.14.2 ImGui.NET	21
1.14.3 AnimatedGif	21
1.14.4 Newtonsoft.Json	21
1.14.5 .NET	21
1.15 Fondamentaux	21
1.15.1 OpenGL	21
1.16 Analyse fonctionnelle	27
1.16.1 Importation de fichiers	27
1.16.2 Exportation de l'espace de travail	28
1.16.3 Affichage des éléments importés	28
1.16.4 Ordre d'affichage et profondeur	31
1.16.5 Modification éléments	32
1.16.6 Créations d'éléments	41
1.16.7 Timeline	43
1.16.8 Outil de debug	44
1.17 Analyse organique	45
1.17.1 Architecture	45
1.17.2 Importation	46
1.17.3 Exportation	63
1.17.4 Sélection d'une forme	65

1.17.5 Modification d'une forme	68
1.17.6 Timeline	69
1.17.7 Interface graphique	70
1.18 Tests	71
1.18.1 Tests fonctionnels	72
1.19 Planification	79
1.19.1 Planification prévisionnelle	79
1.19.2 Planification effective	79
1.20 Conventions	80
1.20.1 Commits	80
1.21 Limitations	80
1.22 Améliorations possibles	80
1.23 Problèmes rencontrés	81
1.24 Conclusion	81
1.25 Remerciements	81
1.26 Détails versions	81
1.27 Bibliographie	82
2. Journal de bord	83
2.1 Lundi 4 avril	83
2.2 Mardi 5 avril	86
2.2.1 Processus d'apprentissage	87
2.3 Mercredi 6 avril	88
2.4 Jeudi 7 avril	89
2.5 Vendredi 8 avril	90
2.6 Lundi 11 avril	91
2.7 Mardi 12 avril	95
2.8 Mercredi 13 avril	95
2.9 Lundi 25 avril	98

2.10 Mardi 26 avril	102
2.11 Mercredi 27 avril	105
2.12 Jeudi 28 avril	106
2.13 Vendredi 29 avril	107
2.14 Lundi 2 mai	108
2.15 Mardi 3 mai	108
2.16 Mercredi 4 mai	110
2.17 Jeudi 5 mai	111
2.18 Vendredi 6 mai	113
2.19 Lundi 9 mai	117
2.20 Mardi 10 mai	118
2.21 Mercredi 11 mai	118
2.22 Jeudi 12 mai	118
2.23 Vendredi 13 mai	119
2.24 Lundi 16 mai	120
2.25 Mardi 17 mai	120
2.26 Mercredi 18 mai	121
2.27 Jeudi 19 mai	121
2.28 Vendredi 20 mai	121
2.29 Lundi 23 mai	122
2.30 Mardi 24 mai	122
2.31 Mercredi 25 mai	123
2.32 Vendredi 27 mai	123
2.33 Lundi 30 mai	123
2.34 Mardi 31 mai	123
2.35 Mercredi 1 juin	123
2.36 Jeudi 2 juin	124
2.37 Vendredi 3 juin	124

2.38 Mardi 7 juin	125
2.39 Mercredi 8 juin	125
2.40 Jeudi 9 juin	125
2.41 Vendredi 10 juin	125
3. Manuel utilisateur	126
3.1 Description de l'interface	126
3.2 Créer une forme	127
3.3 Animer une forme	128
3.4 Modifier une forme	132
3.5 Importer un fichier	133
3.6 Exporter	135
3.7 Table des figures	136



B9B mè → è →

B9C Sff èffi

This document outlines the process of completing my 2022 technician degree work. I have chosen to put myself in an uncomfortable position by choosing an area that I have never explored before. Since this project is in programming, I will apply the knowledge I have gained over the past five years.

Today's graphics engines can create complex photorealistic images. Simply from a sequence of zeros and ones in a computer they can display photorealistic three-dimensional scenes through mathematical calculations. Unfortunately, calculating complex images of this type requires a lot of computing power, which requires the use of graphics processor (also called GPU). And the interaction with these kinds of processors is done through the use of low-level graphics APIs such as Direct3D, OpenGL, Vulkan, etc.

My motivation is to implement one of these low-level graphics APIs (OpenGL) by creating a 2D renderer in C# that can display and interact with vector images.

B9D S è 8

Ce document présente le déroulement de la réalisation de mon travail de diplôme technicien de 2022. J'ai choisi de me mettre dans l'inconfort en choisissant un domaine que je n'avais auparavant jamais exploité. Ce projet étant en programmation, je vais appliquer les connaissances que j'ai acquises ces cinq dernières années.

B9E o ffi

B9F e → ffi

Les moteurs graphiques d'aujourd'hui permettent de créer des images complexes photo-réalistes. Simplement à partir d'une suite de zéros et de un dans un ordinateur, ils peuvent afficher des scènes tridimensionnelles photo-réalistes grâce à des calculs mathématiques. Malheureusement, calculer des images complexes de ce type nécessite beaucoup de puissance de calcul, ce qui oblige l'utilisation de processeur graphique (aussi appelé GPU). Et l'interaction avec ce genre de processeurs se fait via l'utilisation d'API graphiques bas niveau tel que Direct3D, OpenGL, Vulkan, etc.

B9Gi è

Ma motivation est d'implémenter l'une de ces APIs graphiques bas-niveau (à savoir, OpenGL) en créant un moteur de rendu 2D en C# permettant d'afficher et d'interagir avec des images vectorielles.

B9HV →

	Étudiant	Enseignant
Nom	Jeremy Meissner	Antoine Schmid
E-mail	jymeissner@gmail.com	edu-schmidan@eduge.ch

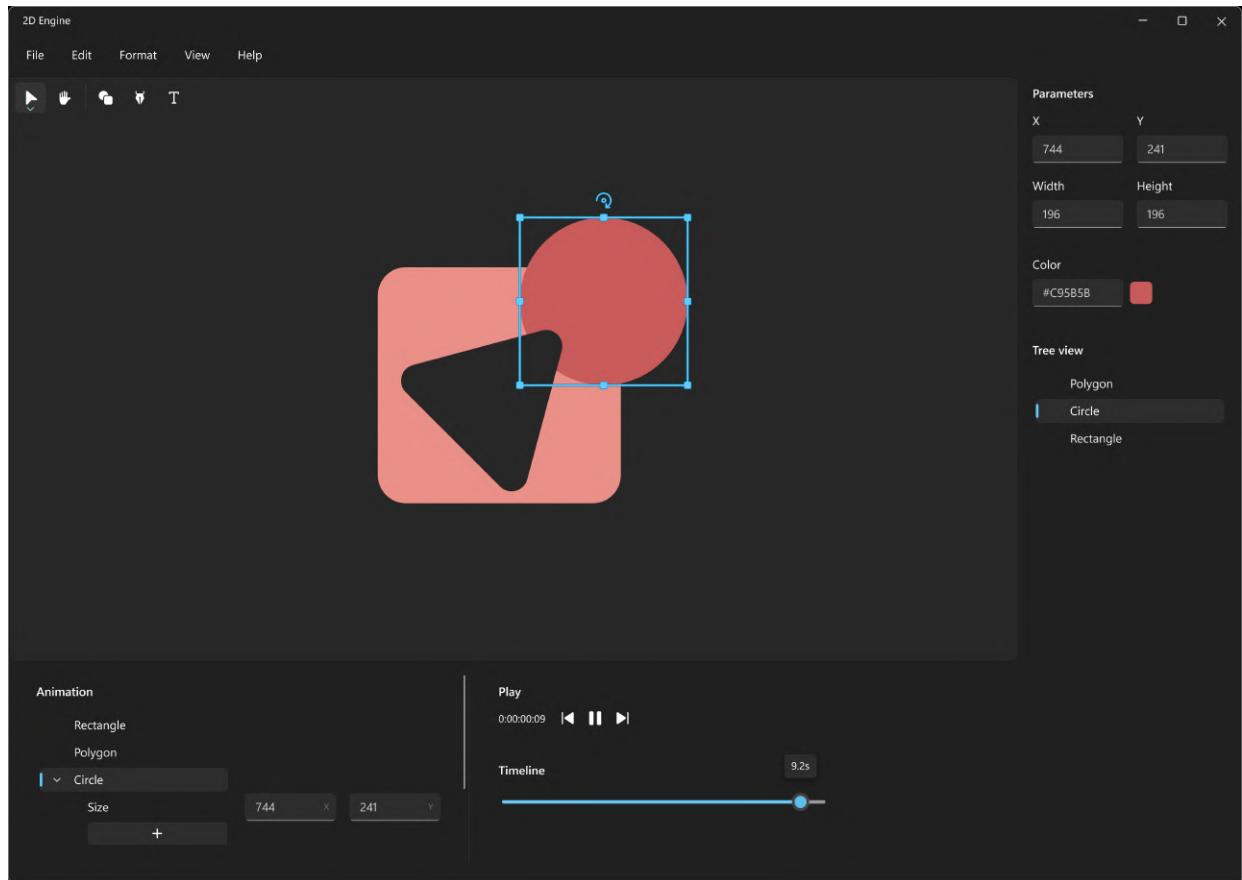
B9 a

• **Ordinateur**

- Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
- NVIDIA GeForce GTX 1060 3GB

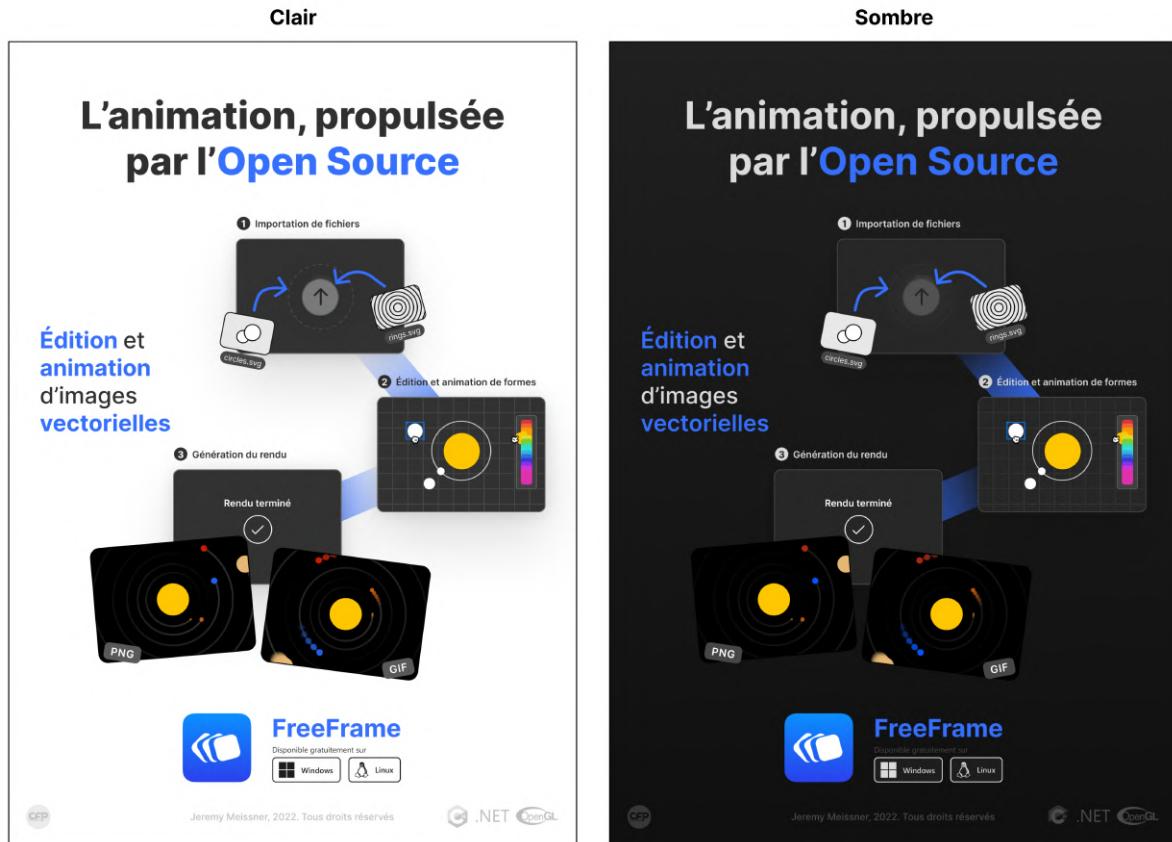
B9J V è è

B9JB i è



[Fig 1]: Maquette de l'application

B9J9CS ffi



[Fig 2]:Affiches

B9BA Vè → ffi è

B9BA9B p

Créer un moteur de rendu 2D en C# sur Windows permettant d'interagir avec des images vectorielles.

B9BA9C b ffi è

Le moteur permet d'**afficher**, de **créer**, de **modifier** et d'**animer** des éléments vectoriels.

Un élément vectoriel est la représentation graphique en C# d'un élément dans un fichier SVG (ex : `<rect .../>`).

L'application dispose d'une interface visuelle avec différents contrôleurs interactifs qui lui permet de réaliser ces actions.

S ffi

Le moteur est capable d'afficher des éléments vectoriels sur une interface en passant par OpenGL.

V

L'interface met à disposition plusieurs outils pour créer les formes suivantes :

- points ;
- lignes ;
- formes primitives (rond, carré, triangle, etc.).

i →

L'utilisateur peut modifier toutes les propriétés d'un élément :

- par l'utilisation de sa souris (en sélectionnant des points et en les glissant) ;
- depuis le panel de modification (en remplaçant les propriétés dans les champs convenus).

Les propriétés d'un élément sont : sa taille, sa couleur, sa position et sa rotation.

S

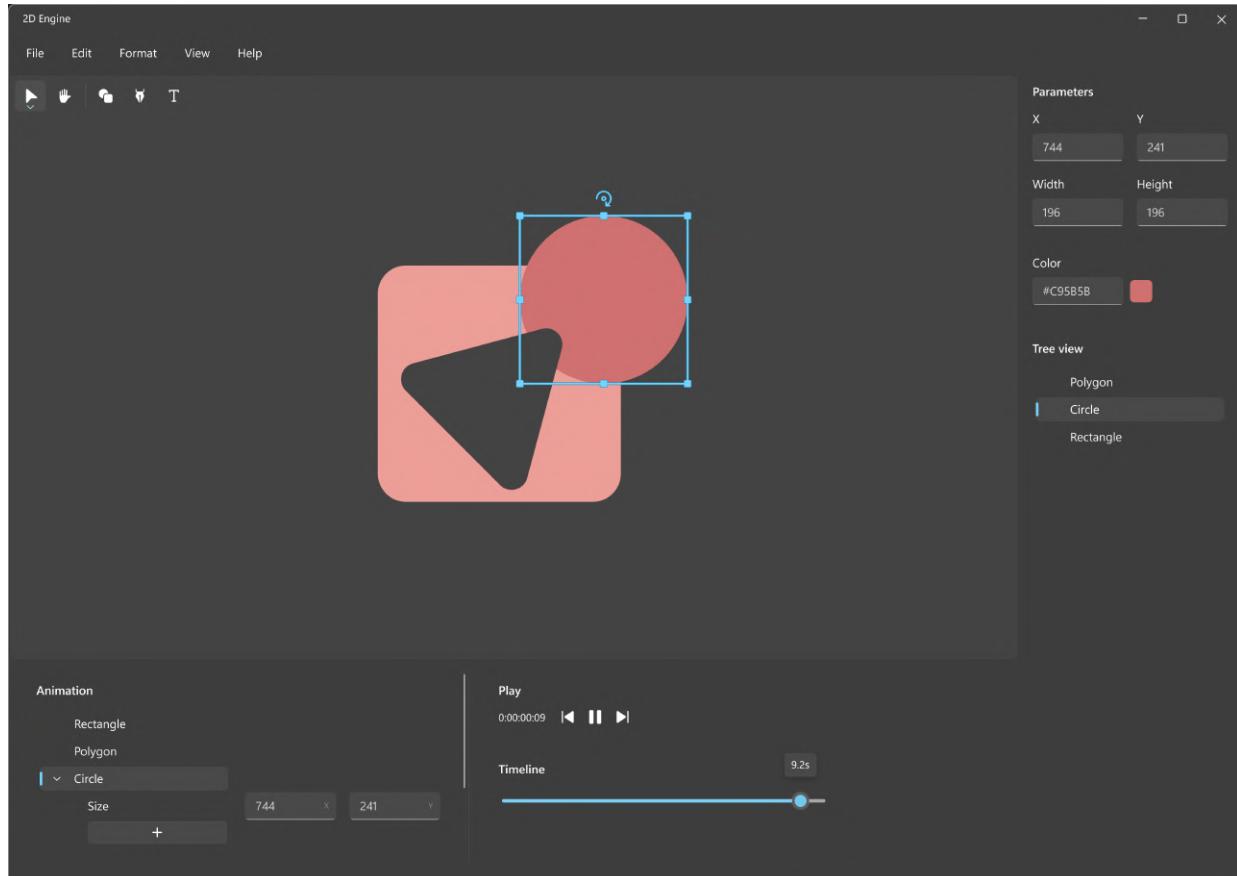
Il est possible de déplacer des éléments à la fois dans l'espace et dans le temps. Comme sur After Effects, on peut créer des animations en se déplaçant dans une timeline.

Les animations sont exportables au format GIF et MP4.

B9B4D e èffi

L'interface est répartie en trois parties majeures :

- le plan de travail central où les éléments sont affichés ;
- un panel à droite où les propriétés de l'élément sélectionné sont affichées ;
- un panel en bas pour l'utilisation de la timeline.



[Fig 3]: Maquette de l'interface de l'application

B9B4E e è ffi

q

La timeline permet de gérer le déplacement dans le temps. Déplacer des éléments en utilisant la timeline permet de créer des animations par interpolation.

La timeline est représentée par un slider. Par défaut, avancer ou reculer dans cette timeline n'a aucun effet, car il n'y a aucun keyframe.

g è

Une keyframe est la sauvegarde de l'état d'un élément à un moment donné dans le temps. L'état d'un élément est défini par ses propriétés (taille, position, couleur, etc.).

a → è K

Un utilisateur souhaite faire une animation qui déplace un rectangle du centre de l'écran vers la gauche de l'écran en une seconde.

Première étape : L'utilisateur positionne un rectangle sur le centre de l'écran. Il clique sur *new keyframe* pour déclarer le début de l'animation.

Deuxième étape : L'utilisateur avance la timeline d'une seconde

Troisième étape : L'utilisateur déplace le rectangle sur la gauche de l'écran. Il clique sur *new keyframe* pour déclarer un nouveau point d'interpolation.

Si à présent l'utilisateur se déplace dans la timeline à 0.5 seconde, le rectangle sera entre le centre et la gauche de l'écran. Les propriétés de l'élément seront interpolées par rapport aux deux keyframes précédemment définies.

e è

Il est possible d'importer des fichiers SVG.

Les balises présentes dans un fichier SVG et leurs informations (`points`, `width`, `height`, `x`, `y`, etc.) sont converties en un tableau de sommets (vertices). Leurs types sont également sauvegardés pour respecter leurs contraintes respectives (ex : le format d'un `circle` est immuable).

Uniquement les fichiers SVG contenant des **formes de bases** sont compatibles.

a è

L'exportation peut se faire par deux moyens : - exporter la scène complète dans un format de fichiers animé (GIF, MP4) ; - exporter un moment précis de la scène au format SVG.

Dans le deuxième cas, le programme convertit les différents éléments sur la scène en **formes de bases** SVG.

B9BAF a

Le moteur est écrit en **C#**, il utilise **OpenGL** (plus précisément la librairie **OpenTK**).

L'application est exclusivement disponible sur Windows 10. Le versioning et la documentation sont réalisés respectivement sur **GitHub** et **LaTeX**.

B9BAGa è

Fonction	Nom
Élève	Jeremy Meissner

B9BA9H h èff

- Manuel utilisateur
- Journal de bord
- Documentation
- Fichiers sources de l'application C#

B9BA9 p ffi



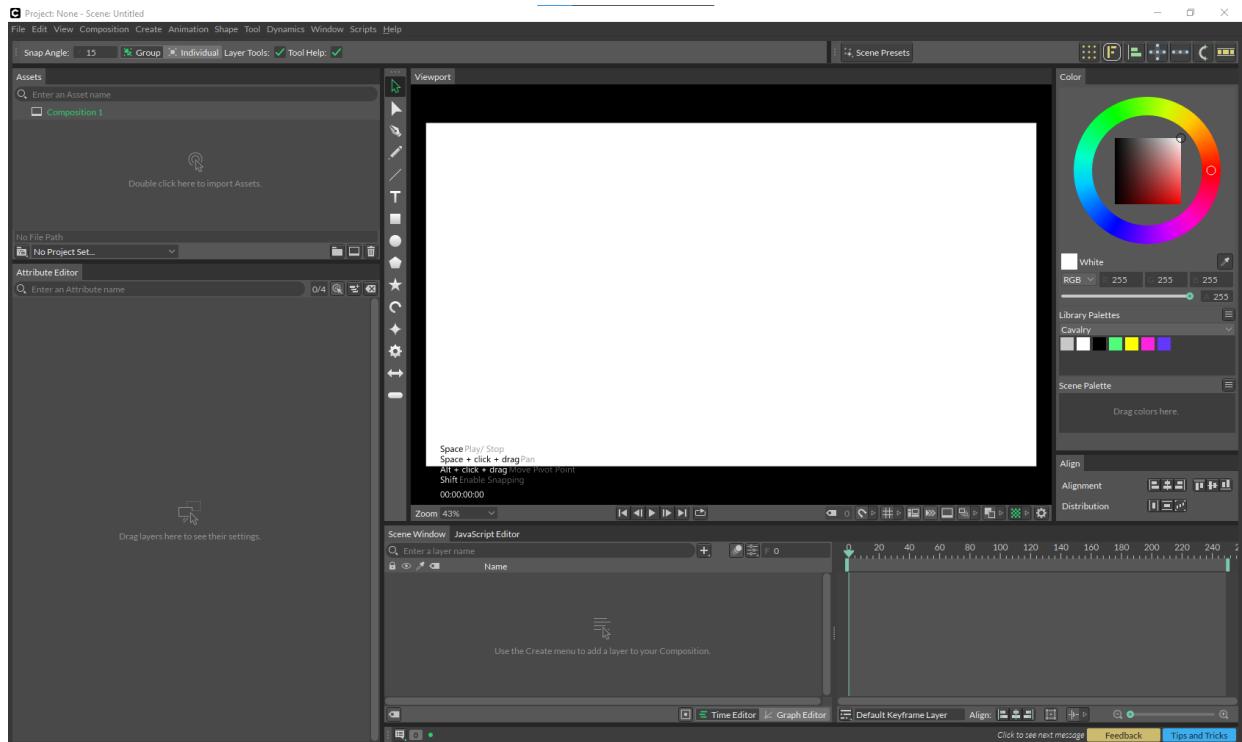
[Fig 4]:Structure interne de l'application

B9BB → →

B9BBB S è → è

Beaucoup de logiciels d'animations 2D existent, le plus connu étant [After Effects](#) (celui-ci permet également de faire de la 3D). Cependant, le projet sur le marché qui se rapproche le plus de mon idée est [Cavalry](#).

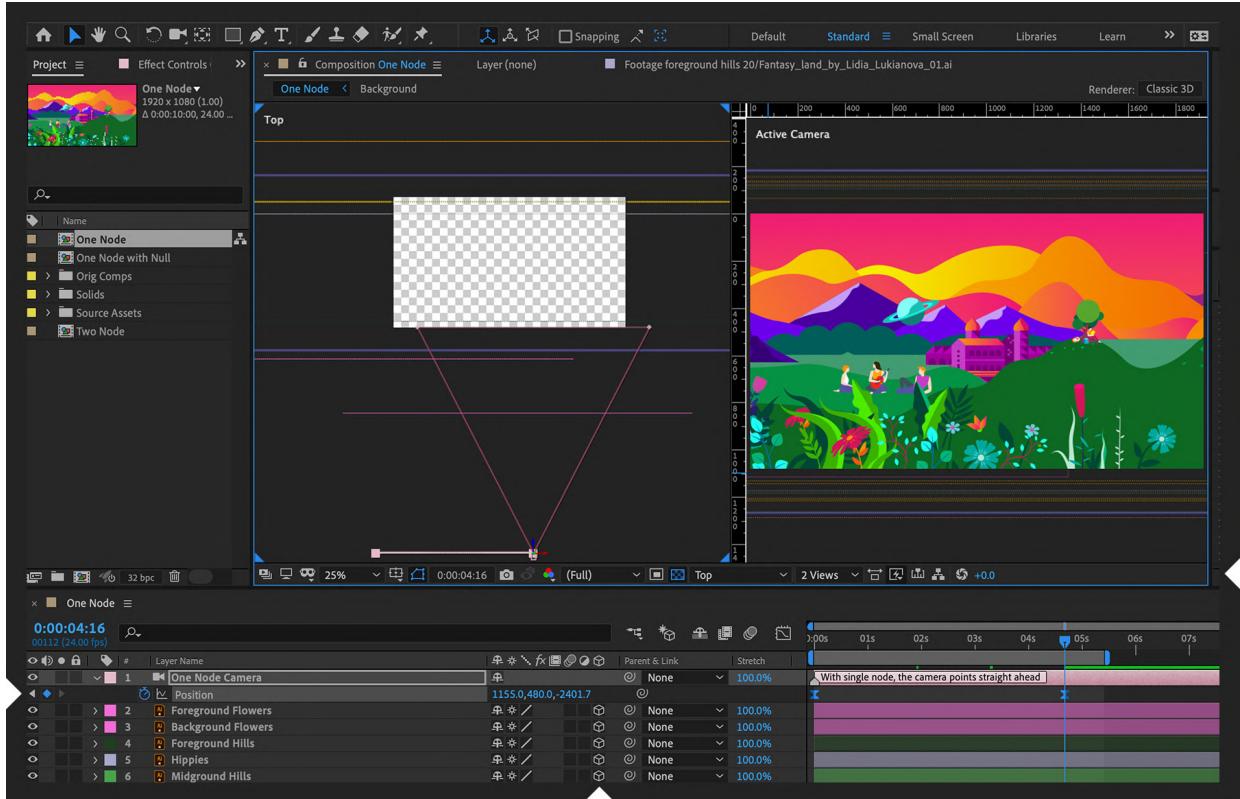
Vè è



[Fig 5]: Outil Cavalry. Source : cavalry.scenegroup.co

Cavalry est un logiciel collaboratif permettant de créer des animations 2D rapidement. Cavalry est écrit en C++ et utilise également OpenGL comme API graphique.

S a ffi



[Fig 6]:Outil After Effects. Source : adobe.com

After Effects était de base un logiciel de montage vidéo (qui est ensuite devenu l'outil que l'on connaît tous aujourd'hui). Ces outils d'animations sont très complets. Il est par exemple possible de faire du détourage avec de l'intelligence artificielle.

Ce logiciel est écrit en C++ et utilise plusieurs APIs graphiques (Direct3D, Metal).

La principale différence entre After Effects et Cavalry est que Cavalry se focalise sur des animations de formes simples, alors qu'After Effects permet de faire des animations avec des effets spéciaux poussés. Cette différence se remarque également dans l'interface. Les différents outils d'éditions que Cavalry propose sont très limités par rapport à ceux d'After Effects.

B9BB9C V → è

L'avantage majeur que mon projet possède comparé aux produits sur le marché actuel est le fait qu'il est Open Source, n'importe qui peut contribuer au projet (en ajoutant ou proposant des fonctionnalités) ou regarder comment le code a été fait. L'intégralité des bibliothèques utilisées par le projet est également Open Source (OpenTK, C#, ImGui, etc.).

B9BCq ffi

B9BC9B I c h



[Fig 7]:Logo OpenGL. Source : khronos.org

Comme dit plus haut dans ce document, OpenGL n'est pas une librairie graphique, c'est une API, soit simplement une liste de fonctions imaginées par un consortium (dans ce cas, [Khronos Group](#)), que les constructeurs de cartes graphiques ont été d'accord d'implémenter, pour permettre aux développeurs d'utiliser le même code sur différentes plateformes.

Il existe une multitude d'APIs graphiques. Celle qui possède la courbe d'apprentissage la plus douce est OpenGL - j'ai vu beaucoup d'avis sur différents sites et tous se portent à dire qu'il était très déconseillé de vouloir apprendre le domaine de l'infographie sans passer par OpenGL.

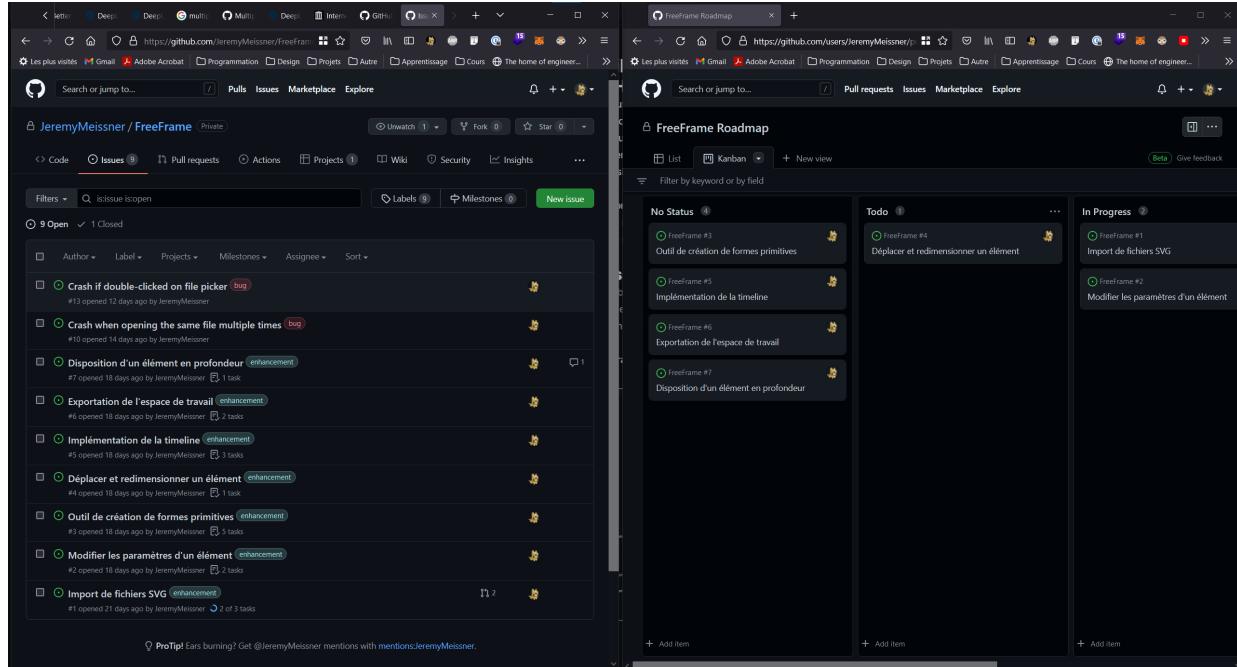
B9BC9C V

Toutes les librairies les plus utilisées pour OpenGL sont écrites en C++ (DearImGui, GLEW, etc.). Elles ont cependant été ré-écrites par d'autres personnes pour les porter sur d'autres langages tels que C#, Java, Rust, etc.

C# est le langage avec lequel je suis le plus à l'aise, je l'ai utilisé pour réaliser différents types de projets, il m'a donc paru comme le choix le plus logique pour mener à bien ce projet.

B9BDI

B9BD9B c d ff m ffi

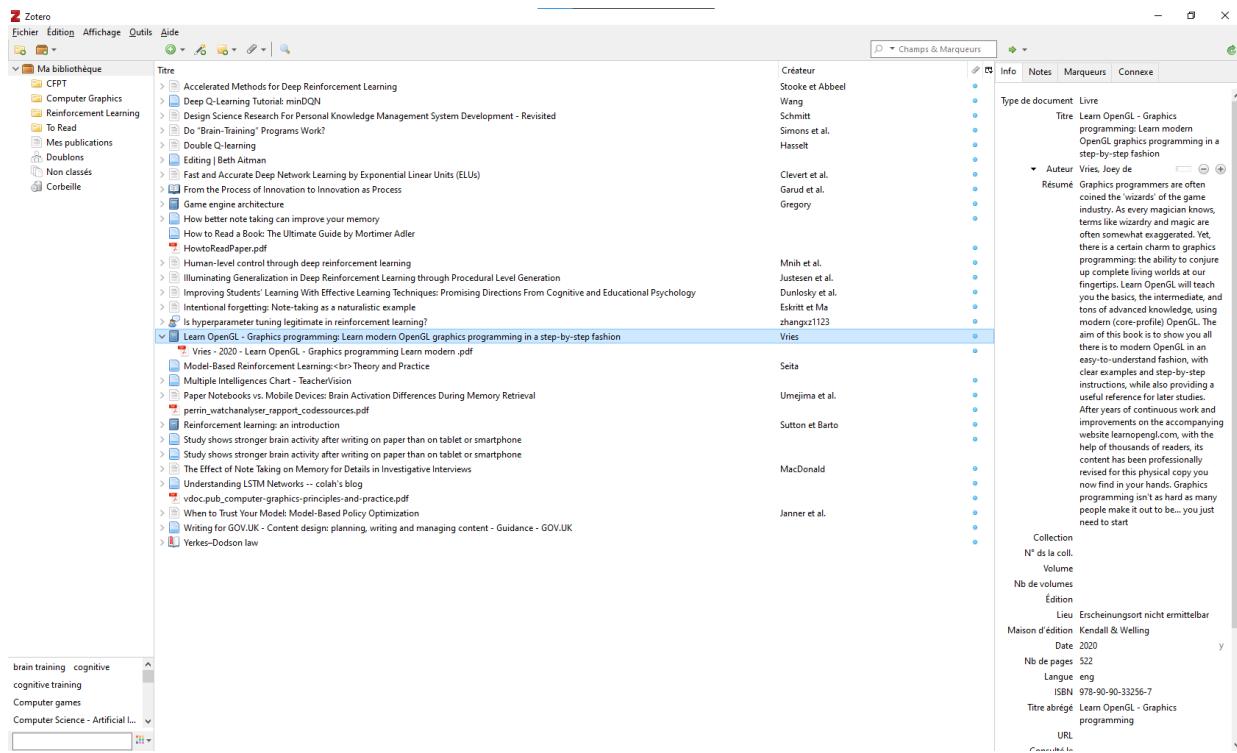


[Fig 8]:Workflow GitHub Projects

J'ai eu la chance de pouvoir appliquer Trello, Jira et Asana sur plusieurs vrais projets. Malgré le fait que ces outils soient très puissants, j'ai décidé pour ce projet d'utiliser [GitHub Project](#). La raison principale pour ce choix est ma motivation de réduire la complexité de gestion de projet en réduisant le nombre d'outils (sachant que ces différents outils démontrent leur potentiel au moment de projets de groupe, ce qui n'est pas le cas sur ce projet) pour me permettre d'être plus efficient.

J'utilise déjà GitHub pour la gestion de mon code et le fait d'avoir mon outil de gestion de tâches au même endroit que mon outil de gestion de code me permet par exemple de créer des branches pour chaque tâche majeure que je prévois de réaliser.

B9BD9C w



[Fig 9]:Espace de travail Zotero

Lorsque je fais des recherches, il m'arrive facilement de remplir ma barre d'onglets avec plus de cinquante différents sites. Pour me permettre de sauvegarder les articles les plus importants, j'utilise Zotero.

C'est un outil qui permet de sauvegarder rapidement l'adresse du site sur lequel je me trouve, mais également une copie de la page (au format HTML pour des articles ou au format PDF pour des documents scientifiques). Comme ça, si le site en question venait à fermer, je garderais toujours une sauvegarde en local (et ça m'évite de passer par [Internet Archive](#)).

Un autre avantage est que depuis l'application, avec seulement un clic, je peux créer la bibliographie complète des articles que j'ai utilisés pour mon projet, ce qui me permet de gagner beaucoup de temps.

B9BE h ff è

B9BE9B I qg

[OpenTK](#) est une librairie Open Source soutenue par une communauté de plus de 500 personnes qui a pour but de rendre accessible l'utilisation d'OpenGL en C#.

B9BEG C e c g aq

[ImGui.NET](#) est une librairie qui permet de créer des interfaces utilisateur rapidement (Comparé à une librairie d'UI qui permet de créer des interfaces poussées, destinées à l'utilisateur moyen).

B9BEG S è -e

[AnimatedGif](#) possède des méthodes permettant d'exporter facilement une séquence d'images en GIF, les librairies de .NET étant trop limités.

B9BEG j g

[Newtonsoft Json](#) permet de faire de la sérialisation/désérialisation. [System.Text.Json](#) existe, cependant il ne possédait pas toutes les fonctionnalités dont j'avais besoin pour un outil de ce type.

B9BEG g aq

[.NET](#) l'environnement de travail parfait pour C#. Il possède toutes les librairies de base nécessaires à la création d'un programme en C#. Il permet également de compiler mon programme C# pour le rendre compatible sur plusieurs plateformes (Windows, Linux, macOS).

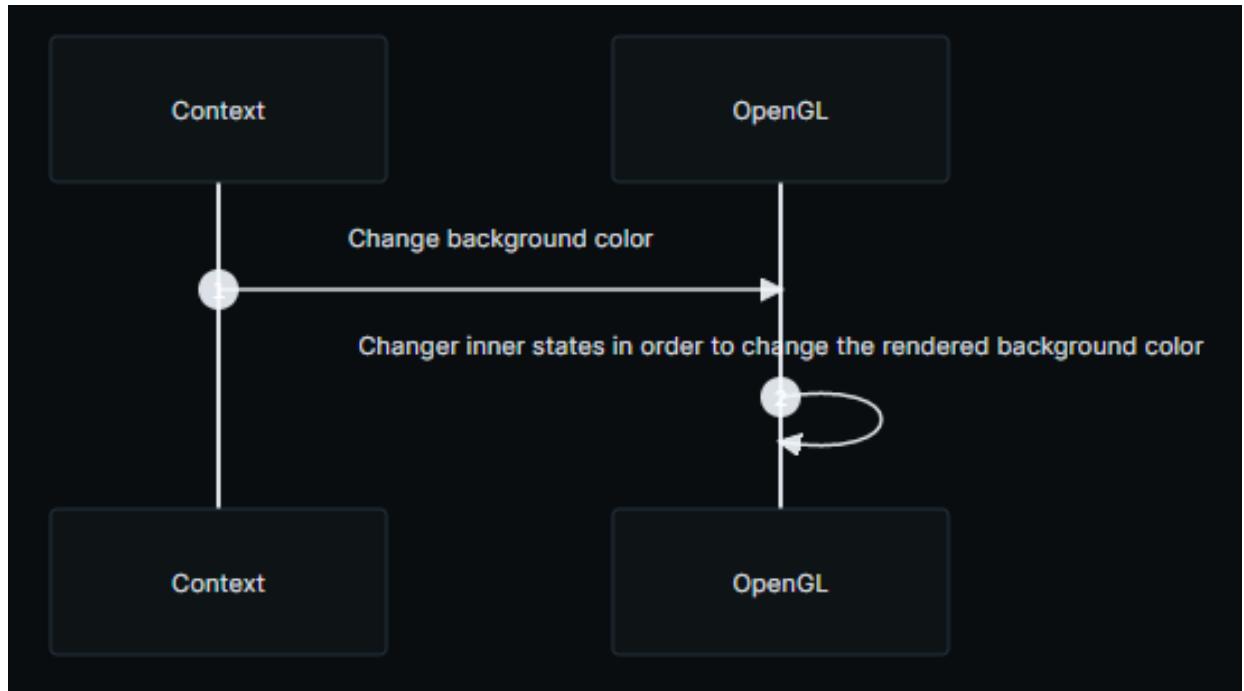
Il est Open Source et il est soutenu par Microsoft.

B9BF b è è

B9BF9B I c h

OpenGL a la particularité d'être structuré dans un modèle dit *State machine*. Dans le cas d'OpenGL, cela signifie qu'il est possible d'activer/désactiver/modifier ce qu'on appelle des états qui influenceront son fonctionnement.

Exemple de changement d'un état dans OpenGL :

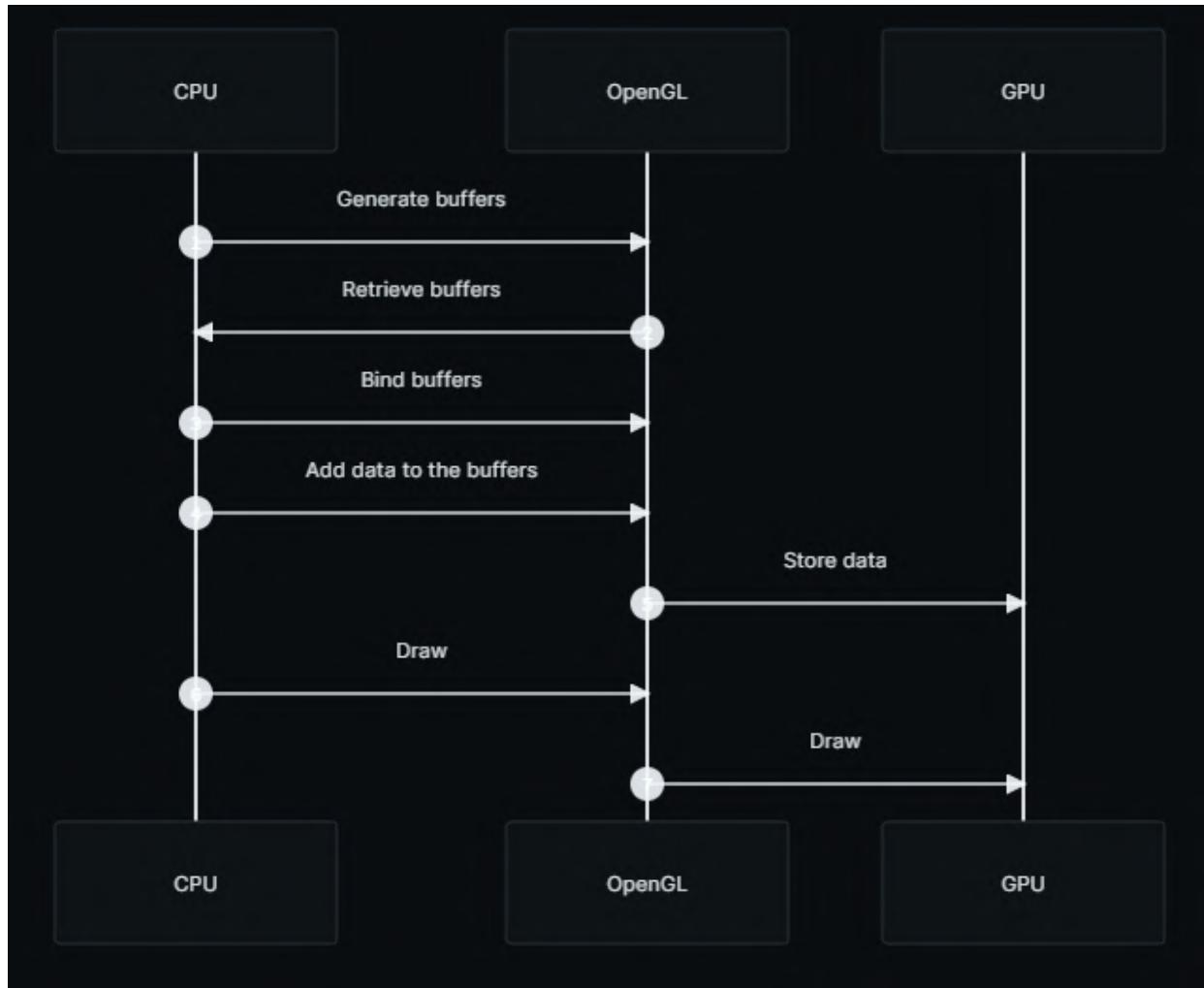


[Fig 10]:Diagramme1

Grâce à *State Machine*, OpenGL possède une structure d'utilisation bien précise.

Si l'on souhaite par exemple afficher un triangle sur l'écran, il faut :

1. Demander à OpenGL de créer un endroit où les différentes propriétés de notre triangle vont être stockées (position des sommets, couleurs, textures, etc.). Cet endroit est appelé un **buffer**.
2. On va dire à OpenGL les **buffers** que l'on souhaite utiliser.
 - a. Car si l'on souhaite afficher plusieurs triangles il faudra créer différents **buffer** pour les propriétés de chaque triangle.
3. Ajouter des données sur les **buffers**. OpenGL comprend sur quel buffer on souhaite envoyer les données, car on lui a justement dit auparavant.
4. Finalement, il suffit de demander à OpenGL d'afficher les données des **buffers**.



[Fig 11]:Diagramme1

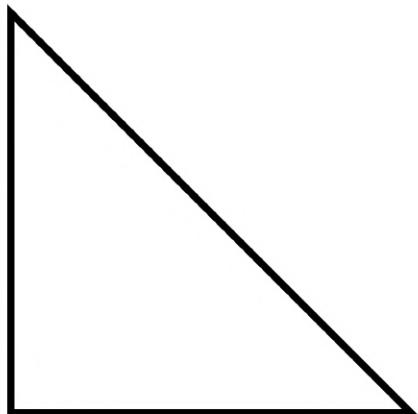
I ff

Le principal **buffer** utilisé sur OpenGL est le **Vertex Buffer Object**(VBO), il permet de sauvegarder les propriétés d'un sommet, donc comme dit plus haut, la position des sommets, leur couleur, leur texture, etc.

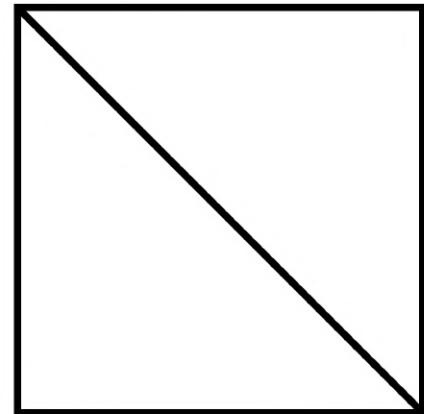
Il en existe d'autres, par exemple l'**Index Buffer Object**(IBO) est également souvent utilisé. Pour comprendre son besoin, il faut tout d'abord comprendre comment fonctionne la gestion de mémoire sur OpenGL :

Dès le moment où l'on affiche des objets qui possèdent plus de trois sommets, par exemple un carré (qui en possède quatre), ce carré va être représenté comme étant par deux triangles.

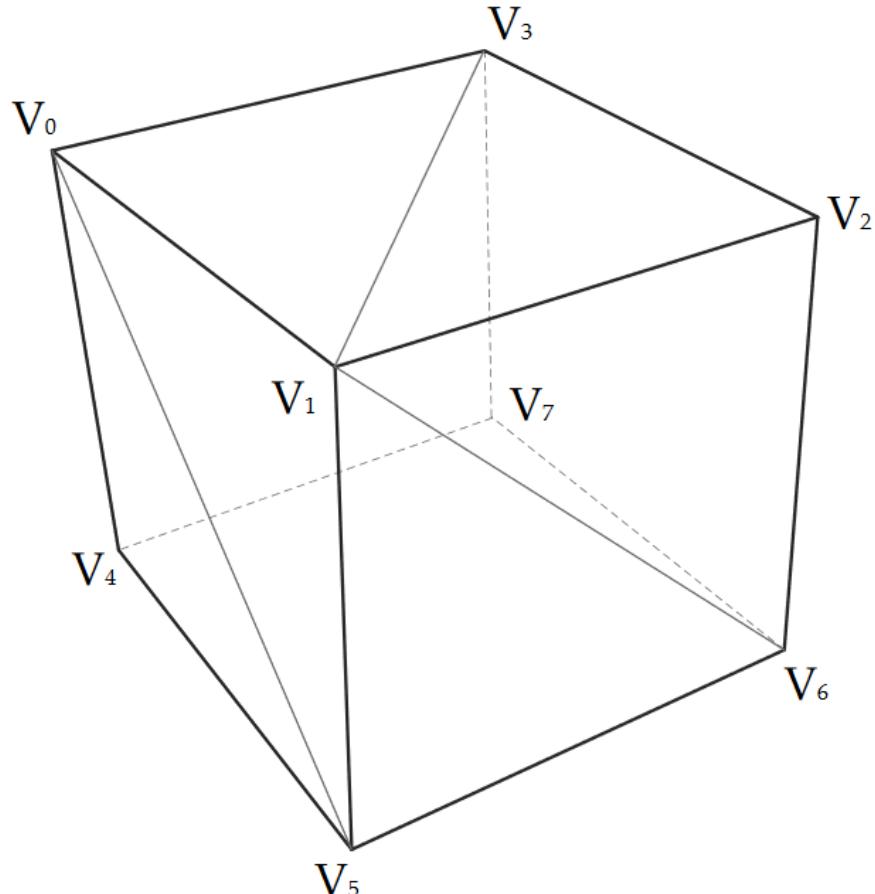
Triangle



Square

*[Fig 12]:Différence Triangle et Square*

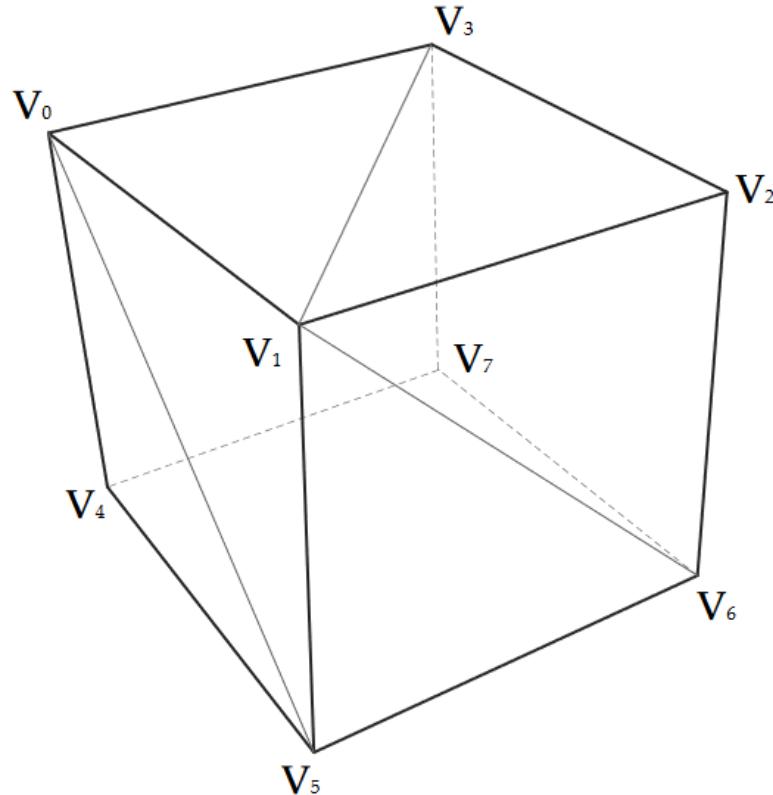
On remarque donc que dans cet exemple, les triangles partagent deux sommets, et par défaut OpenGL va sauvegarder en double les propriétés de ces sommets, ce qui inutilement redondant.



V ₀	V ₁	V ₃	V ₁	V ₂	V ₃	V ₀	V ₅	V ₁	...	V ₅	V ₇	V ₆
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	-----	----------------	----------------	----------------

[Fig 13]:Triangle Lists. Source : Game engine architecture - Jason Gregory

Une des solutions possibles est de stocker une seule et unique fois chaque sommet et en parallèle d'avoir un tableau d'entier qui va retenir à quel triangle appartient quel sommet. Un tableau d'entier étant beaucoup plus léger qu'un sommet pouvant posséder une position, des couleurs, des textures, etc.



Vertices	V₀	V₁	V₂	V₃	V₄	V₅	V₆	V₇
----------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------

Indices	0	1	3	1	2	3	0	5	1	...	5	7	6
---------	---	---	---	---	---	---	---	---	---	-----	---	---	---

[Fig 14]:Indexed Triangle Lists. Source : Game engine architecture - Jason Gregory

L'**Index Buffer Object** est donc simplement un tableau d'entier qui va stocker les sommets que possède chaque triangle.

Sur cette figure :

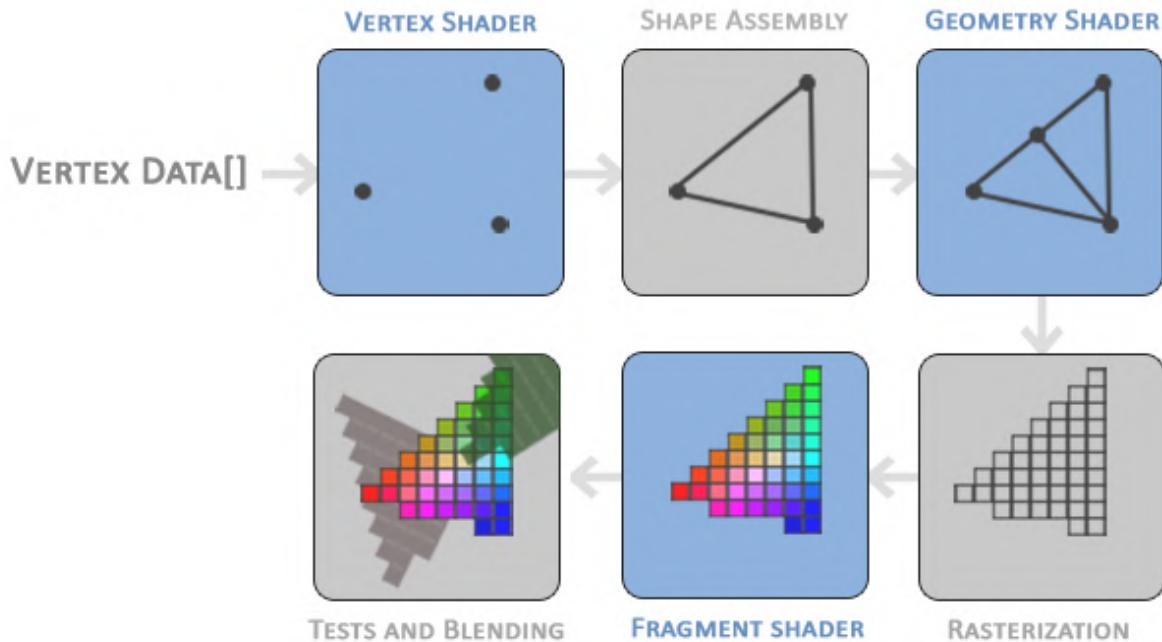
- Vertices est le **Vertex Buffer Object**
- Indices est l'**Index Buffer Object**

Finalement, pour que OpenGL puisse lier ces deux buffers, on va utiliser ce qu'on appelle un **VertexArray Object(VAO)**

p è→

Un **shader** est simplement un programme qui va être exécuté au niveau du GPU au moment de la pipeline de rendu.

La pipeline de rendu possède six étapes, chacune utilisant un **shader** spécifique. Notez que dans l'image ci-dessous, les sections bleues représentent les étapes où l'on peut changer le **shader**.



[Fig 15]:Pipeline de rendu dans OpenGL. Source : learnopengl.com

Dans la grande majorité des cas, uniquement deux **shaders** sont utilisés, le *Vertex Shader* et le *Fragment Shader*.

Le *Vertex Shader* va être exécuté pour chaque sommet du triangle (donc dans ce cas, trois fois), il permet de décider de la position finale des sommets dans l'espace, à partir des *Vertex Data[]* donnés.

Le *Fragment Shader* va quant à lui être exécuté pour chaque pixel du triangle, il permet de décider de la couleur du pixel ou de la texture appliquée.

B9G9S è ffi

B9G9B e è → ffi

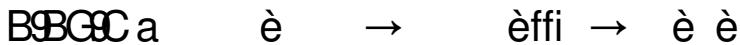
Il est possible d'importer un fichier au format SVG.

Il y a deux méthodes d'importations :

- Importer en écrasant ;
- Importer en ajoutant.

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect width="100" height="80" x="0" y="0" rx="0" ry="0"/>
  <circle cx="50" cy="50" r="50"/>
  <line x1="0" y1="80" x2="100" y2="20"/>
</svg>
```

L'application est seulement compatible avec certaines balises SVG. Si un élément ajouté n'est pas compatible, un message s'affichera à l'écran de l'utilisateur pour l'avertir que certaines formes n'ont pas pu être rajoutées. Si le fichier SVG est corrompu, un message d'erreur préviendra l'utilisateur.



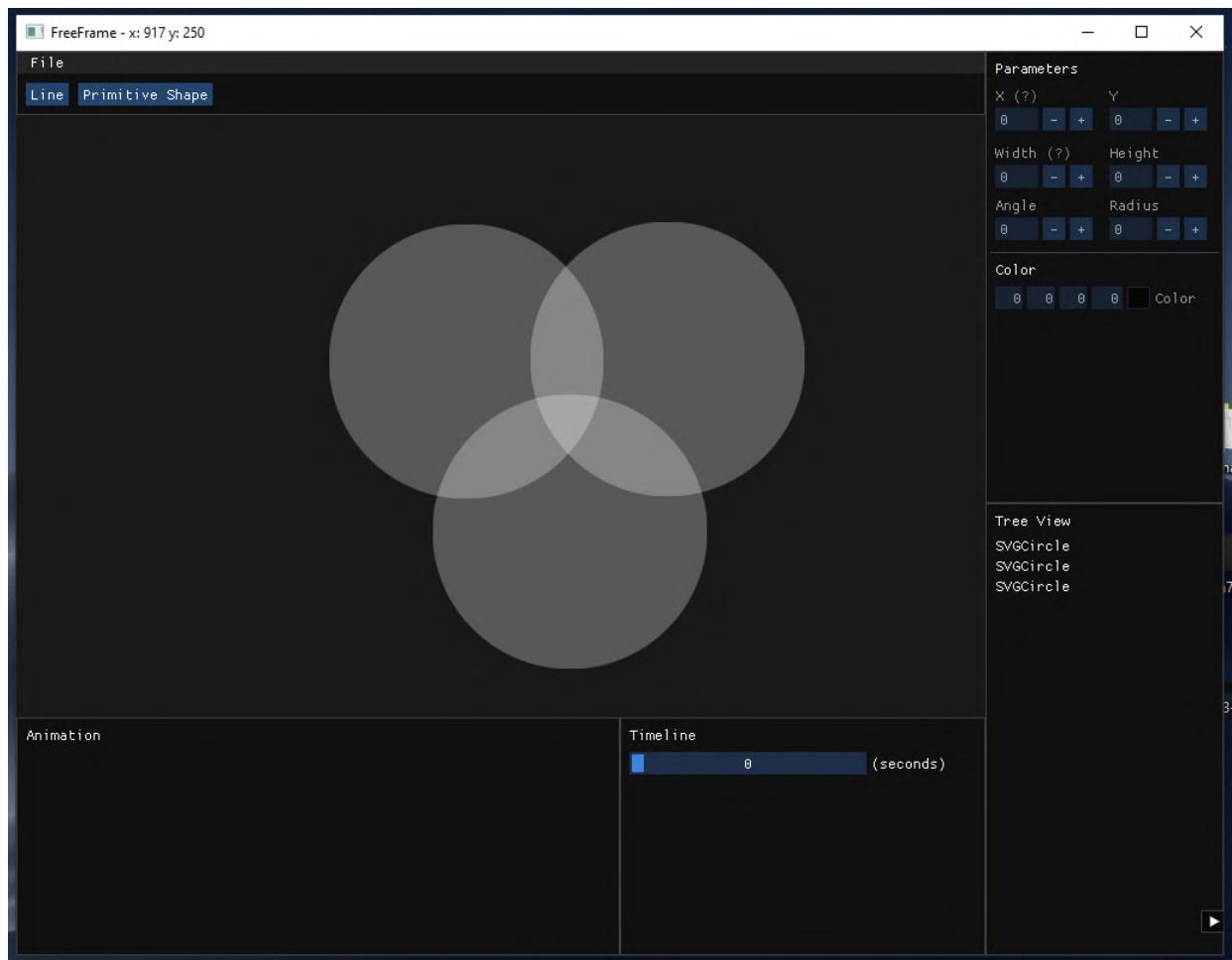
Il est possible d'exporter l'espace de travail en cinq formats :

- SVG ;
- PNG ;
- GIF ;
- MP4 ;
- FreeFrame.

Pour exporter, il suffit de cliquer sur *File* puis sur le format souhaité.

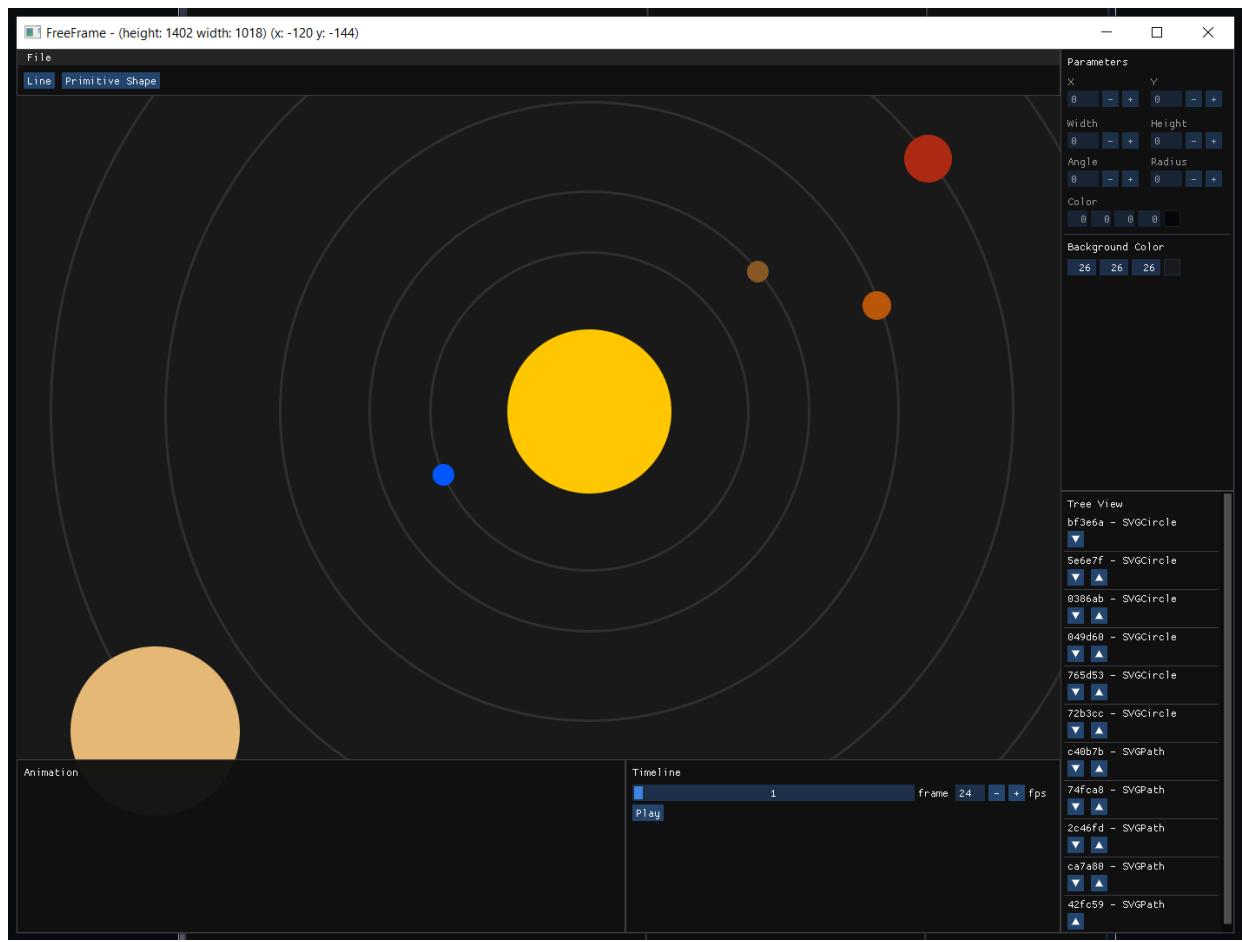


Une fois les éléments chargés en mémoire, ils seront tous affichés sur l'espace de travail selon les propriétés qu'ils possèdent. Cela peut être la forme de l'élément, une couleur, une opacité, une position, une taille, une rotation, un arrondi de bordures.



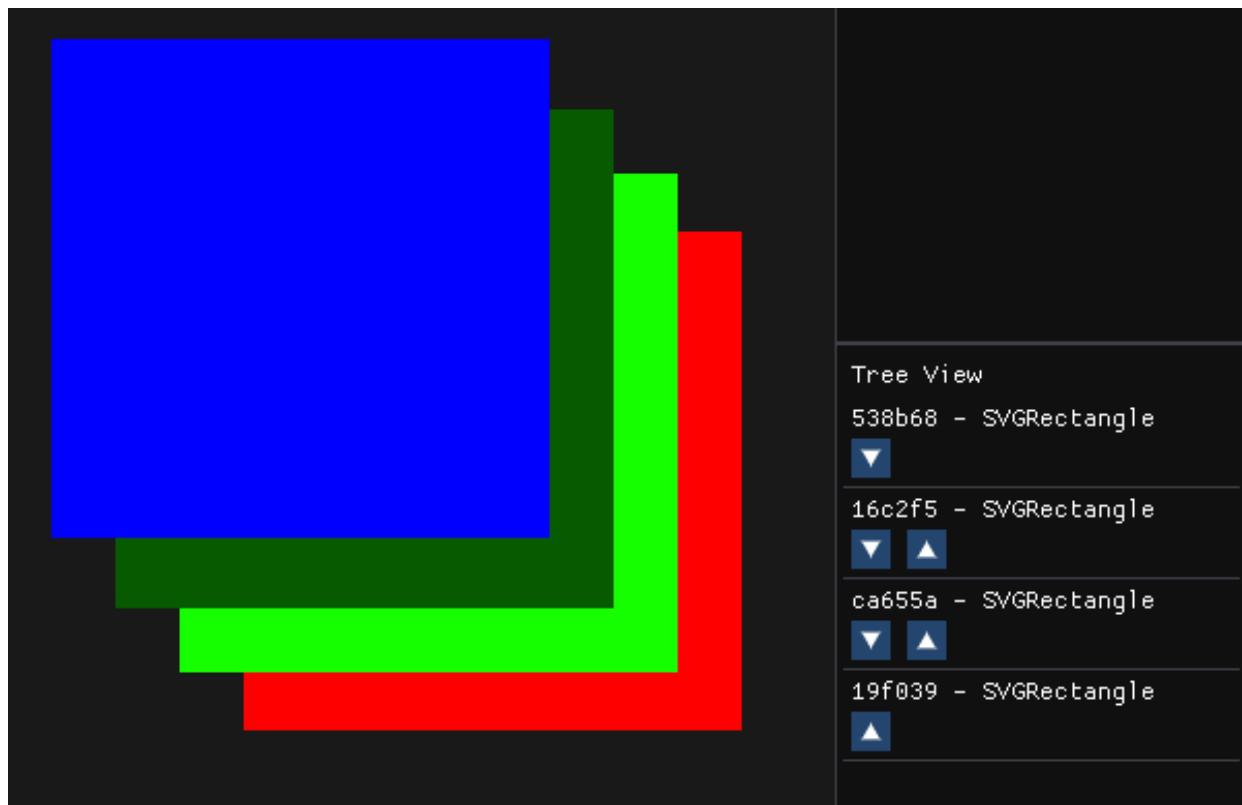
[Fig 16]:Trois ronds superposés

La limite des éléments compatibles n'empêche pas de réaliser des scènes très complexes, comme le montre la représentation du système solaire ci-dessous.



[Fig 17]:Représentation du système solaire

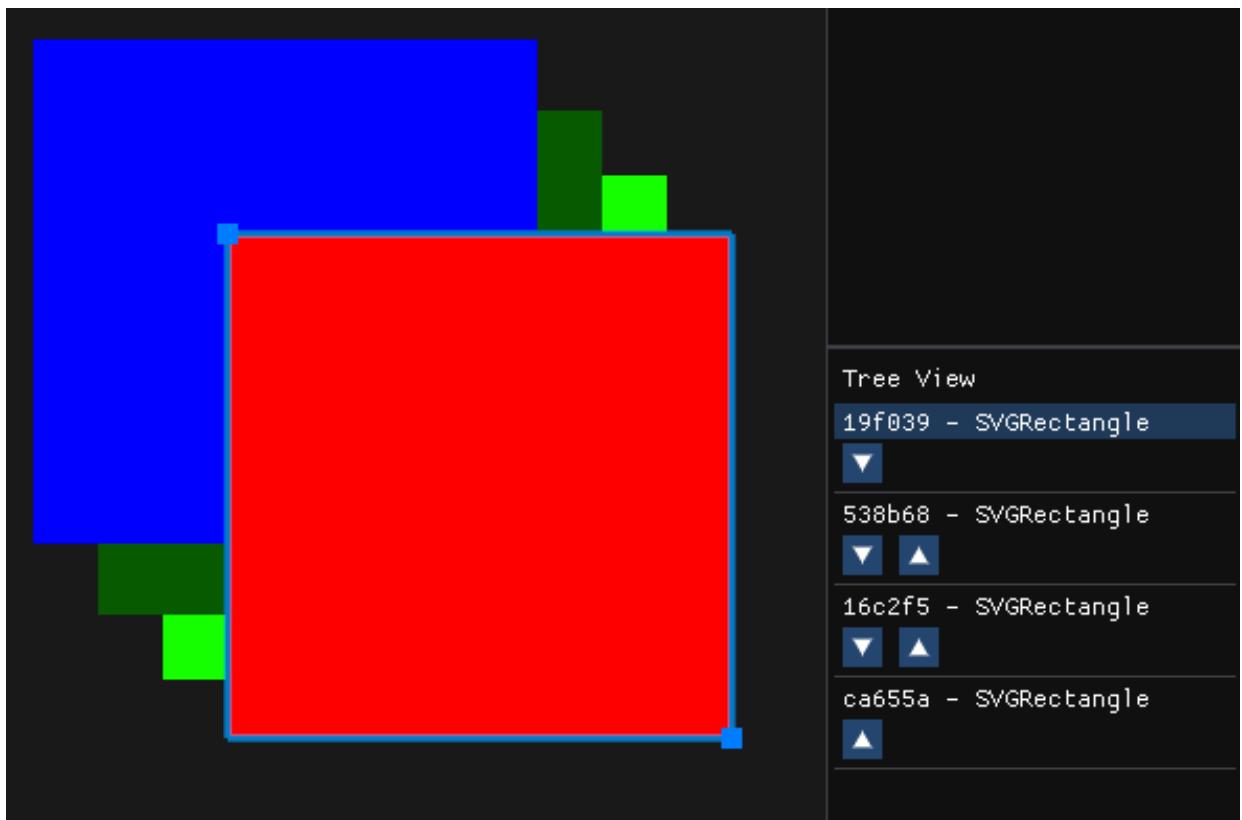
B90E1 → → è ffi è →



[Fig 18]:Quatre rectangles colorés superposés, ordonné

Il est possible de changer l'ordre d'affichage des formes.

En utilisant l'affichage Tree View l'utilisateur peut cliquer sur les boutons qui permettent de descendre ou monter la priorité d'affichage d'une forme.



[Fig 19]:Quatre rectangles colorés superposés, désordonné

On peut identifier les formes par un id unique qui leur est attribué à leur création.

B9GOF i → ffè

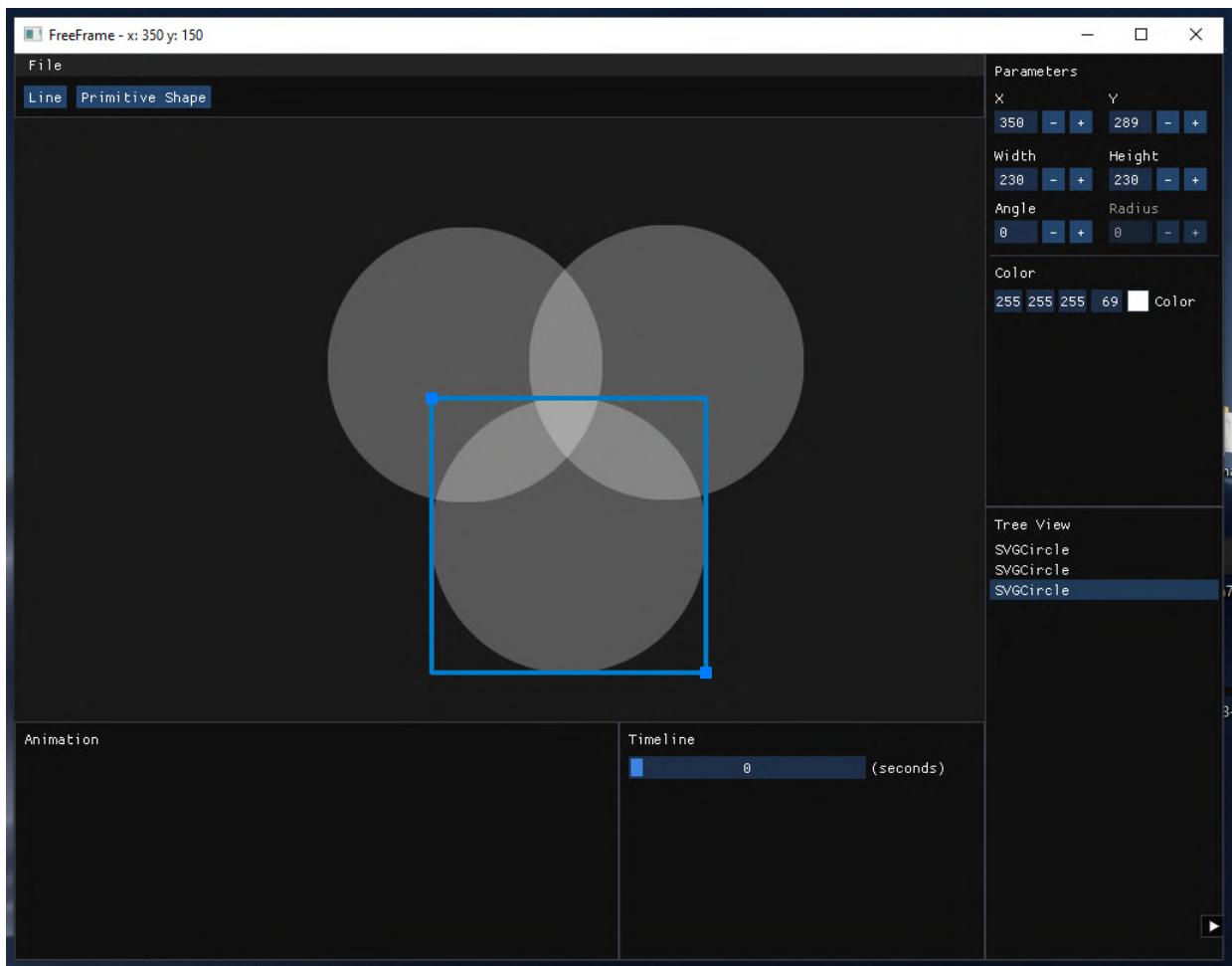
p ffi →

Pour modifier les propriétés d'une forme, il est impératif que cet élément soit sélectionné.

Pour sélectionner une forme, il suffit d'appliquer un clic gauche dans l'espace de travail. La forme la plus proche du clic va ensuite être définie comme sélectionnée. Il n'est donc pas obligatoire que la souris soit au-dessus de la forme, appliquer un clic dans le vide va simplement sélectionner la forme la plus proche.

Une boîte de sélection bleue s'affiche autour de l'élément qui est actuellement sélectionné - le sélecteur bleu permet entre autres de déplacer et changer la taille de l'élément sélectionné.

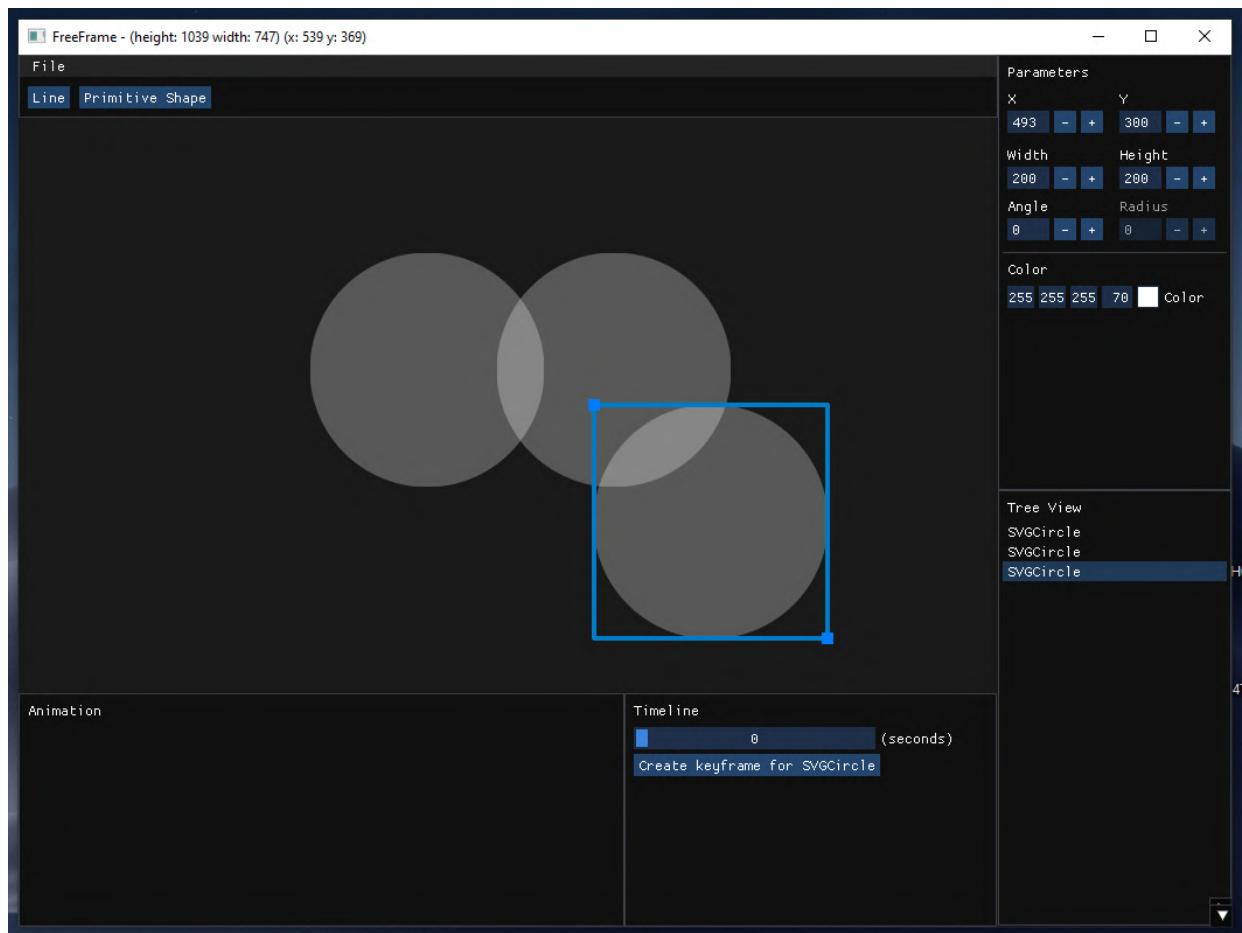
Pour stopper la sélection d'un élément, il suffit d'appuyer sur la touche **Esc**.



[Fig 20]:Trois ronds superposés, un sélectionné

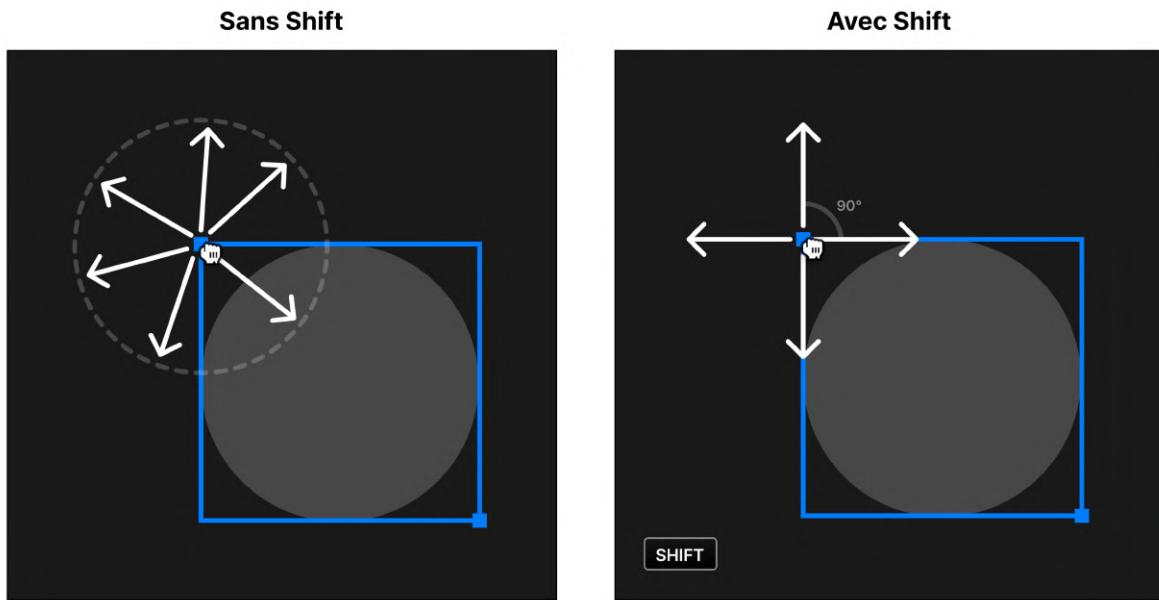
Z èffi →

Il est possible de déplacer la forme sélectionnée soit via l'interface - en changeant directement les valeurs de position **X** et **Y**. Soit en cliquant et déplaçant le point en haut à gauche du sélecteur bleu.



[Fig 21]:Trois ronds superposés, un décalé

Le déplacement avec le sélecteur bleu possède une option d'alignement. Si l'utilisateur appuie sur la touche **Shift** lors du déplacement, le déplacement sera aligné à 90° par rapport à la position de départ.

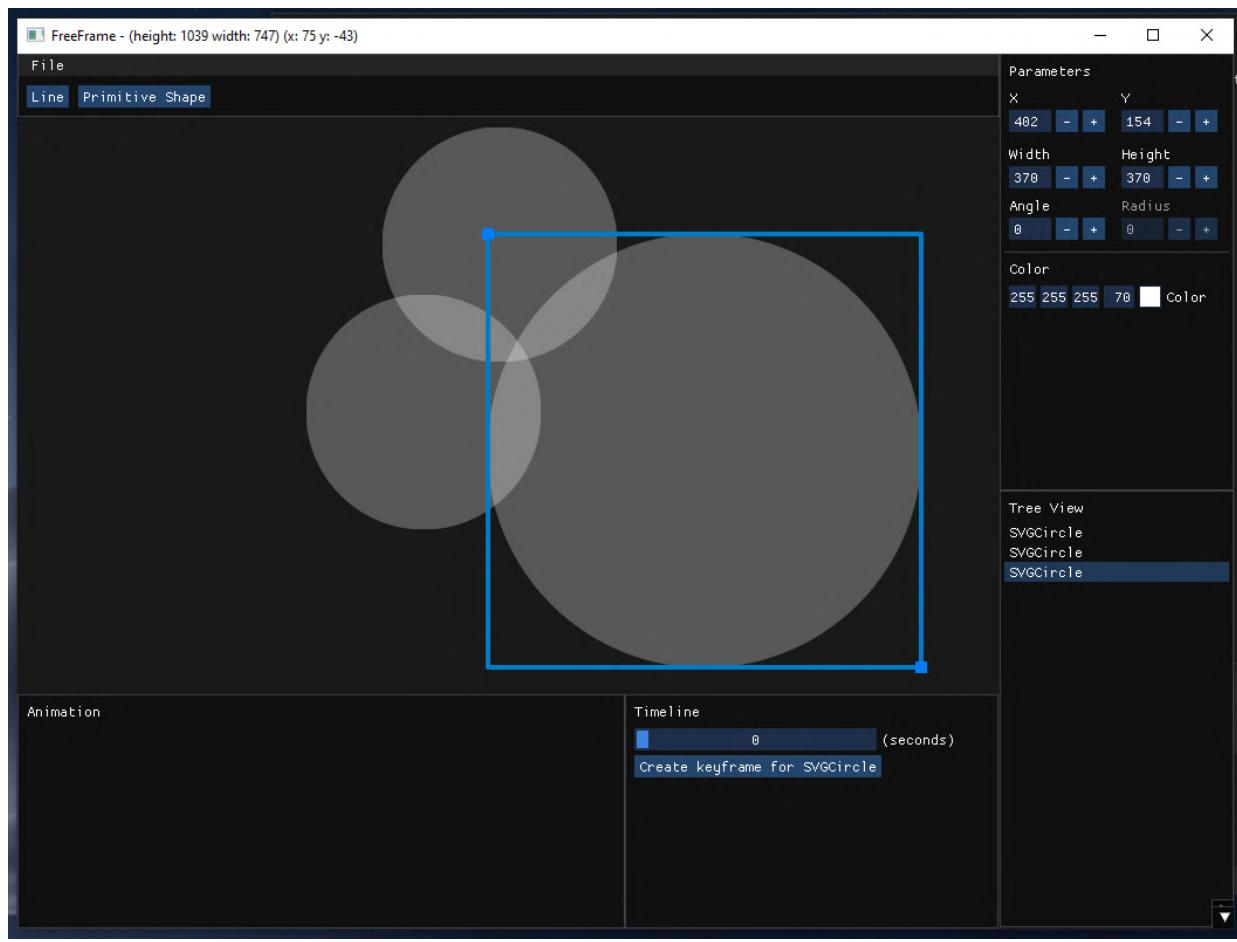


[Fig 22]:Déplacement d'un élément

Le programme autorise une forme à avoir une position négative.

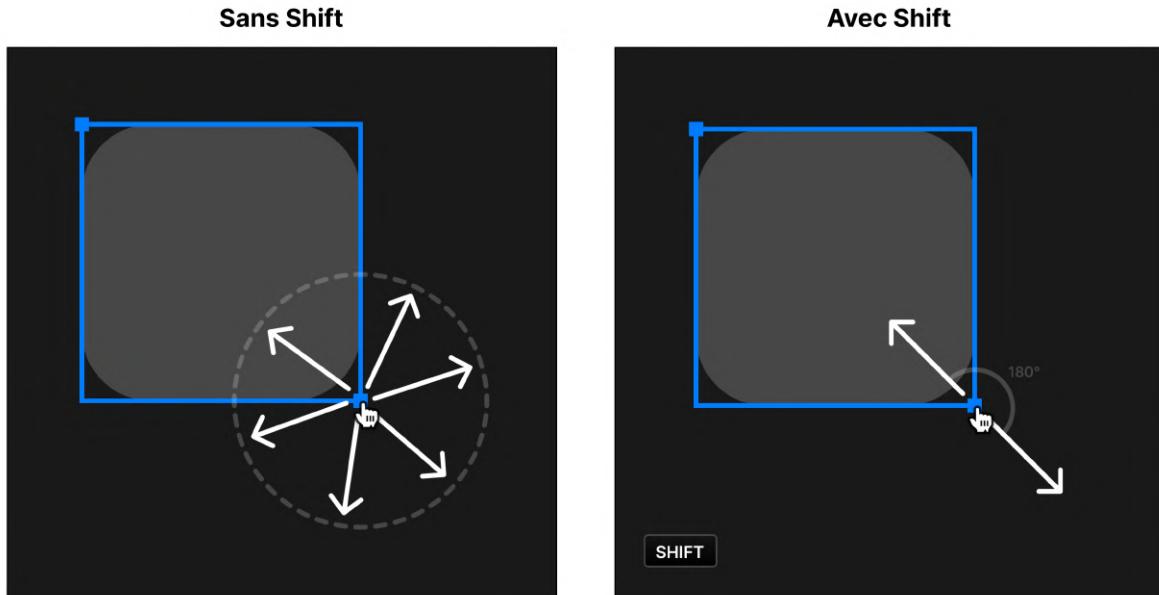
0 →

Tout comme le déplacement, l'utilisateur peut soit passer par l'interface de l'application, soit sélectionner le sélecteur bleu en bas à droite, en faisant un clic gauche puis en glissant avec la souris.



[Fig 23]:Trois ronds superposés, un grand

Le redimensionnement possède également une option d'alignement. Si l'utilisateur appuie sur la touche **Shift** lors du redimensionnement, alors la forme gardera une échelle 1:1.



[Fig 24]:Redimension d'élément

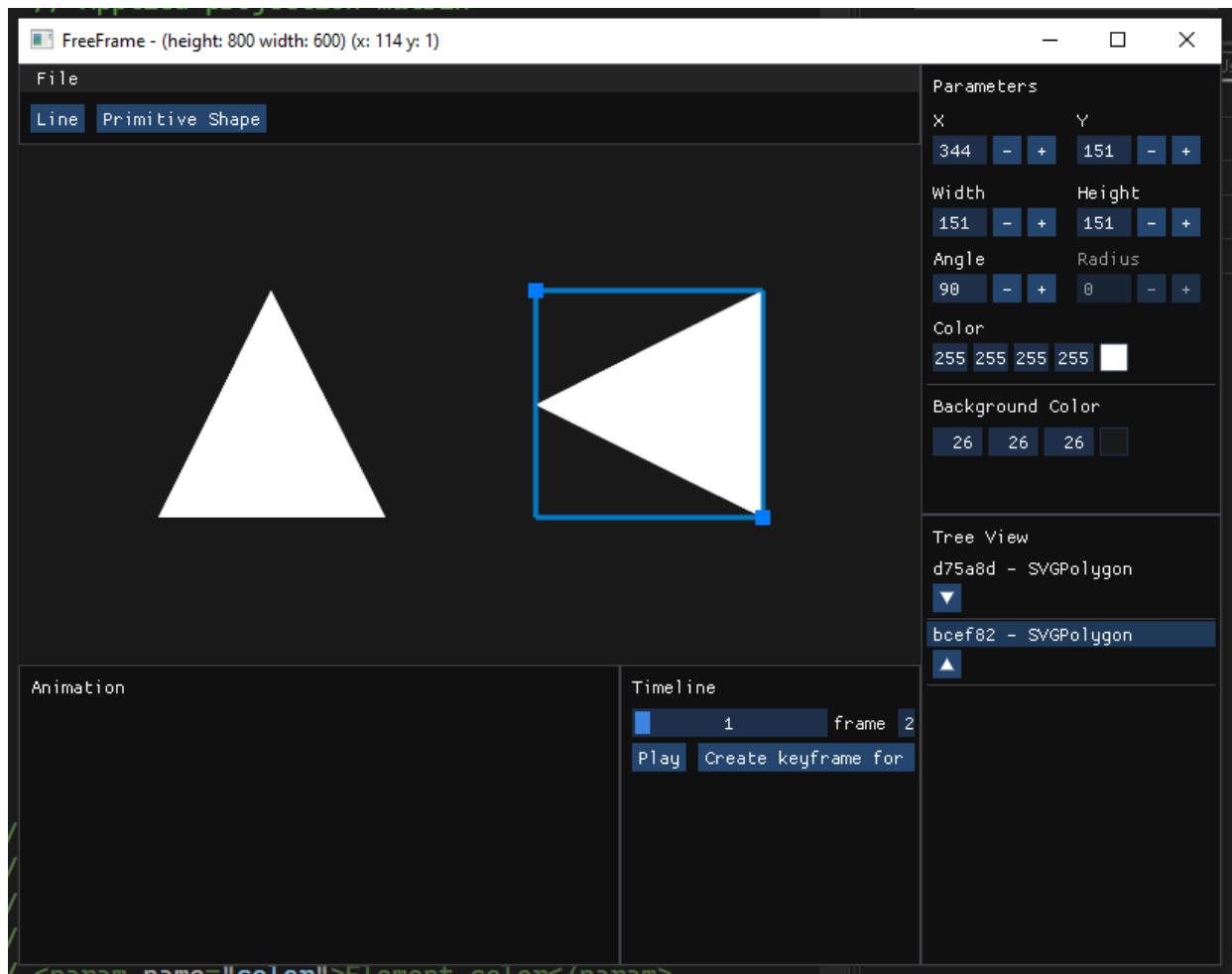
La forme **circle** a cette option activée par défaut, car elle ne doit pas perdre l'intégrité de son échelle.

Il est également important de noter que la forme **path** n'est pas redimensionnables.

Le programme contraint l'utilisateur à avoir une forme de taille positive.

o è →

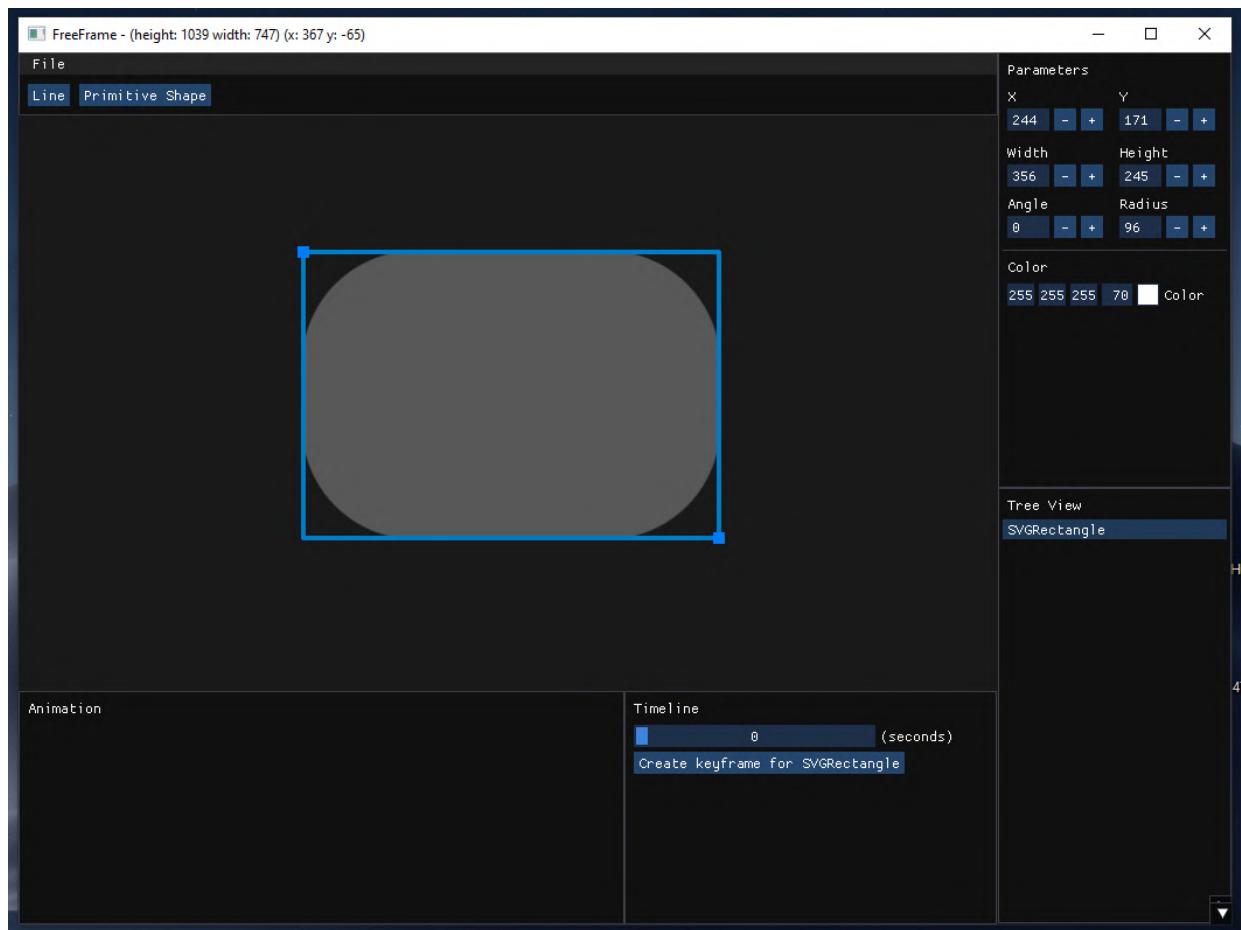
Il est possible de changer l'angle d'une forme. Pour ce faire, il est obligatoire d'utiliser l'interface graphique et de directement changer la valeur depuis le champ *Angle*.



[Fig 25]:Deux triangles, un avec une rotation

S →

Il est possible d'arrondir les bords d'une forme. Pour ce faire, il est obligatoire d'utiliser l'interface graphique et de directement changer la valeur depuis le champ *Radius*.

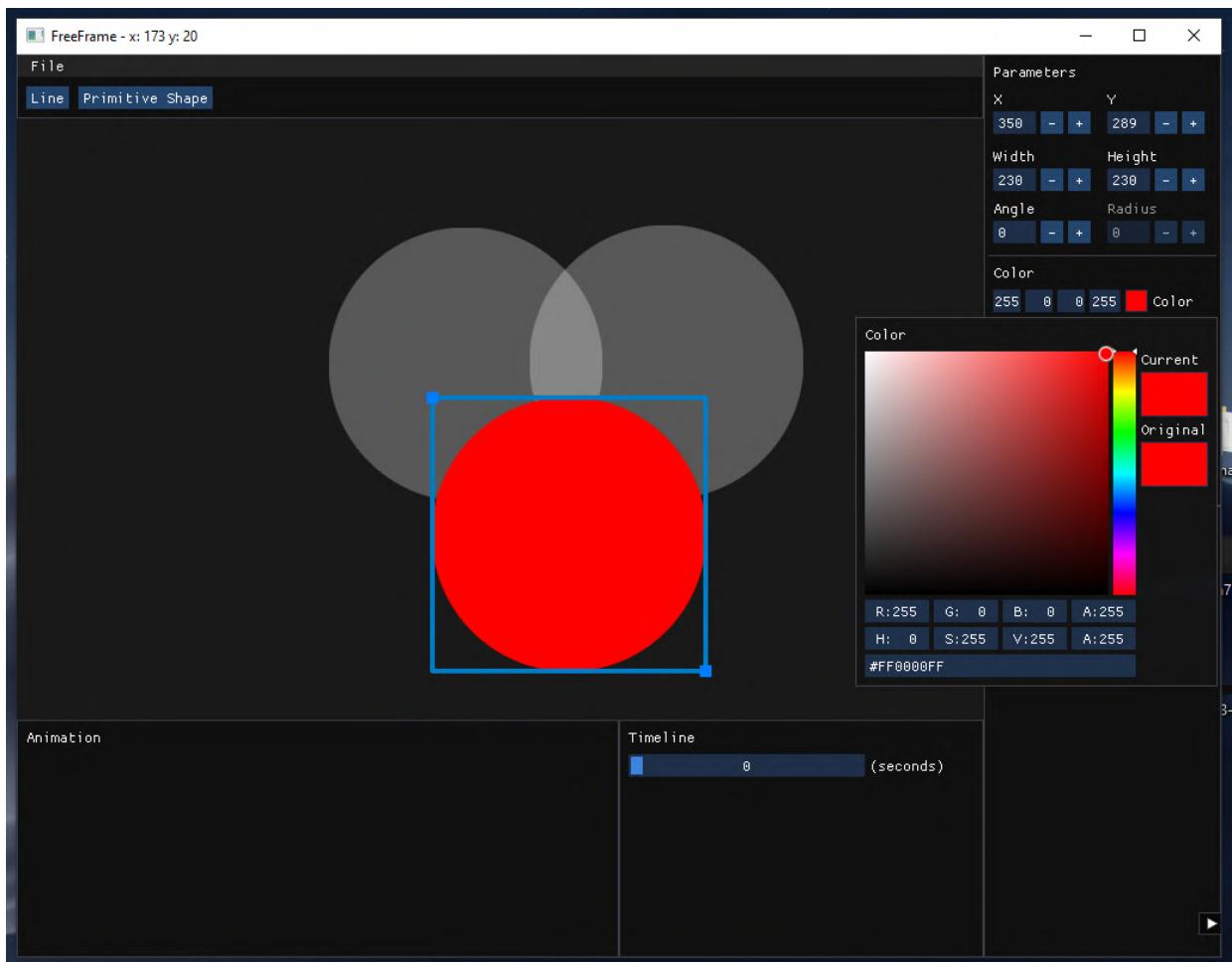


[Fig 26]:Un rectangle arrondis

Cette propriété est uniquement disponible pour la forme **Rectangle**.

V è → ffi

Il est possible de changer la couleur d'une forme depuis l'interface de l'application, soit en utilisant la palette de couleurs, soit en modifiant les champs des formats RGBA, HSVA ou hexadécimal.



[Fig 27]:Trois ronds superposés, un rouge

p ffi

Le Selector est le cadre de sélection qui permet de déplacer et de redimensionner une forme. Pour permettre à la Window de savoir si un clic a été effectué sur le sélecteur ou non, on utilise la méthode `(bool, SelectorType?) HitBox(Vector2i mousePosition)`.

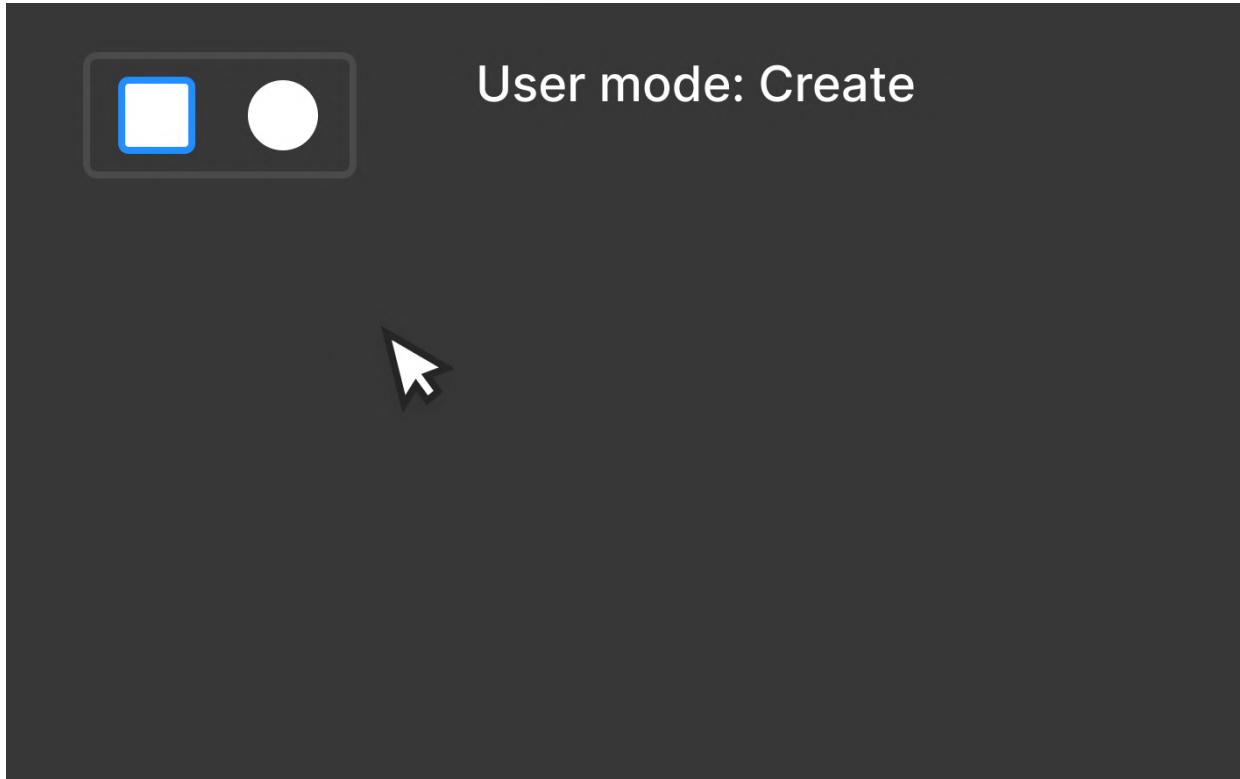
- `bool` -> True si le clic est sur le sélecteur
- `SelectorType?` -> Partie du sélecteur qui a été cliqué. `null` si `bool` vaut `False`.

BBOGV è →

Il est possible de créer les formes suivantes :

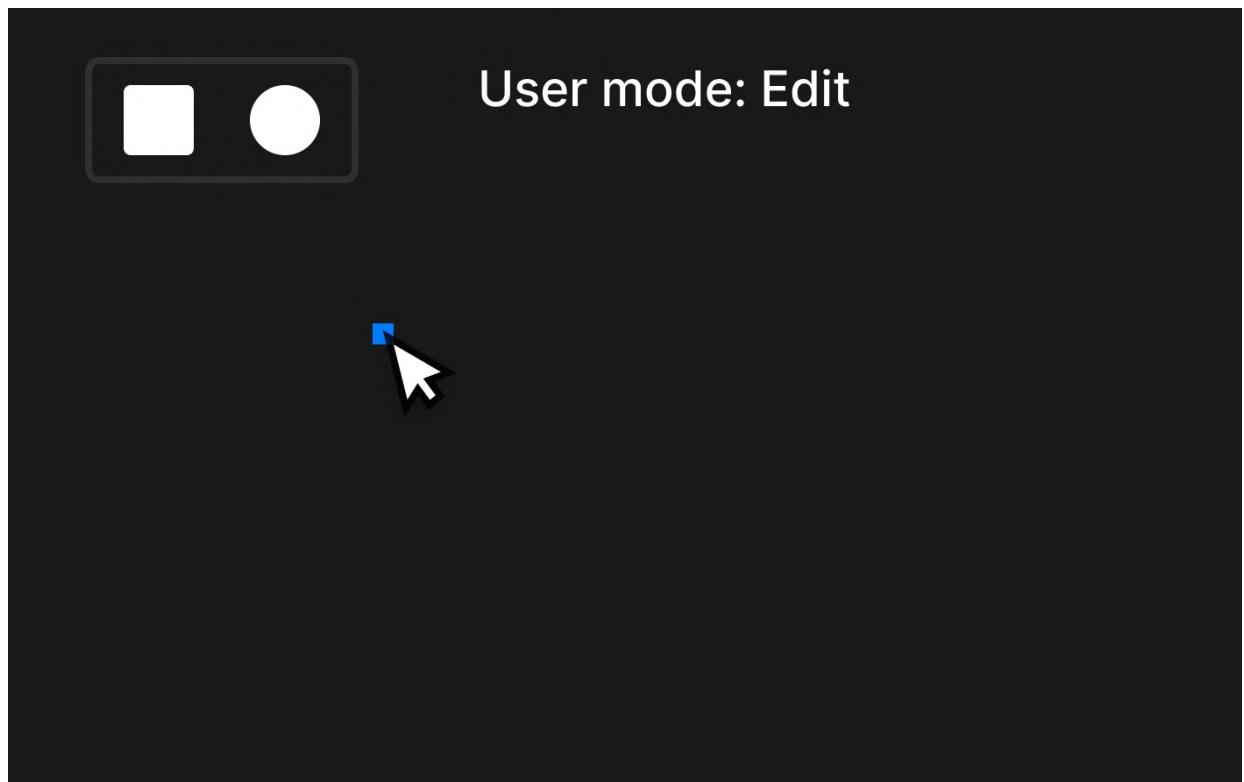
- Ronds ;
- Rectangles ;
- Lignes.

Le choix de la forme créée se fait via l'interface (en dessous de la barre de menu).



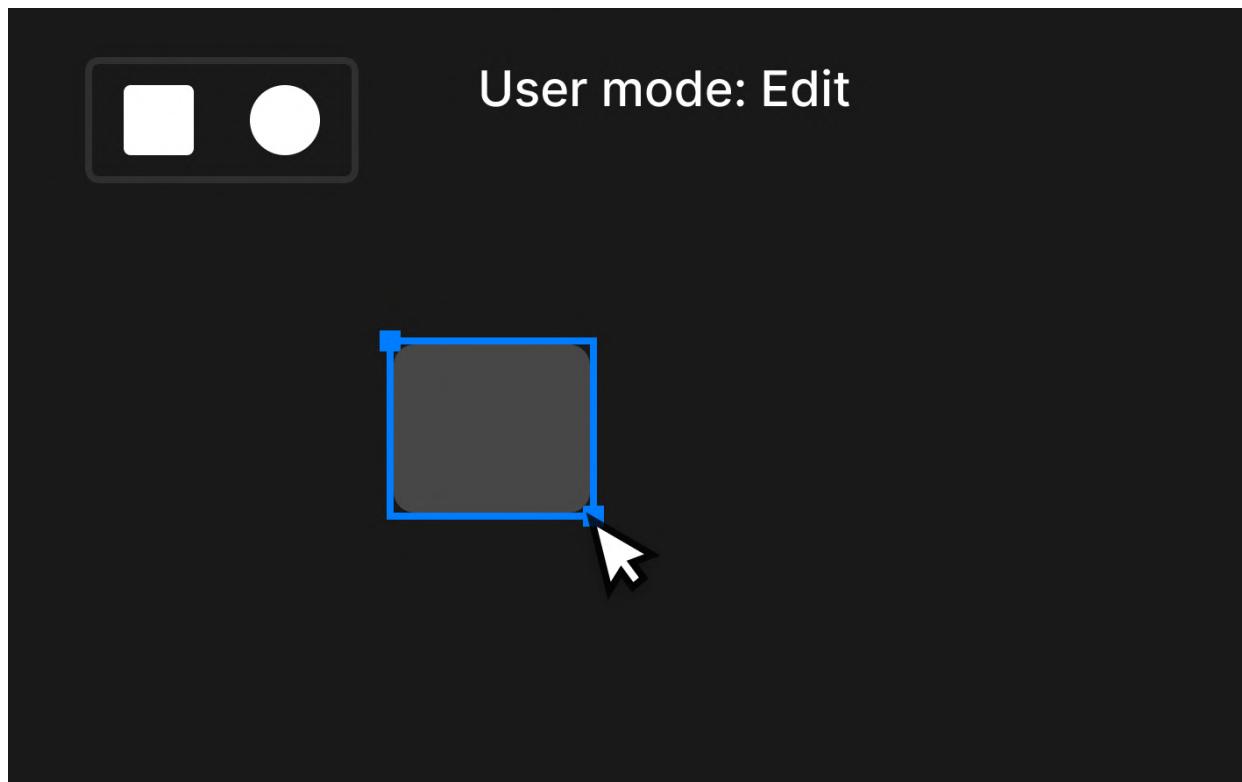
[Fig 28]:Création d'un rectangle, étape une

Une fois la forme voulue choisie, il suffit de faire un clic gauche avec la souris dans l'espace de travail pour la créer. L'application va ensuite passer en mode *Redimension de l'élément*, ce qui permet de directement choisir une taille à sa création.



[Fig 29]:Création d'un rectangle, étape deux

Tout en restant appuyée sur le clic gauche de la souris et en la glissant, la forme se redimensionnera.



[Fig 30]:Création d'un rectangle, étape trois

B9BG9Hq

La timeline est le composant qui permet d'animer n'importe quels éléments.

Elle peut sauvegarder l'état d'une forme à un élément donné - l'état d'une forme correspond aux propriétés qu'elle possède, ex : position, taille, couleur.

En sauvegardant l'état d'une forme à plusieurs moments, l'utilisateur peut ensuite l'animer en déplaçant la timeline.

g è

Une keyframe est simplement la sauvegarde de l'état d'une forme à un moment donné.

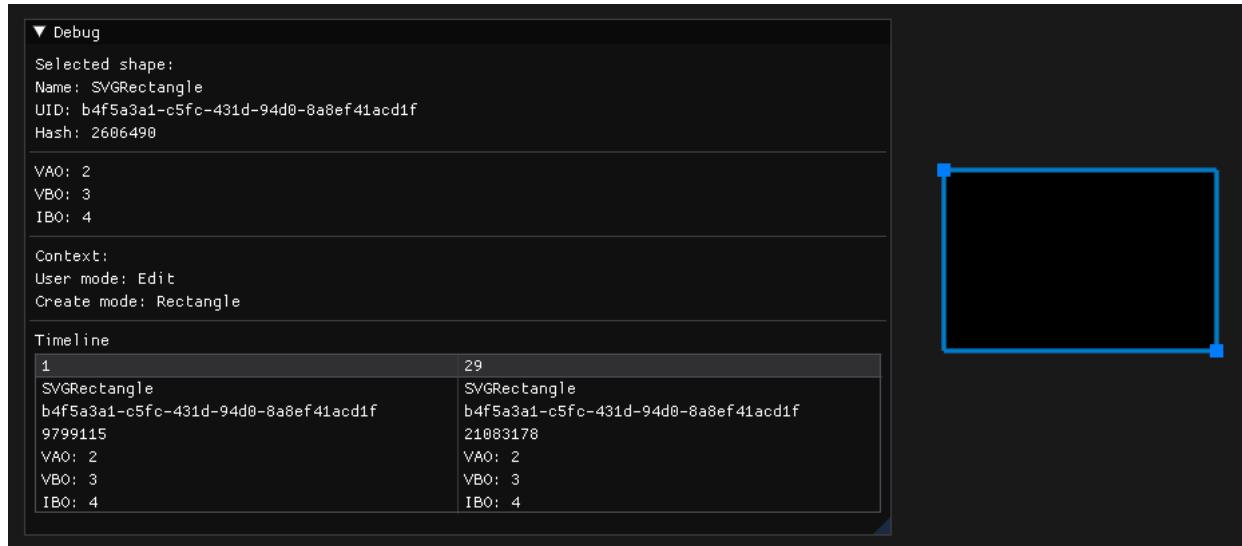
h

Il est possible de démarrer l'avancement de la timeline en appuyant sur le bouton *Play*. La timeline à un nombre d'images fixe (100 images). Il est cependant possible de changer les fps. (entre 1 et 120).

Par défaut, la timeline va continuer en boucle. Lorsqu'elle atteint la centième image, elle redémarre sur la première.

À noter que l'interpolation se fait uniquement entre des keyframes du même tour.

B9BG9 | → → ff



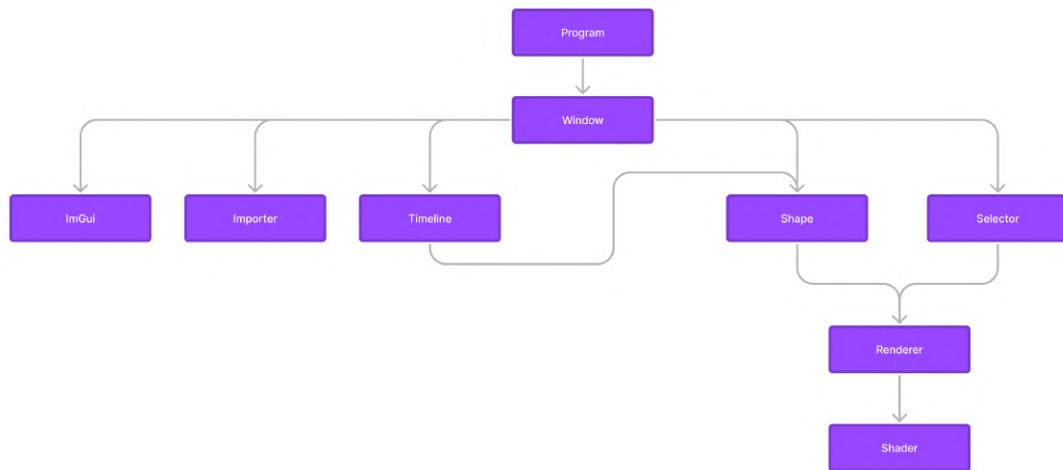
[Fig 31]:Outil de debug

L'outil de debug permet d'avoir rapidement les informations internes essentielles sur les formes. Il affiche la liste des identifiants que l'application lui a accordés, mais également ceux qu'OpenGL lui a accordés.

On peut aussi voir les différents éléments qu'il y a dans la timeline et leurs identifiants. Ça permet par exemple de savoir si OpenGL utilise bien le même id pour une forme qui a plusieurs états.

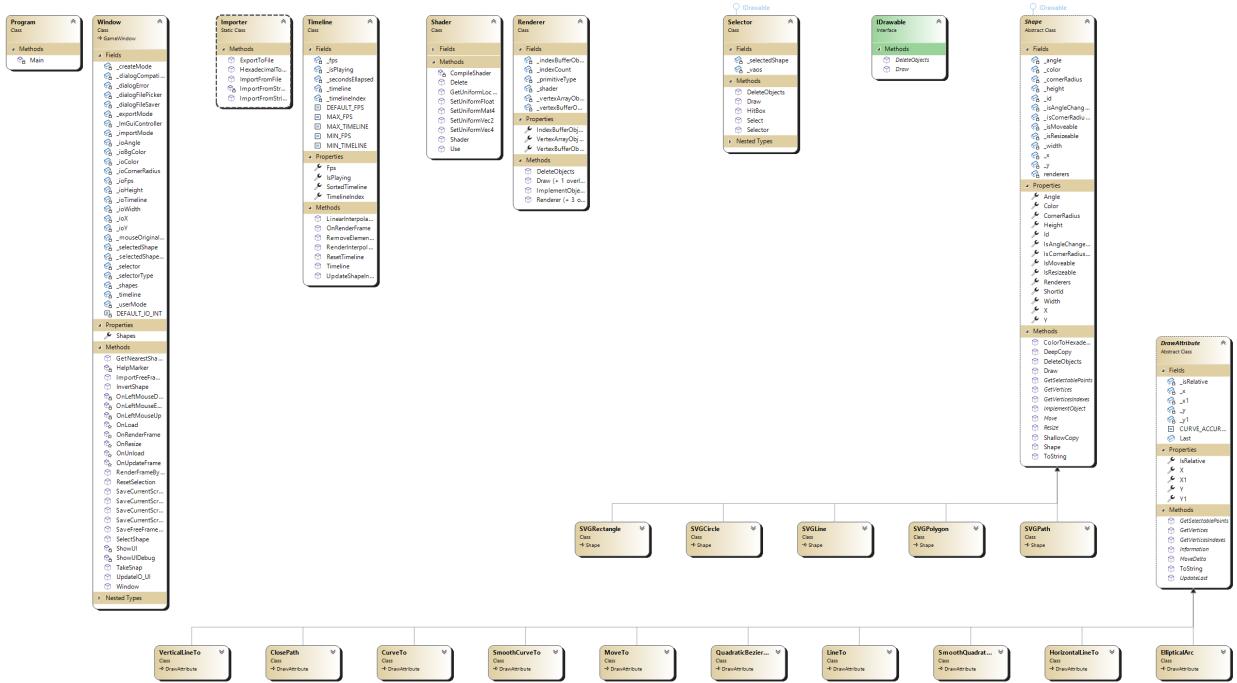
B9H S è è

B9H S ffi ffi



[Fig 32]:Architecture de l'application

Z è è → ffiè



[Fig 33]:Diagramme de classe

B9H9C e

L'importation se fait via l'utilisation de la classe **Importer**.

Chaque élément contenu dans la balise `<svg></svg>` qui est valide (qui est connu de l'application) est considéré comme une forme, soit :

- <polyline> ;
 - <path> ;
 - <rect> ;
 - <circle> ;
 - <line> .

b ffi è ff

V ffi

Catégorie : [basic-shapes](#)

```
<svg xmlns="http://www.w3.org/2000/svg">
  <circle cx="50" cy="50" r="50"/>
</svg>
```

cx : La coordonnée du centre du cercle sur l'axe x. *Type : entier. Valeur par défaut : 0.*

cy : La coordonnée du centre du cercle sur l'axe y. *Type : entier. Valeur par défaut : 0.*

r : Le rayon du cercle *Type : entier. Valeur par défaut : 0.*

fill : Couleur d'affichage du cercle. *Type : hexadécimal. Valeur par défaut : #000000FF.*

h

Catégorie : [basic-shapes](#)

```
<svg xmlns="http://www.w3.org/2000/svg">
  <line x1="0" y1="80" x2="100" y2="20"/>
</svg>
```

x1 : Définit la coordonnée sur l'axe x du point de départ de la ligne. *Type : entier. Valeur par défaut : 0.*

x2 : Définit la coordonnée sur l'axe x du point de fin de la ligne. *Type : entier. Valeur par défaut : 0.*

y1 : Définit la coordonnée sur l'axe y du point de départ de la ligne. *Type : entier. Valeur par défaut : 0.*

y2 : Définit la coordonnée sur l'axe y du point de fin de la ligne. *Type : entier. Valeur par défaut : 0.*

fill : Couleur d'affichage de la ligne. *Type : hexadécimal. Valeur par défaut : #000000FF.*

o ffiè

Catégorie : [basic-shapes](#)

```
<svg xmlns="http://www.w3.org/2000/svg">
  <!-- Simple rectangle -->
  <rect width="100" height="100" />
```

```
<!-- Rounded corner rectangle -->
<rect x="120" width="100" height="100" rx="15" />
</svg>
```

x : La coordonnée x du rectangle. *Type : entier. Valeur par défaut : 0.*

y : La coordonnée y du rectangle. *Type : entier. Valeur par défaut : 0.*

width : La largeur du rectangle. *Type : entier. Valeur par défaut : 0.*

height : La hauteur du rectangle. *Type : entier. Valeur par défaut : 0.*

rx : L'arrondi des coins du rectangle. *Type : entier. Valeur par défaut : 0.*

ry : L'arrondi des coins du rectangle. *Type : entier. Valeur par défaut : 0.*

La valeur la plus grande entre rx et ry est attribuée pour le l'arrondi des coins du rectangle.

fill : Couleur d'affichage du rectangle. *Type : hexadécimal. Valeur par défaut : #000000FF.*

S ffi è ff

mè

```
<svg xmlns="http://www.w3.org/2000/svg">
  <path d="M 10,30
    A 20,20 0,0,1 50,30
    A 20,20 0,0,1 90,30
    Q 90,60 50,90
    Q 10,60 10,30 z"/>
</svg>
```

d : Défini la trajectoire du chemin. *Type : chaîne de caractère. Valeur par défaut : ""*

fill : Couleur d'affichage du chemin. *Type : hexadécimal. Valeur par défaut : #000000FF.*

→

La différence entre les commandes en majuscule et minuscule (ex : M et m) est qu'en minuscule les positions des commandes deviennent relatives alors qu'avec une majuscule elles sont absolues.

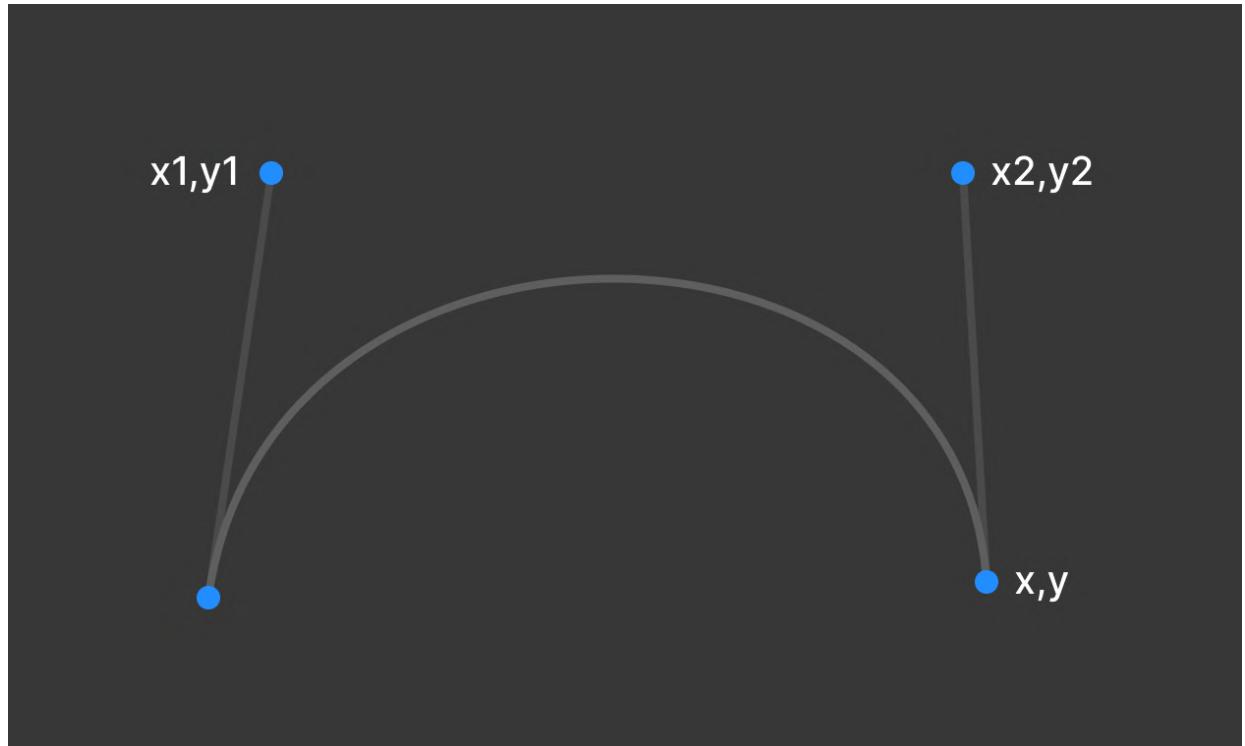
M ou m : Déplacement du curseur vers la position donnée. *Format : (x, y).*

L ou l : Dessine une ligne entre la position du curseur et la position donnée. *Format : (x, y).*

H ou h : Dessine une ligne horizontale entre la position du curseur et la position x données. Format : (x).

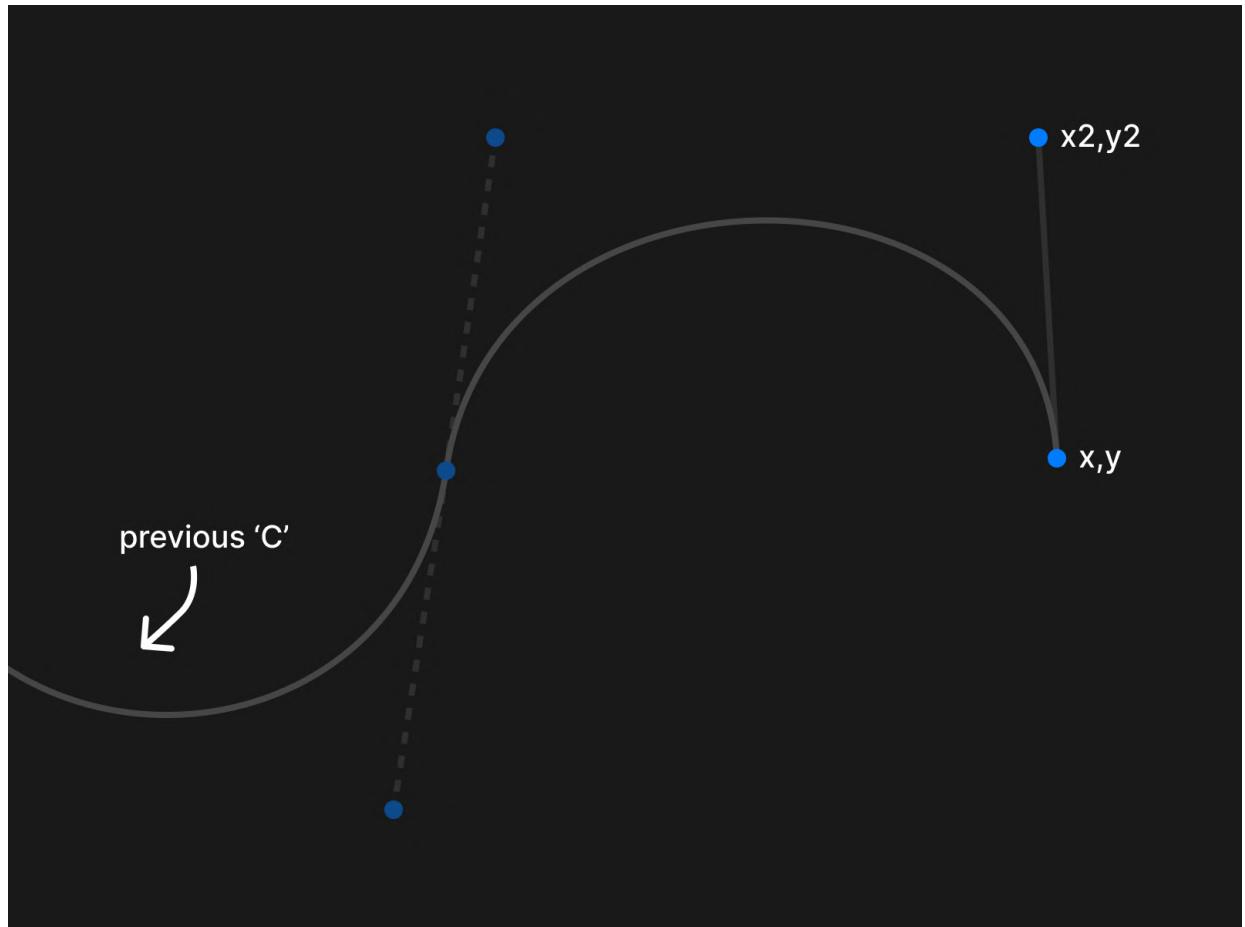
V ou v : Dessine une ligne verticale entre la position du curseur et la position y donnée. Format : (x).

C ou c : Dessine une courbe de Bézier cubique. La position du curseur correspond au point de départ, la position finale est spécifiée par (x,y). Le point de contrôle de départ est spécifié par (x1,y1) et le point de contrôle final est spécifié par (x2,y2). Format : (x1,y1,x2,y2,x,y).



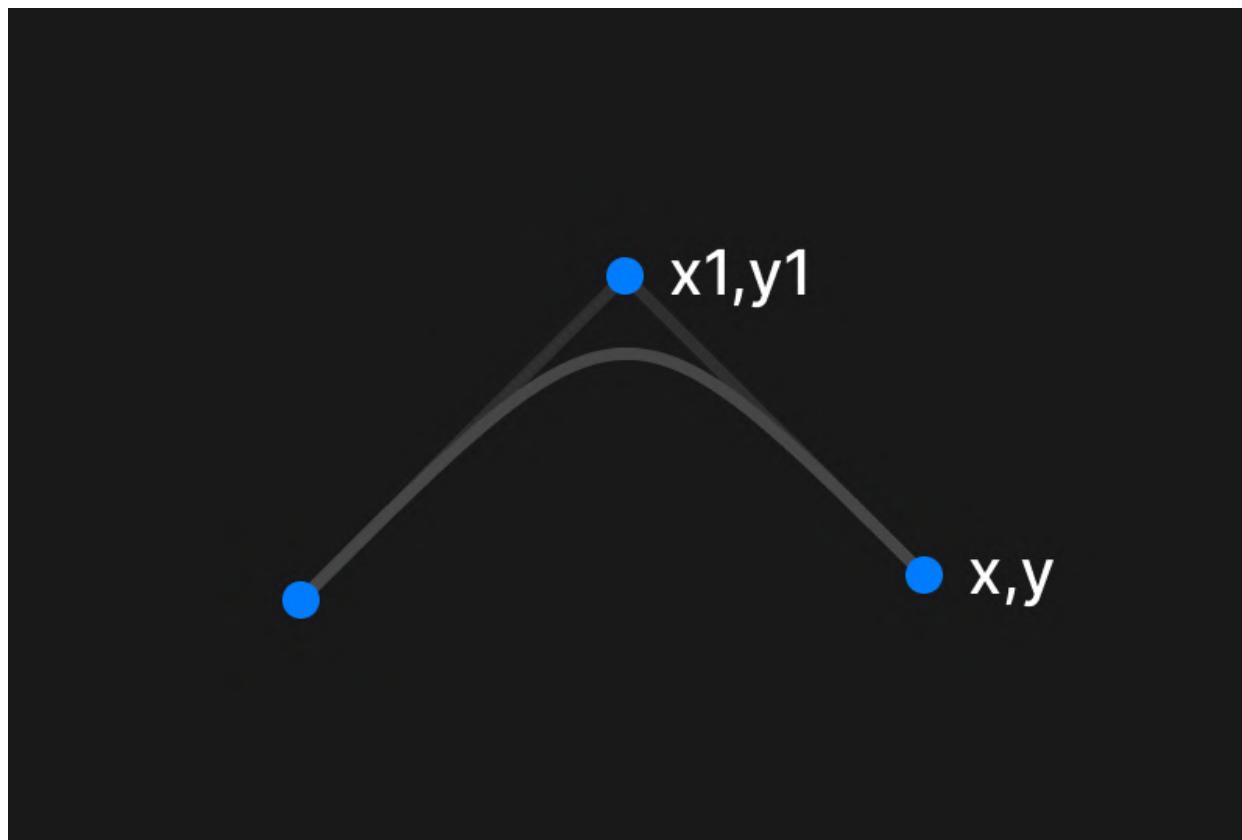
[Fig 34]:Courbe de Bézier cubique

S ou s : Dessine une courbe de Bézier cubique. La position du curseur correspond au point de départ, la position finale est spécifiée par (x,y). Le point de contrôle de départ est spécifié par le point de contrôle de la courbe de Bézier cubique précédente et le point de contrôle final est spécifié par (x2,y2). Format : (x2,y2,x,y).



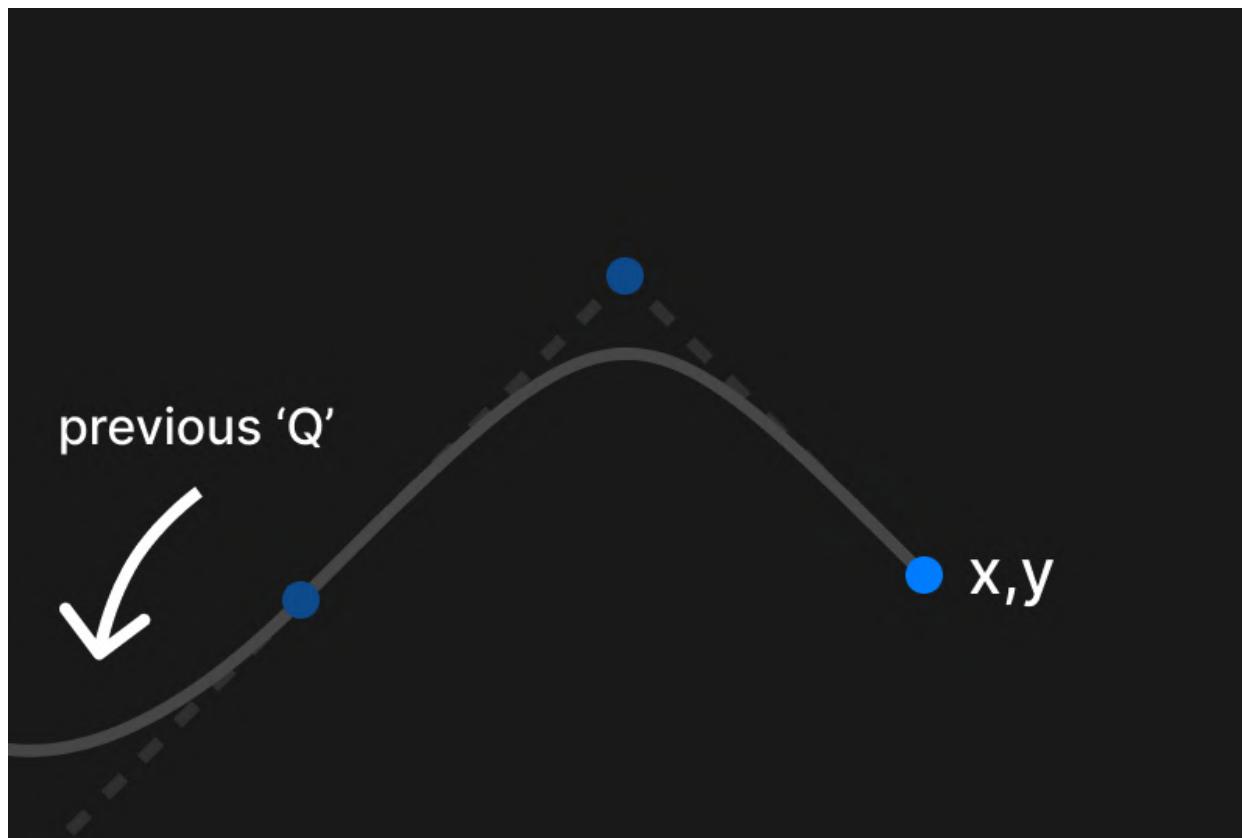
[Fig 35]: Courbe de Bézier cubique dépendante

Q ou q : Dessine une courbe de Bézier quadratique. La position du curseur correspond au point de départ, la position finale est spécifiée par (x,y) . Le point de contrôle de départ et le point de contrôle final est spécifié par $(x1,y1)$. Format : $(x2,y2,x,y)$.



[Fig 36]:Courbe de Bézier quadratique

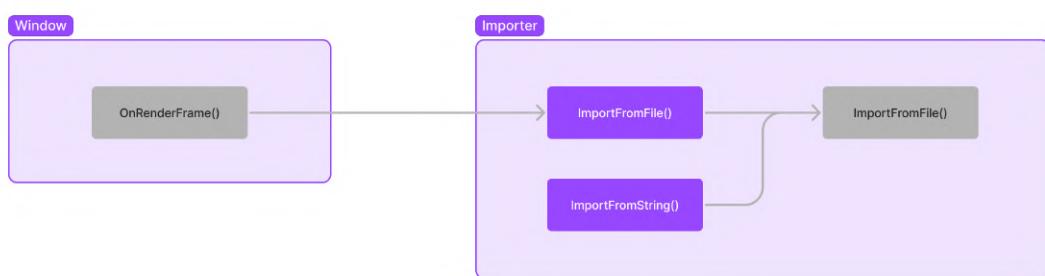
Tout : Dessine une courbe de Bézier quadratique. La position du curseur correspond au point de départ, la position finale est spécifiée par (x,y) . Le point de contrôle de départ et le point de contrôle final sont spécifiés par celui de la courbe de Bézier quadratique précédente. Format : (x,y) .



[Fig 37]:Courbe de Bézier quadratique dépendante

Z ou z : Dessine une ligne entre la position du curseur et la position de départ du dessin.

Z ffi



[Fig 38]:Architecture d'import d'un fichier

```
# Importer.cs
(List<Shape>, SortedDictionary<int, List<Shape>>, bool) ImportFromStream(Stream
pStream)
```

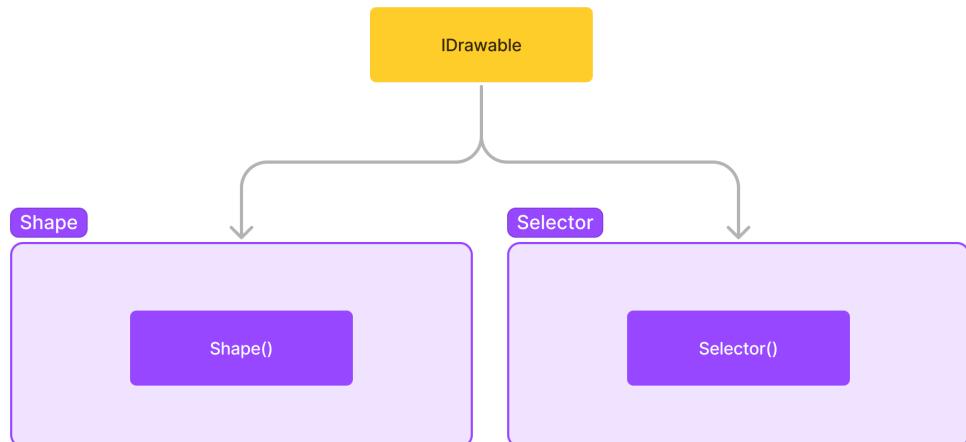
Cette méthode permet de récupérer l'ensemble des formes contenu dans une séquence d'octets ainsi que leurs différents états possibles.

Le `Stream` est ensuite lu à l'aide d'un `XmlReader`.

- `List<Shape>` La liste des formes principales/mères trouvées dans le fichier
- `SortedDictionary<int, List<Shape>>` -> Contient la liste de chaque forme enfants pour chaque seconde de la timeline
- `bool` -> Flag qui permet de savoir si un des éléments n'est pas compatible

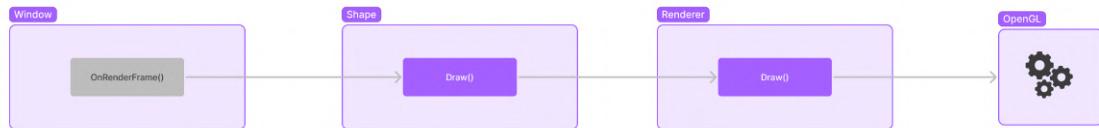
o è V

p è



[Fig 39]:Architecture des classes de `IDrawable`

Les classes qui peuvent être affichées à l'écran doivent implémenter l'interface **IDrawable**, la classe **Shape** en fait partie.



[Fig 40]:Architecture d'appel d'OpenGL

Le rendu des formes est déclenché par `OnRenderFrame()`, qui est appelé par la boucle interne d'OpenTK.

Le seul prérequis pour qu'une forme soit affichée est qu'elle doit être au préalable implémentée (en utilisant `ImplementObject()`). Ensuite, pour l'afficher, il suffit d'appeler la méthode `Draw()` qui lui a été hérité par **Shape**.

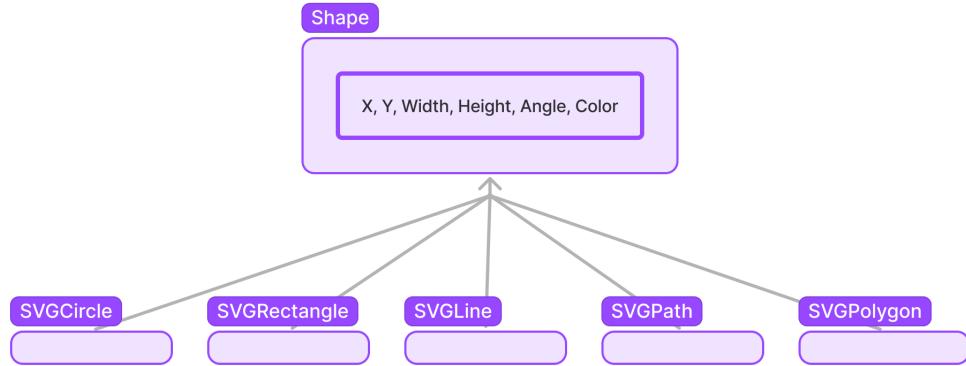
La méthode `Draw()` de **Shape** va ensuite appeler la méthode `Draw()` des **Renderers**.

Une forme peut posséder plusieurs **Renderers** car ça peut être un mélange de formes. Par exemple, avec une seule forme `SVGPath` il est à la fois possible d'avoir des courbes de Bézier et des lignes qui sont indépendantes les unes des autres.

La méthode `Draw()` de **Renderer** va permettre de :

- Dire à OpenGL quels shaders utiliser
- Remplir les uniforms
- Dire à OpenGL quel *Vertex Array* utiliser
- Dire à OpenGL de dessiner la forme

Chaque forme SVG est représentée par une classe qui hérite de la classe **Shape**.



[Fig 41]:Architecture de Shape

La classe abstraite **Shape** possède les propriétés **X**, **Y**, **Width** et **Height** qui sont communes à toutes les formes.

Le but du constructeur de chaque forme est de convertir les propriétés SVG en ces quatre valeurs.

Shape possède d'autres valeurs, mais qui ne déterminent cependant pas ni la position et ni de la taille d'une forme :

- **Id** -> Identifiant permettant de retrouver les formes similaires dans la timeline (car la timeline fait une copie de la forme, le seul moyen de faire le lien entre les différentes instances est cet identifiant)
- **Renderers** -> Liste des éléments OpenGL nécessaires à son affichage
- **Color** -> Couleur
- **Angle** -> Rotation en degré
- **CornerRadius** -> Arrondit des bordures
- **IsMoveable** -> Détermine si oui ou non elle peut être déplacée
- **IsResizeable** -> Détermine si oui ou non elle peut être redimensionnée
- **IsAngleChangeable** -> Détermine si oui ou non l'on peut changer l'angle de rotation
- **IsCornerRadiusChangeable** -> Détermine si oui ou non l'on peut changer l'arrondi des bordures

psc h

Les propriétés SVG d'une ligne primitives sont :

- `x1` et `y1` Position de départ de la ligne
- `x2` et `y2` Position d'arrivée de la ligne

Cependant, étant donné que les uniques propriétés d'un élément **Shape** sont **X**, **Y**, **Width** et **Height**, une conversion est nécessaire.

X et **Y** prennent simplement les valeurs de `x1` et `y1`.

Width correspond à la distance entre `x1` et `x2`.

Height correspond à la distance entre `y1` et `y2`.

psc o ffiè

Les propriétés SVG d'un rectangle simple sont :

- `x` ;
- `y` ;
- `width` ;
- `height` .

Elles correspondent exactement avec les propriétés possibles d'un élément **Shape** donc il n'y a rien à changer.

psc V ffi

Les propriétés SVG d'un cercle sont :

- `r` -> rayon
- `CX` -> position x du centre du cercle
- `CY` -> position y du centre du cercle

X correspondra à la position x du centre du cercle moins le rayon, soit `CX - r`. **Y** correspondra à la position y du centre du cercle moins le rayon, soit `CY - r`. **Width** vaudra le diamètre, soit deux fois le rayon. **Height** vaudra la même valeur que **Width**.

psc mè

Uniquement la propriété `d` de l'élément `<path>` détermine son parcours.

La propriété `d` possède elle-même plusieurs propriétés (elles sont énumérées en détail dans la partie *Formes primitives compatibles*).

Pour sa lecture, il est obligatoire d'utiliser des [expressions régulières](#).

```
# SVGPath.cs
readonly Dictionary<char, Regex> _dAttributeRegex = new()
{
    { 'm', new Regex(@" *(-?\d+) *, *(-?\d+) *") },
    { 'l', new Regex(@" *(-?\d+) *, *(-?\d+) *") },
    { 'h', new Regex(@" *(-?\d+) *") },
    { 'v', new Regex(@" *(-?\d+) *") },
    { 'c', new Regex(@" *(-?\d+) *, *(-?\d+) +(-?\d+) *, *(-?\d+) +(-?\d+) *, *(-?\d+) *") },
    { 's', new Regex(@" *(-?\d+) *, *(-?\d+) +(-?\d+) *, *(-?\d+) *") },
    { 'q', new Regex(@" *(-?\d+) *, *(-?\d+) +(-?\d+) *, *(-?\d+) *") },
    { 't', new Regex(@" *(-?\d+) *, *(-?\d+) *") },
    { 'a', new Regex(@" *(-?\d+) +(-?\d+) +(-?\d+) +(-?\d+) +(-?\d+) +(-?\d+) *, *(-?\d+) *") },
    { 'z', new Regex("") },
};
```

Par la suite, pour chaque propriété, une nouvelle classe héritière de **DrawAttribute** est instancié.

psc m

Tout comme le `<path>`, l'élément SVG `<polygon>` possède également un attribut de texte complexe, le `points`.

Il est donc aussi nécessaire d'utiliser les expressions régulières pour en récupérer ses points.

```
# SVGPolygon.cs
readonly Regex _pointsAttributeRegex = new(@" *(\d+) *, *(\d+) *");
```

o è | c h

La classe **Shape** oblige à ses héritiers d'implémenter différentes méthodes qui permettent le bon fonctionnement d'OpenGL :

- `float[] GetVertices()` -> Rend la liste des sommets de la forme. Généralement en se basant uniquement sur **X**, **Y**, **Width** et **Height**.
- `uint[] GetVerticesIndexes()` -> Rend la liste des identifiants correspondant à chaque type primitif d'OpenGL (triangle, lignes).
- `ImplementObjects()` -> Implémenter la forme pour OpenGL.

psc h

```
# SVGLine.cs
public override float[] GetVertices() => new float[] { X, Y, X + Width, Y +
Height };
public override uint[] GetVerticesIndexes() => new uint[] { 0, 1 };
```

- `GetVertices()` -> Il existe un type **primitif ligne** dans OpenGL, il suffit donc de rendre en tant que sommet les positions des deux points.

psc o ffiè

```
# SVGRectangle
public override float[] GetVertices() => new float[] { X, Y, X + Width, Y, X +
Width, Y + Height, X, Y + Height };
public override uint[] GetVerticesIndexes() => new uint[] { 0, 1, 2, 0, 2, 3 };
```

- `GetVertices()` -> Il n'existe pas de type primitive rectangle sur OpenGL, pour créer un rectangle, il est obligatoire de le créer à partir de deux triangles.

psc V ffi

- `GetVertices()` -> Le cercle fonctionne de la même façon que le rectangle (car c'est uniquement du côté du *Fragment Shader* que le cercle ressemble à un cercle).

psc m

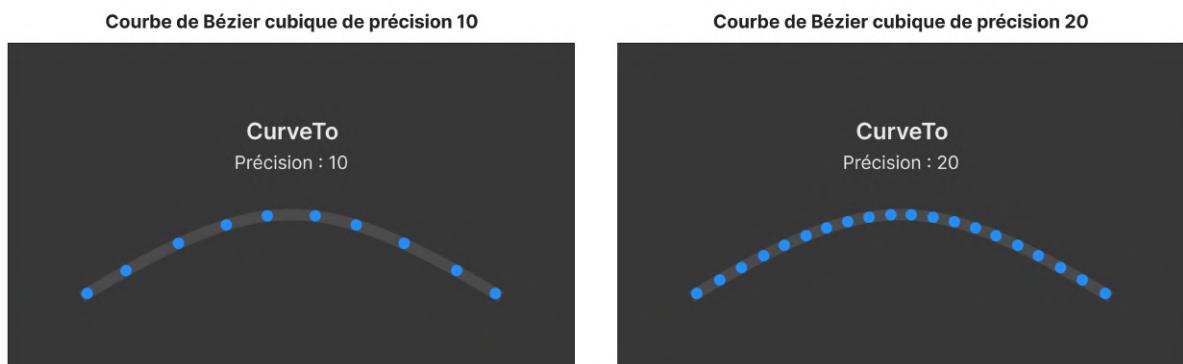
- `GetVertices()` -> Tout comme `SVGLine`, `SVGPolygon` utilise le type primitif ligne, en rendant la liste entière des points qu'il possède.

psc mè

`GetVertices()` et `GetVerticesIndexes()` ne sont pas utilisés, car la propriété `d` est trop complexe, ce sont donc directement les propriétés de `d` (soit les classes qui héritent de `DrawAttribute`) qui possède ces deux méthodes.

Par exemple, la méthode `GetVertices()` de la classe `CurveTo` (qui hérite de `DrawAttribute`), utilise la fonction mathématique des [courbes de Bézier cubiques](#).

Pour dessiner la courbe de `CurveTo`, on utilise une liste de points qui vont être lus et rendus comme des primitifs de type ligne. Le nombre de points choisi est ce qu'on appelle la précision de la courbe, ça permet de déterminer le nombre de lignes l'on souhaite pour sa courbe.



[Fig 42]: Courbe de Bézier cubique précisions

Le principe est exactement le même pour toutes les propriétés de courbes et de lignes dans la propriété `d`.

e | ff ffi 34

Cette méthode implémente la forme pour OpenGL. Elle prend en paramètre les sommets (via `GetVertices()`) et les identifiants (via `GetVerticesIndexes()`). Elle est structurée en six étapes :

1. Dire à OpenGL quel *VertexArray* utiliser
2. Dire à OpenGL quel *Buffer* utiliser pour le *Vertex Buffer*
3. Envoyer les données du *Vertex Buffer*
4. Dire à OpenGL quel *Buffer* utiliser pour l'*Index Buffer*
5. Envoyer les données de l'*Index Buffer*
6. Dire à OpenGL comment lire les données

Le *Vertex Array* est l'identifiant principal qui va lier le *Vertex Buffer* et l'*Index Buffer*.

p è→

s è→

Le *Vertex Shader* est le shader qui va être utilisé par le GPU pour la position des sommets d'une forme sur l'écran. Il y a uniquement un seul *Vertex Shader* pour la simple raison qu'il n'y avait pas besoin d'avoir une disposition des sommets différent suivant les formes.

```
#version 330 core

layout(location = 0) in vec2 position;

uniform mat4 u_Model_To_NDC;
uniform mat4 u_Transformation;

uniform vec2 u_Resolution;
uniform vec2 u_Position;
uniform vec2 u_Size;

void main()
{
    vec4 finalPosition = vec4(position, 1.0, 1.0);

    vec4 origin = vec4(u_Position.x + u_Size.x/2.0, u_Position.y + u_Size.y/2.0,
0.0, 0.0);

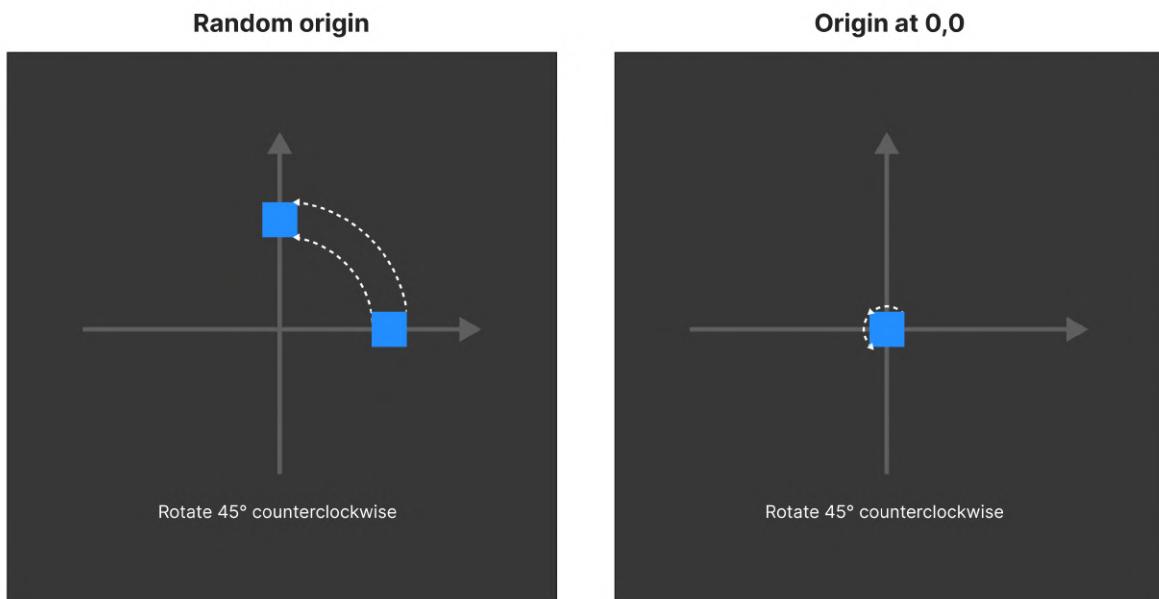
    finalPosition -= origin;
```

```

finalPosition = finalPosition * u_Transformation;
finalPosition += origin;
finalPosition = u_Model_To_NDC * finalPosition;
gl_Position = finalPosition;
}

```

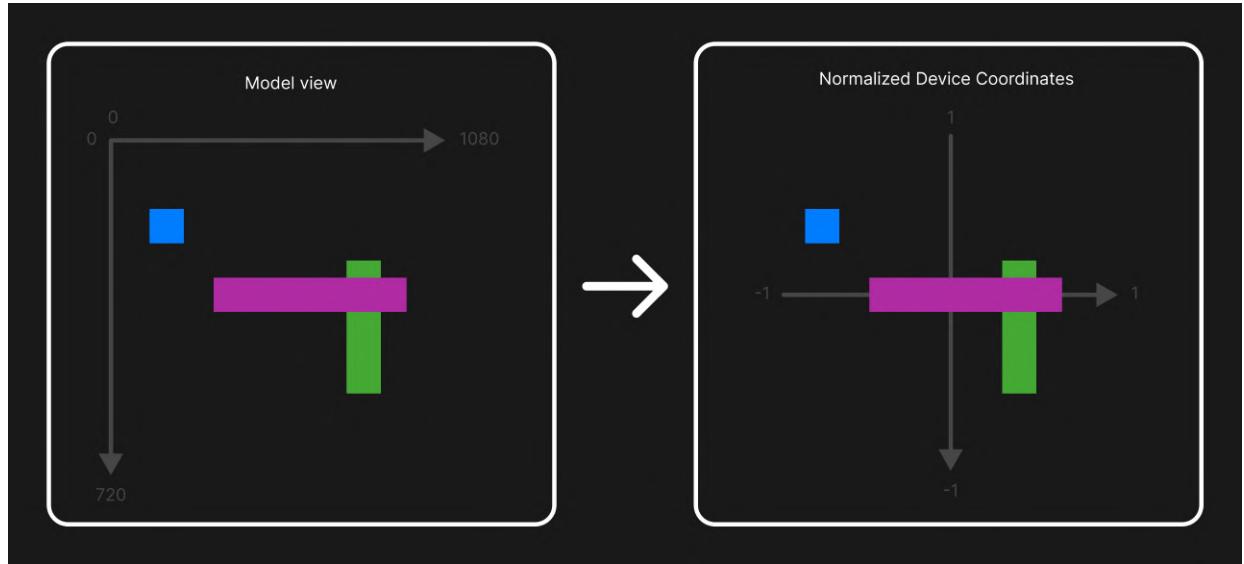
Les transformations se font via l'utilisation de matrices qui sont passées grâce à des *uniforms*. Dans ce shader on peut voir que l'origine est soustraite, que la transformation est effectuée et que l'origine est remise. Cela permet que les matrices de rotation ne déplacent pas la forme, comme le montre ce schéma ci-dessous :



[Fig 43]:Rotation origine comparaison

En [GLSL](#), un *uniform* est simplement une méthode pour transmettre une valeur du CPU au GPU.

i → j ZV



[Fig 44]:Conversion model view to NDC

Au niveau de l'utilisateur, les valeurs des différents champs sont représentées en pixels. Cependant, sur OpenGL, il faut que ces données soient converties au format NDC. Pour ce faire, une matrice avec la méthode `CreateOrthographicOffCenter()` permet de faire cette conversion.

b è è→

Le *Fragment Shader* est le shader qui va être utilisé par le GPU au moment de l'affichage des pixels. Il est exécuté autant de fois que la forme possède de pixel. Il en existe un par défaut qui va simplement remplir la zone de pixel que la forme possède avec la couleur demandée. Celui par défaut est par exemple utilisé pour les lignes et les triangles.

V ffi

Le shader `circle.frag` permet de gérer l'affichage des cercles dans OpenGL. Pour rendre un cercle à l'écran, l'utilisation d'un shader a été préféré. Une autre méthode possible aurait été de créer un cercle à partir de plusieurs triangles, mais cette méthode est plus complexe en termes d'implémentation et de puissance de calcul.

o ffiè

Le shader `rect.frag` permet de gérer l'affichage des rectangles dans OpenGL. Il a été nécessaire d'utiliser un shader spécifique pour le rectangle pour permettre à l'utilisateur d'arrondir les bords des rectangles.

À noter que la méthode d'antialiasing **MSAA** est appliquée à l'intégralité des éléments affichés par OpenGL.

B9H9D a è

a psc

Les méthodes `ToString()` de chaque forme rendent une chaîne de caractère de la forme au format SVG.

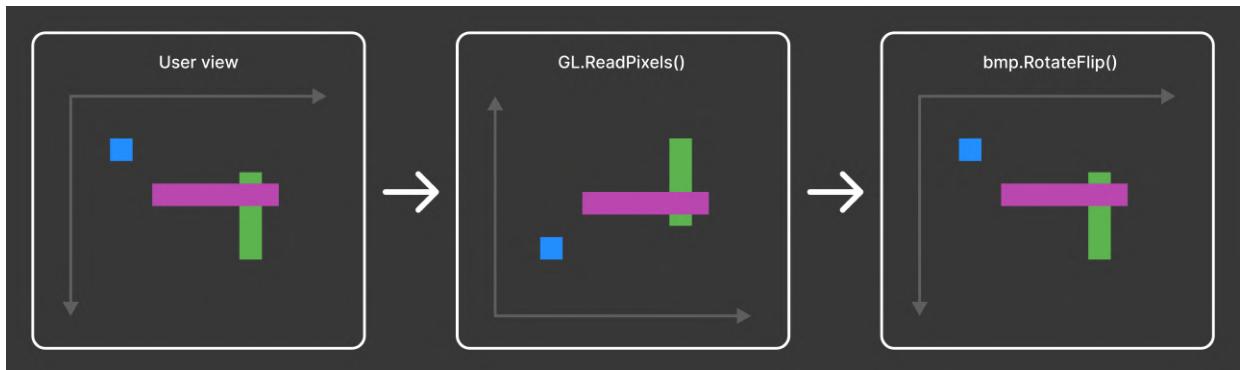
Par exemple, le `ToString()` d'un objet **Circle** retourne

```
<circle cx="20" cy="20" r="5" fill="#FFFFFF"/>.
```

Donc pour exporter au format SVG, il suffit d'appeler la méthode `ToString()` pour chaque élément actuellement à l'écran et de rendre ce texte dans un fichier.

a mj c

La méthode `GL.ReadPixels()` permet de récupérer un pointeur sur un buffer contenant les pixels actuellement rendus par OpenGL. Seulement, OpenGL rend ce pointeur en l'inversant verticalement, il est donc obligatoire d'utiliser `bmp.RotateFlip()` pour l'inverser une seconde fois.



[Fig 45]: Schéma, problème d'export PNG

a i mE

Pour exporter au format MP4 il me suffit de parcourir la timeline et de faire une capture d'écran sur chaque index (en utilisant la même méthode que pour l'export au format PNG).

Ensuite, j'ai utilisé l'objet **VideoWriter** de la librairie [Emgu CV](#) (qui est un portage d'OpenCV pour .NET) qui permet de créer la vidéo.

a c b

Similaire à l'exportation en MP4, j'ai utilisé l'objet **AnimatedGif** de la librairie [AnimatedGif](#) qui permet de créer un GIF en faisant des captures d'écrans en se déplaçant dans la timeline.

a è è b b è

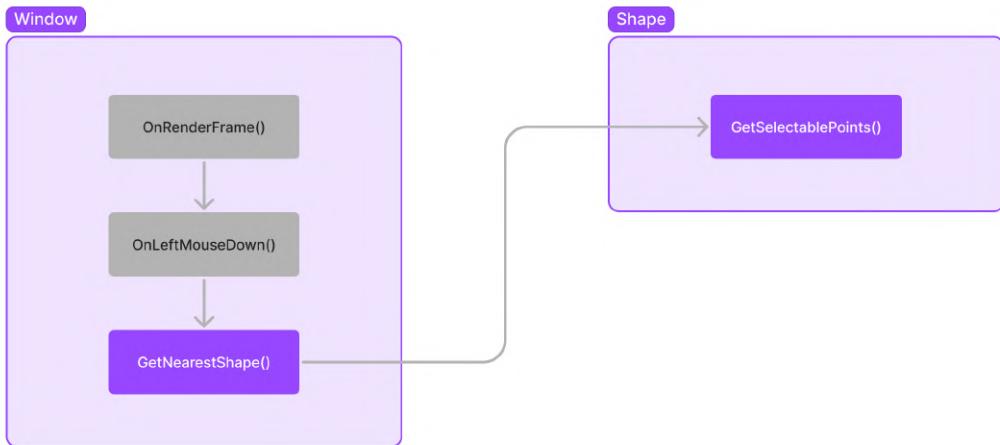
Exporter au format FreeFrame permet de sauvegarde les informations de l'espace de travail en cours, à savoir :

- la timeline ;
- la couleur de fond.

La sauvegarde a été implémentée grâce à de la sérialisation au format JSON.

L'utilisation de [Newtonsoft JSON](#) a été préféré à [System.Text.Json](#) car il est plus complet et permet en autre de sauvegarder les types de mes objets - puisque la liste principale de l'espace de travail contient des formes de type **Shape** (`List<Shape>`) qui est une classe abstraite.

BOBHEE p ffi →



[Fig 46]:Architecture sélection d'une forme

La méthode `GetNearestShape()` permet de récupérer la forme la plus proche d'un point (x, y) donné.

```

# Window.cs
public Shape? GetNearestShape(Vector2i currentLocation)
{
    (Shape? shape, double pythagore) nearest = (null, double.MaxValue);

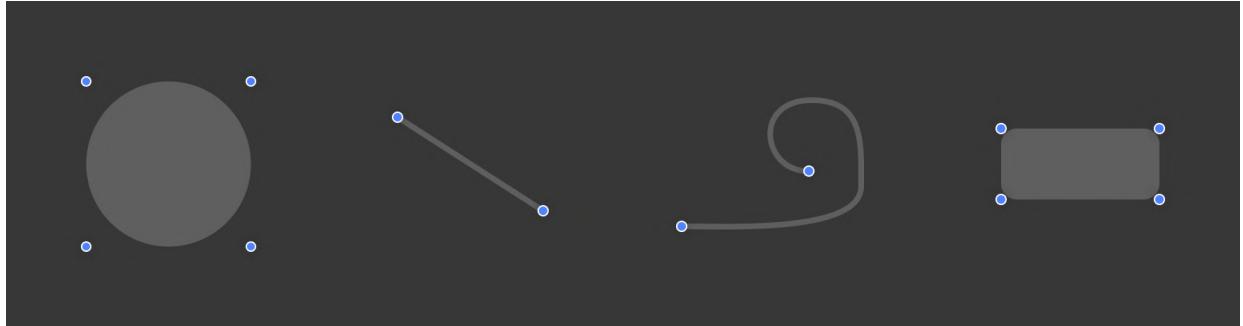
    foreach (Shape shape in Shapes)
    {
        List<Vector2i> points = shape.GetSelectablePoints();

        foreach (Vector2i point in points)
        {
            double pythagore = Math.Sqrt(Math.Pow(point.X - currentLocation.X, 2) + Math.Pow(point.Y - currentLocation.Y, 2));

            if (pythagore < nearest.pythagore) // Get the nearest
                pythagore value
                nearest = (shape, pythagore);
        }
    }
    return nearest.shape;
}
  
```

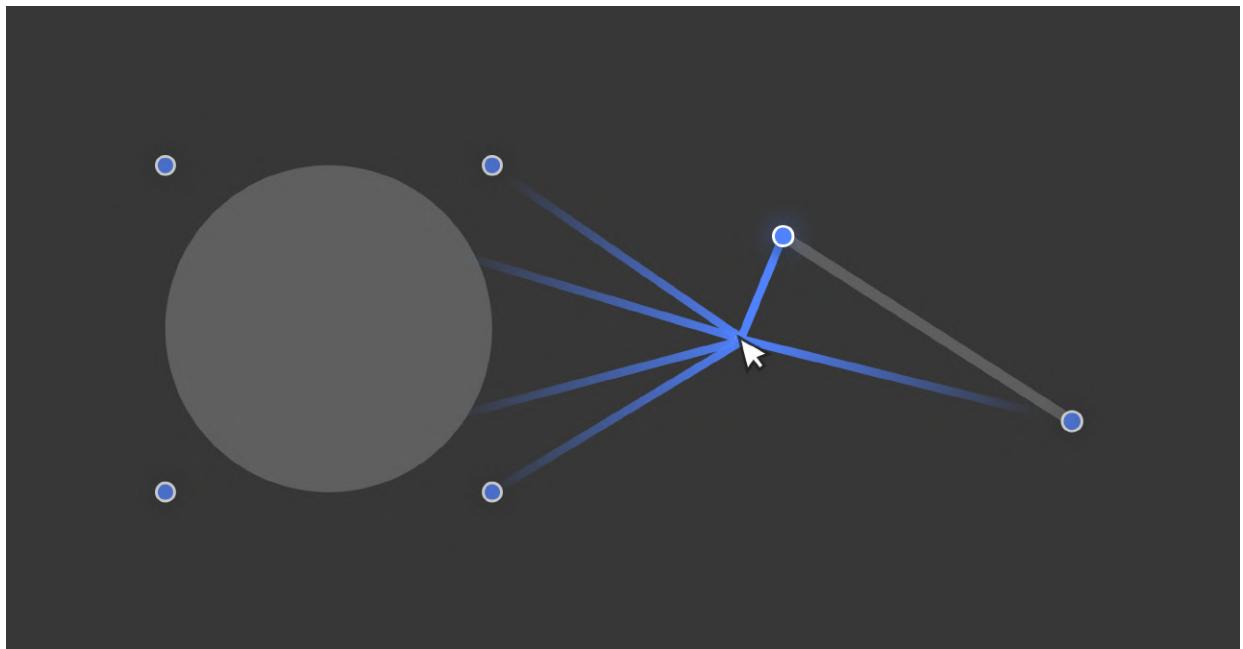
m → ffi

La forme la plus proche est celle qui possède les points de sélection les plus proches de la souris au moment du clic. Les points de sélections sont différents suivant la forme, en voici une liste :



[Fig 47]:Formes et leurs points de sélections

Lors du clic, une droite est tirée entre la souris et chaque point de sélection sur l'espace de travail. La droite la plus courte est donc celle qui va être choisie, pour ensuite sélectionner la forme qui y est associée.

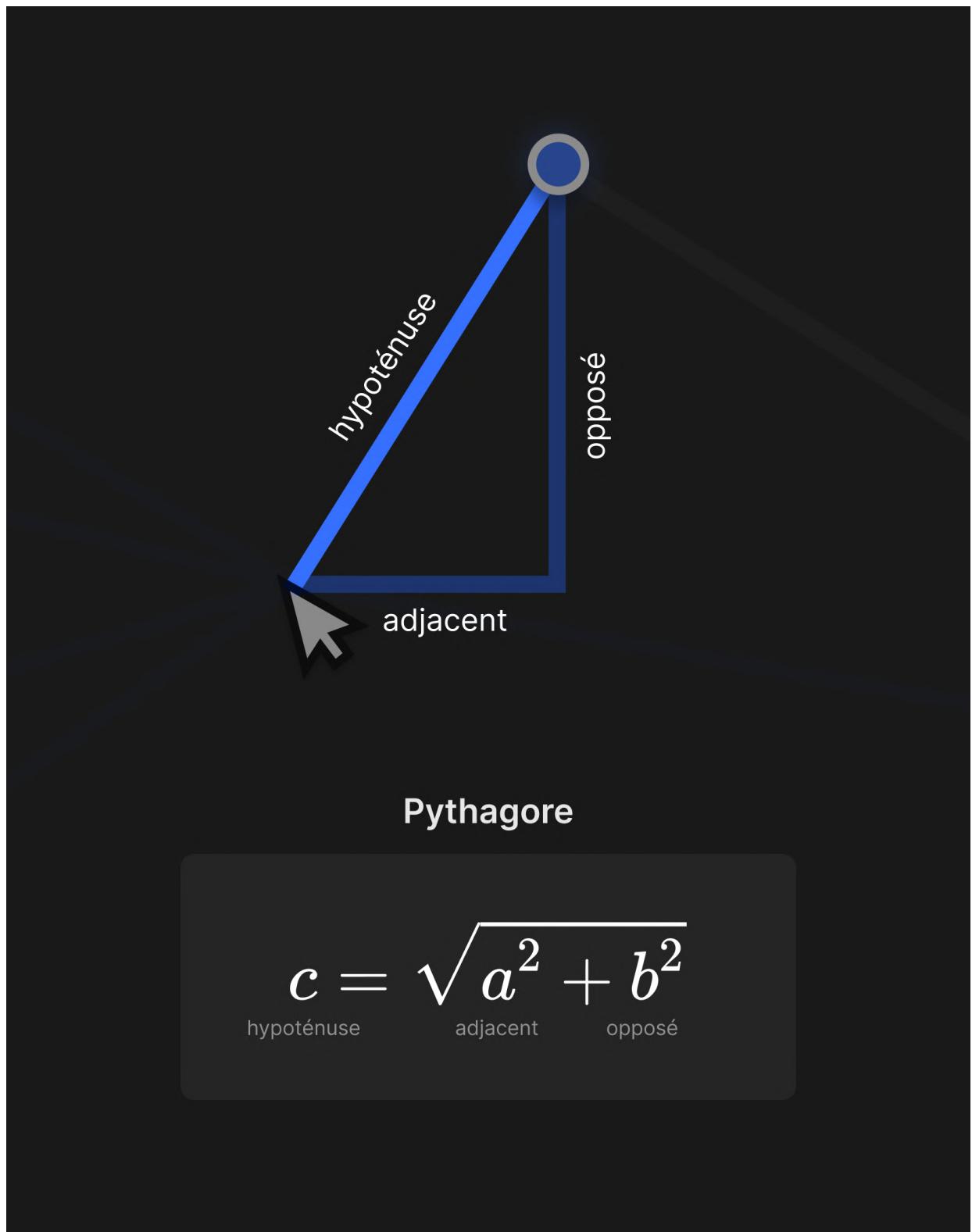


[Fig 48]:Points de sélections choisis lors d'un clic

S

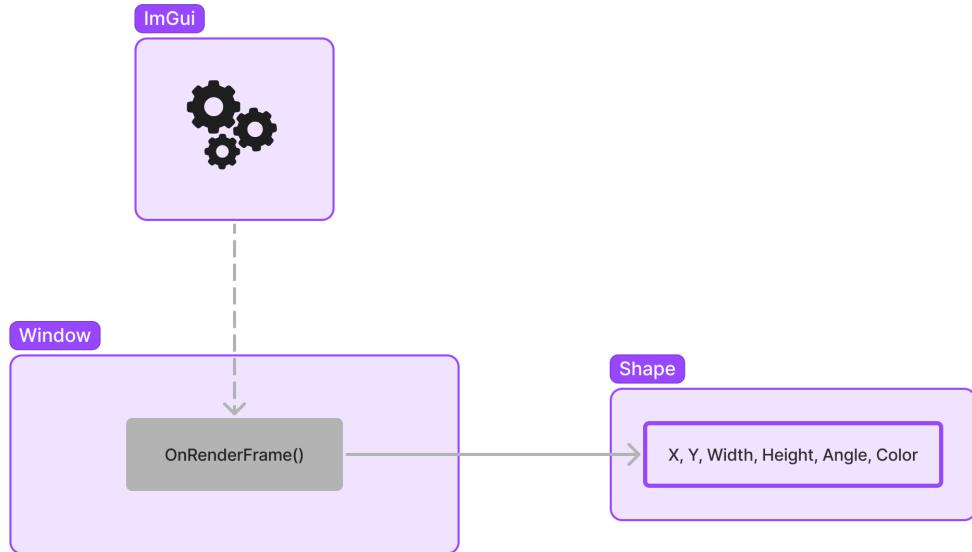
Pythagore a été utilisé pour calculer la droite entre deux points.

Après l'avoir calculé pour chaque point de sélection, il suffit de prendre la valeur la plus petite.



[Fig 49]: Schéma calcul d'hypoténuse

BOBHOFF i → ffè →



[Fig 50]: Architecture modification d'une forme

Pour mettre à jour la forme en interne, lorsque l'utilisateur la modifie, soit via l'interface, le programme vérifie si une de ces valeurs a changé, et si c'est le cas, de mettre à jour la forme sélectionnée avec celles-ci.

```

# Window.cs
if (_selectedShape.X != _ioX ||
    _selectedShape.Y != _ioY ||
    _selectedShape.Width != _ioWidth ||
    _selectedShape.Height != _ioHeight ||
    _selectedShape.Color != new Color4(_ioColor.X,
    _ioColor.Y, _ioColor.Z, _ioColor.W) ||
    _selectedShape.Angle != _ioAngle ||
    _selectedShape.CornerRadius != _ioCornerRadius) // If a
property needs to be updated
{
    _selectedShape.X = _ioX;
    _selectedShape.Y = _ioY;
    _selectedShape.Width = _ioWidth;
    _selectedShape.Height = _ioHeight;
    _selectedShape.Color = new Color4(_ioColor.X,
    _ioColor.Y, _ioColor.Z, _ioColor.W);
}

```

```

    _selectedShape.Angle = _ioAngle;
    _selectedShape.CornerRadius = _ioCornerRadius;

    // Update the shape in the timeline
    _timeline.UpdateShapeInTimeline(_selectedShape);

    _selectedShape.ImplementObject();
    _selector.Select(_selectedShape);
}

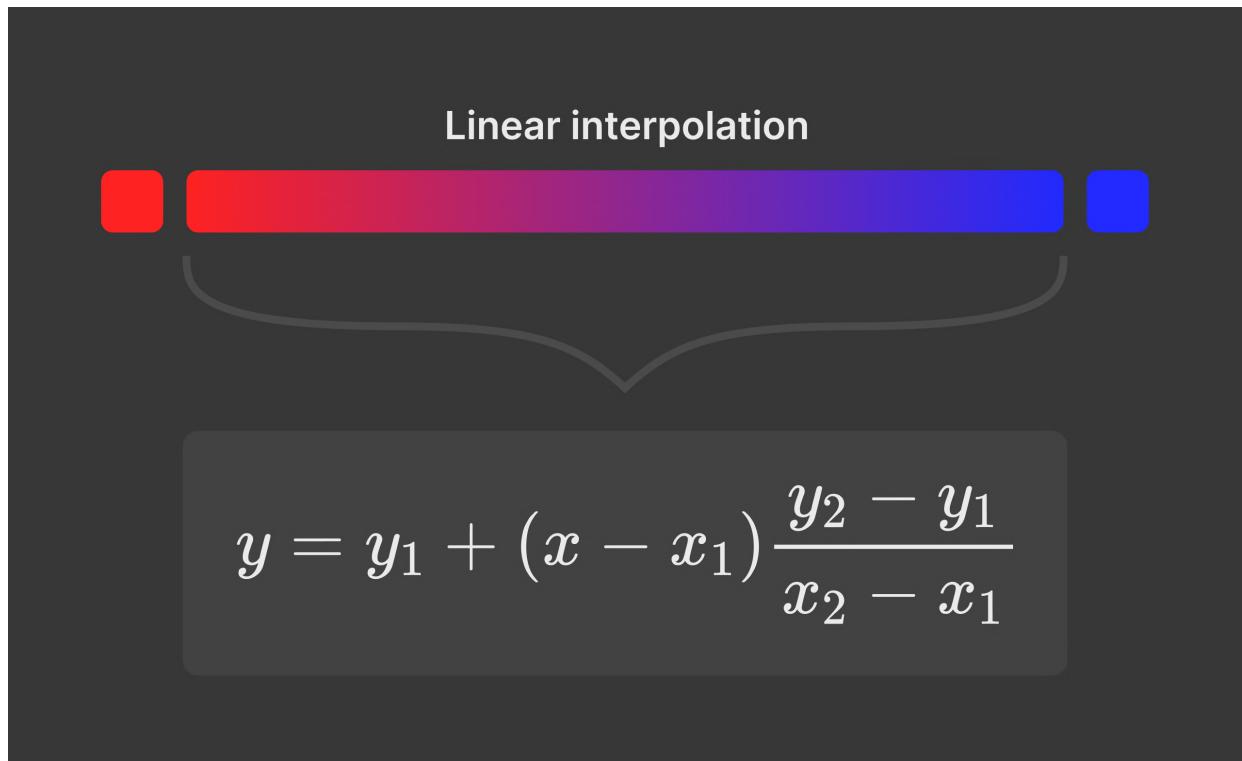
```

Une alternative qui rendrait le code plus court aurait été d'instancier une forme et de les comparer. Cependant, aucune forme ne possède un constructeur prenant les sept paramètres testés ce qui m'aurait obligé à créer une forme générique ou à rendre la classe **Shape** non abstraite.

B9H9Gq

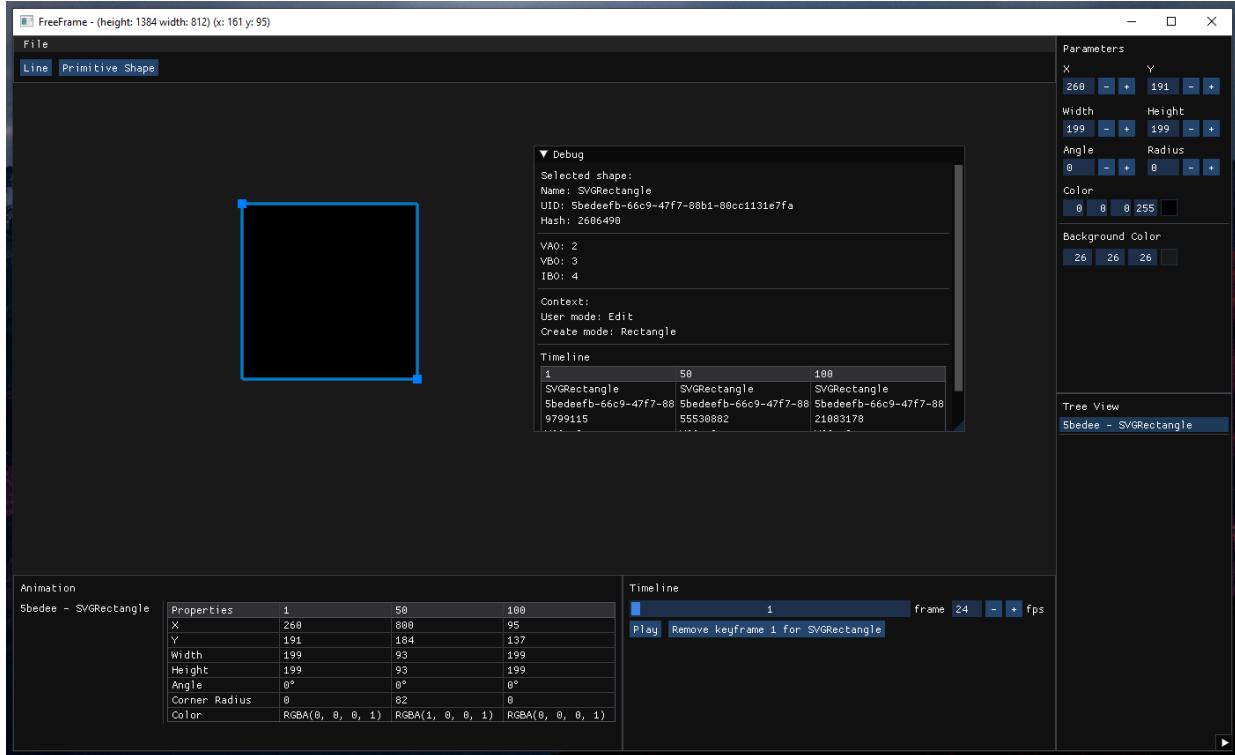
Si l'utilisateur change la couleur d'une forme (ici en rouge) pour la sauvegarde dans la timeline. Ensuite qu'il modifie la couleur de cette forme à un autre moment (ici en bleu). L'utilisateur aura sauvégardé l'état de sa forme à deux moments donnés. Il aura alors créé deux **keyframes**.

Dans ce cas, l'utilisateur aura uniquement créé deux keyframes, donc lorsque la timeline se trouve entre les deux keyframes, de l'[interpolation linéaire](#) doit être calculé pour chaque propriété de la forme.



[Fig 51]:Schéma interpolation linéaire

B9H9He èffi è



[Fig 52]:Interface de l'application

ImGui ne possède pas d'outil pour des explorateurs de fichiers et les plugins d'ImGui rajoutant cette fonctionnalité sont uniquement disponibles en C++. En cherchant, j'ai trouvé un [GitHub Gist](#) qui implémente un outil d'explorateur de fichier pour ImGui.NET. Je l'ai donc utilisé et modifié pour l'adapter à mes besoins.

e c 9 aq

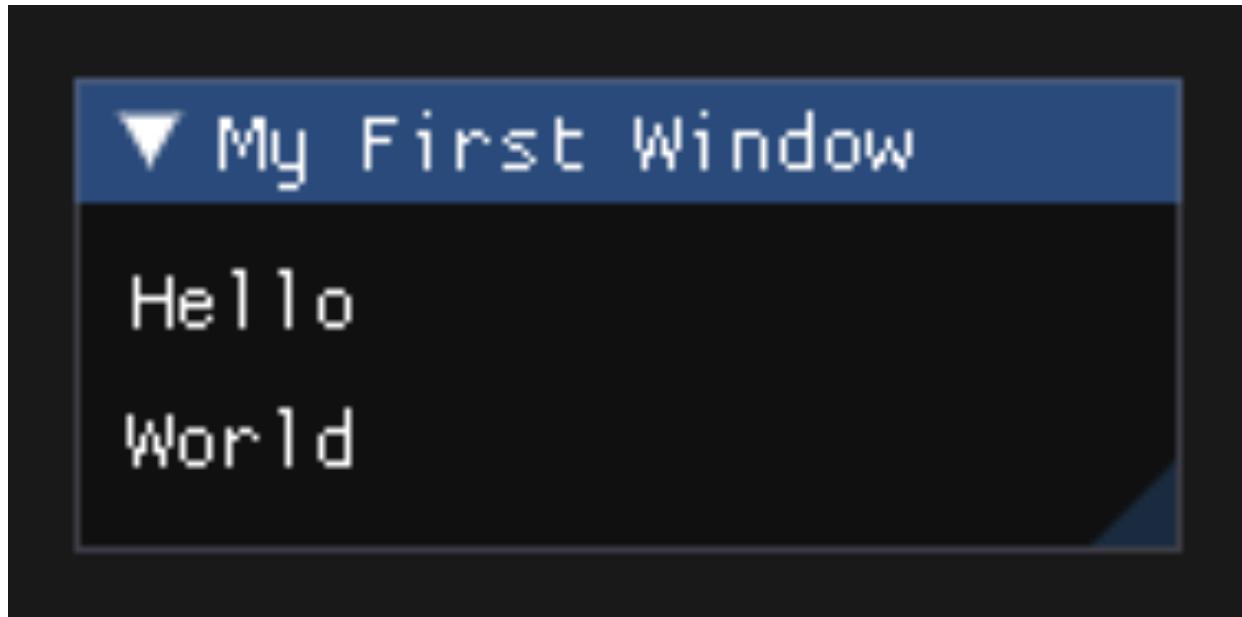
L'implémentation de ImGui est structurée ainsi :

```
ImGui.Begin("My First Window");

ImGui.Text("Hello");
ImGui.Spacing();
ImGui.Text("World");

ImGui.End();
```

Pour créer une nouvelle fenêtre avec du texte, il suffit de créer un élément `Text` et de l'entourer par les éléments `Begin` et `End`.



[Fig 53]:Hello world dans ImGui

B9BI q

Plusieurs stratégies ont été mises en place pour vérifier que chaque nouvelle fonctionnalité rajoutée ne détériore pas les précédentes.

Premièrement, des tests unitaires préventifs ont été mis en place qui permet aux mainteniciens de tester les différents systèmes d'importation.

Également, l'outil de debug expliqué dans sa partie qui lui est dédié permet de résoudre les différents problèmes de gestion des identifiants d'OpenGL.

Le reste des fonctionnalités peut être facilement testé manuellement, car il n'y a pas de restrictions de droit d'accès, chaque fonctionnalité est accessible en quelques clics.

V è

Propriété	Valeur
Nom	Création d'une ligne
Cas	L'utilisateur peut utiliser l'outil <i>Line</i> en cliquant sur <i>Line</i> pour créer une nouvelle ligne sur l'espace de travail
Résultat attendu	Une nouvelle ligne apparaît sur l'écran avec la taille définie précédemment pas la souris
Propriété	Valeur
Nom	Création d'un rond
Cas	L'utilisateur peut utiliser l'outil <i>Circle</i> en cliquant sur <i>Primitive Shape/Circle</i> pour créer un nouveau rond sur l'espace de travail
Résultat attendu	Un nouveau rond apparaît sur l'écran avec la taille définie précédemment pas la souris
Propriété	Valeur
Nom	Création d'un rectangle
Cas	L'utilisateur peut utiliser l'outil <i>Rectangle</i> en cliquant sur <i>Primitive Shape/Rectangle</i> pour créer un nouveau rectangle sur l'espace de travail
Résultat attendu	Un nouveau rectangle apparaît sur l'écran avec la taille définie précédemment pas la souris
Propriété	Valeur
Nom	Création d'un triangle
Cas	L'utilisateur peut utiliser l'outil <i>Triangle</i> en cliquant sur <i>Primitive Shape/Triangle</i> pour créer un nouveau triangle sur l'espace de travail
Résultat attendu	Un nouveau triangle apparaît sur l'écran avec la taille définie précédemment pas la souris

i → ffè

Propriété	Valeur
Nom	Sélection d'une forme
Cas	L'utilisateur peut cliquer sur l'espace de travail pour sélectionner une forme
Résultat attendu	La forme la plus proche du clic est sélectionnée et possède un sélecteur bleu autour d'elle

Propriété	Valeur
Nom	Changement de position via interface
Cas	L'utilisateur peut déplacer une forme en utilisant les champs de l'interface X et Y dans <i>Parameters</i>
Résultat attendu	La forme visible est déplacée à l'endroit choisi

Propriété	Valeur
Nom	Changement de taille via interface
Cas	L'utilisateur peut redimensionner une forme en utilisant les champs de l'interface <i>Width</i> et <i>Height</i> dans <i>Parameters</i>
Résultat attendu	La forme visible possède la taille choisie
Condition	Cette option est désactivée pour la forme <i>SVGPath</i>

Propriété	Valeur
Nom	Changement de position via l'espace de travail
Cas 1	L'utilisateur peut déplacer une forme en utilisant le rectangle en haut à gauche du sélecteur
Résultat attendu	La forme visible est déplacée à l'endroit choisi
Cas 2	L'utilisateur peut effectuer le Cas 1 en ajoutant une contrainte de déplacement en appuyant sur la touche Shift
Résultat attendu	La forme se déplace uniquement horizontalement ou verticalement sur l'axe de départ
Propriété	Valeur
Nom	Changement de taille via l'espace de travail
Cas 1	L'utilisateur peut déplacer une forme en utilisant le rectangle en bas à droite du sélecteur
Résultat attendu	La forme visible possède la taille choisie
Condition	Cette option est désactivée pour la forme <i>SVGPath</i>
Cas 2	L'utilisateur peut effectuer le Cas 1 en ajoutant une contrainte de redimension en appuyant sur la touche Shift
Résultat attendu	La forme se redimensionne en gardant le ratio 1:1
Condition	Cette option est désactivée pour la forme <i>SVGPath</i>
Propriété	Valeur
Nom	Rotation
Cas	L'utilisateur peut changer l'angle d'affichage une forme en utilisant le champ de l'interface <i>Angle</i> dans <i>Parameters</i>
Résultat attendu	La forme visible est affichée avec l'angle demandé
Condition	Cette option est désactivée pour la forme <i>SVGPath</i>

Propriété	Valeur
Nom	Arrondis des bords
Cas	L'utilisateur peut changer l'arrondissement des bords une forme en utilisant le champ de l'interface <i>Radius</i> dans <i>Parameters</i>
Résultat attendu	La forme visible possède des bords arrondis avec l'intensité spécifiée
Condition	Cette option est uniquement active pour la forme <i>Rectangle</i>
Propriété	Valeur
Nom	Changement couleur de forme
Cas	L'utilisateur peut utiliser l'outil de sélection de couleur en cliquant sur le carré de couleur en bas de <i>Color</i> dans la section <i>Parameters</i> pour changer la couleur de la forme sélectionnée
Résultat attendu	La nouvelle couleur de la forme correspond à celle choisie
Propriété	Valeur
Nom	Changement couleur de fond
Cas	L'utilisateur peut utiliser l'outil de sélection de couleur en cliquant sur le carré de couleur en bas de <i>Background Color</i> dans la section <i>Parameters</i> pour changer la couleur de fond de l'espace de travail
Résultat attendu	La nouvelle couleur de fond correspond à celle choisie par l'utilisateur
Propriété	Valeur
Nom	Changement ordre de disposition
Cas	L'utilisateur peut changer l'ordre de disposition de plusieurs formes dans la section <i>Tree View</i> de l'interface
Résultat attendu	La nouvelle disposition correspond à celle indiquée dans la <i>Tree View</i>

q

Propriété	Valeur
Nom	Ajout d'une keyframe
Cas	L'utilisateur peut ajouter une keyframe de la forme sélectionnée en cliquant sur le bouton <i>Create keyframe for X</i> dans la section <i>Timeline</i>
Résultat attendu	La keyframe ajoutée s'affiche dans la section <i>Animation</i>

Propriété	Valeur
Nom	Avancement de la timeline
Cas 1	L'utilisateur peut démarrer l'avancement dans la timeline en appuyant sur le bouton <i>Play</i> de la section <i>Timeline</i>
Résultat attendu	La timeline avance et les formes sur l'espace de travail sont interpolées en fonction des keyframes présentes
Cas 2	L'utilisateur peut effectuer le Cas 1 en changeant le nombre d'images affiché par seconde grâce au champ <i>fps</i> de la section <i>Timeline</i>
Résultat attendu	La timeline avance avec la cadence spécifiée

a è

Propriété	Valeur
Nom	Exportation de FreeFrame
Cas	L'utilisateur peut utiliser l'outil Save en cliquant sur <i>File/Save/Save</i> pour créer un nouveau fichier FreeFrame
Résultat attendu	Un fichier avec l'extension <code>.freeframe</code> est créé à l'endroit demandé
Propriété	Valeur
Nom	Exportation de SVG
Cas	L'utilisateur peut utiliser l'outil <i>Save as SVG</i> en cliquant sur <i>File/Save/Save as SVG</i> pour créer un nouveau fichier SVG
Résultat attendu	Un fichier avec l'extension <code>.svg</code> est créé à l'endroit demandé et il contient toutes les informations de l'état de l'espace de travail visible
Propriété	Valeur
Nom	Exportation de PNG
Cas	L'utilisateur peut utiliser l'outil <i>Save as PNG</i> en cliquant sur <i>File/Save/Save as PNG</i> pour créer un nouveau fichier PNG
Résultat attendu	Un fichier avec l'extension <code>.png</code> est créé à l'endroit demandé et il est la représentation graphique de l'état de l'espace de travail visible
Propriété	Valeur
Nom	Exportation de MP4
Cas	L'utilisateur peut utiliser l'outil <i>Save as MP4</i> en cliquant sur <i>File/Save/Save as MP4</i> pour créer un nouveau fichier MP4
Résultat attendu	Un fichier vidéo avec l'extension <code>.mp4</code> est créé à l'endroit demandé et il affiche dynamiquement chaque état de l'espace de travail visible dans la timeline

Propriété	Valeur
Nom	Exportation de GIF
Cas	L'utilisateur peut utiliser l'outil <i>Save as GIF</i> en cliquant sur <i>File/Save/Save as gif</i> pour créer un nouveau fichier GIF
Résultat attendu	Un fichier vidéo avec l'extension <code>.gif</code> est créé à l'endroit demandé et il affiche dynamiquement chaque état de l'espace de travail visible dans la timeline

e è

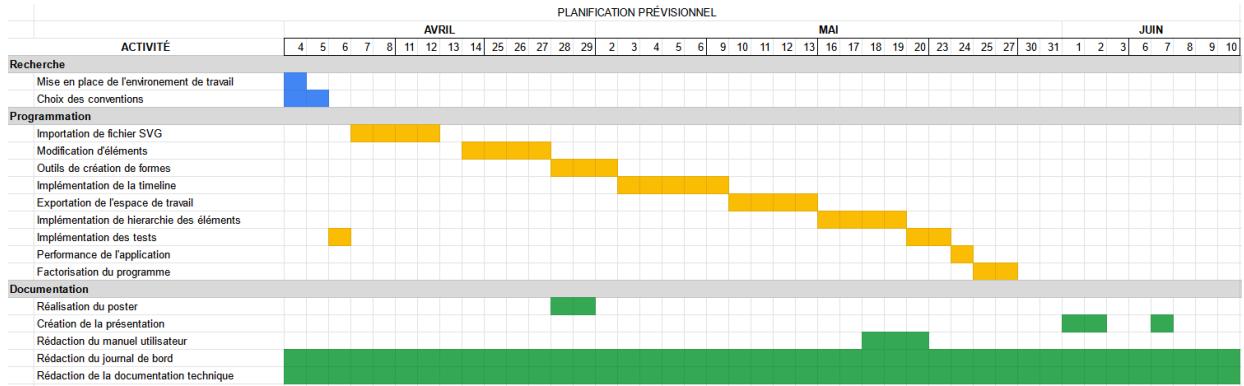
Propriété	Valeur
Nom	Importation de FreeFrame
Cas	L'utilisateur peut utiliser l'outil <i>Open a new project</i> en cliquant sur <i>File/Open/Open a new project</i> pour importer un fichier FreeFrame
Résultat attendu	L'espace de travail est remplacé par les informations (timeline, formes, fond de l'espace de travail) contenues dans le fichier

Propriété	Valeur
Nom	Ajout de SVG
Cas	L'utilisateur peut utiliser l'outil <i>Import a SVG</i> en cliquant sur <i>File/Open/Import a SVG</i> pour importer un fichier SVG
Résultat attendu	L'espace de travail est additionné avec les informations de formes contenues dans le fichier

Propriété	Valeur
Nom	Importation de SVG
Cas	L'utilisateur peut utiliser l'outil <i>Open a new project from a SVG</i> en cliquant sur <i>File/Open/Open a new project from a SVG</i> pour importer un fichier SVG
Résultat attendu	L'espace de travail est remplacé par les informations de formes contenues dans le fichier

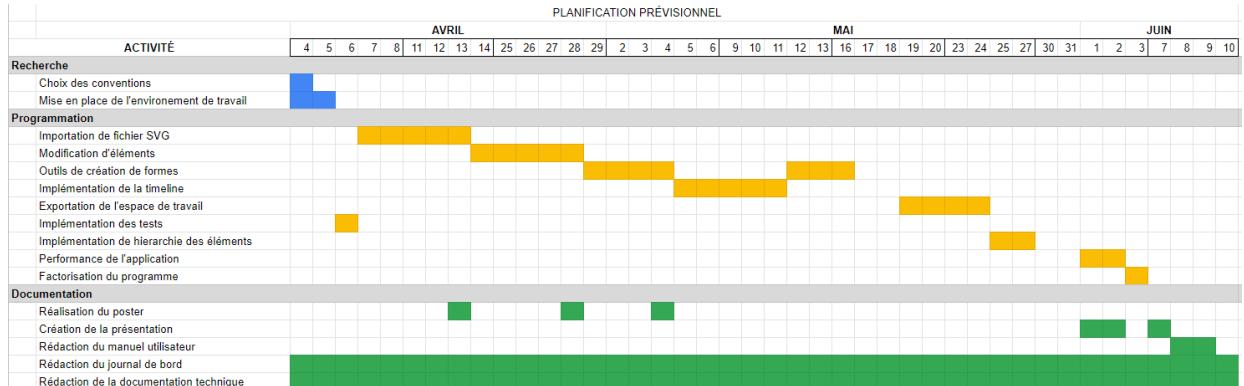
B9J mè ffè

B9J9B mè ffè



[Fig 54]:Planification prévisionnelle

B9J9C mè ffi



[Fig 55]:Planification effective

B9CA V

B9CA9B V

<type>[component][!]: <description>

1. `fix`: pour résoudre un bug
2. `feat`: pour ajouter une nouvelle fonctionnalité
3. `style`: mise à jour lié au style
4. `refactor`: factorisé du code
5. `test`: lié aux tests
6. `docs`: lié à la documentation
7. `chore`: maintenance du code régulière

B9CB h è

La première chose que l'on peut remarquer est que la liste d'éléments SVG compatibles est courte. Implémenter l'intégralité des éléments compatibles n'été pas le but du projet, de plus que ç'aurait été très répétitif et long et finalement n'ajoute pas de technicité au projet.

Actuellement l'outil d'exportation de GIF ne permet pas de faire des animations de plus de 50 fps, lié à un [bug interne](#) de la librairie que j'utilise.

B9CC S è ff

Malgré les différentes recherches faites en amont du projet, lorsque je l'ai commencé, beaucoup d'aspect du projet m'était encore très obscure. Au fur et à mesure que je me plongeais dans l'apprentissage, je découvrais les limitations d'OpenGL et j'apprenais peu à peu à le maîtriser. Une des choses que je ferais différents serait d'au lieu d'implémenter chaque forme indépendamment, de gérer chaque élément importé comme un élément possédant que des lignes ou des courbes de Béziers. De ce fait, il serait possible de déformer n'importe quel élément. Pour que ça fonctionne, il faudrait également implémenter un système de triangulation.

Mon application est compatible sur Windows et Linux, car elle a été fondée sur OpenGL et malheureusement, Apple ne maintient plus OpenGL sur leurs plateformes. Il aurait été intéressant de dissocier OpenGL du projet et de ce fait pouvoir utiliser plusieurs autres API graphiques en fonction de la plateforme de l'utilisateur (Direct3D et Metal par exemple).

B9CD m ff

Je pense que le problème principal a été d'apprendre OpenGL et les notions de computer graphics. J'avais tout d'abord suivi les tutoriels de la chaîne YouTube [The Cherno](#) (qui est un ancien employé d'Ubisoft). Comme on peut le voir dans le journal de bord, j'ai dû faire beaucoup de schémas pour être sûr que j'avais bien compris les concepts.

Après la modification d'un shader je dois le déplacer dans le répertoire de lancement de Visual Studio. Au départ je ne le savais pas, mais j'ai ensuite compris et j'ai commencé à les déplacer à la main. J'ai rapidement remarqué que c'était une perte de temps, j'ai cherché s'il existait un moyen de dire à Visual Studio de déplacer des fichiers à l'exécution du programme. C'est alors que j'ai découvert les commandes *post-build* et décidé d'implémenter la mienne qui puisse déplacer automatiquement les shaders dans le répertoire de lancement.

B9CE V ff

Pour la réalisation de ce document, j'ai préféré d'expliquer le plus possible par des schémas, car je trouve beaucoup plus simple à comprendre que de lire du texte.

Je suis très content d'avoir réussi le défi que j'avais décidé de relever qui était d'apprendre le domaine de la computer graphics sans connaître le moindre concept. Ça m'a permis de comprendre qu'en information, il était possible d'apprendre n'importe quel domaine, simplement grâce à une connexion internet.

B9CF o ff

Je tiens premièrement à remercier M. Schmid d'avoir pris le temps de m'avoir aidé pour la direction de ce projet et pour les différents retours sur la réalisation de ce document. Je tiens également à remercier M. Bonvin pour le partage de sa motivation. Je remercie M. Zanardi pour sa bonne humeur et sa bienveillance. Merci à mes collègues M. Rhomer, M. Tayan et M. Upchurch d'avoir donné des avis constructifs qui m'ont beaucoup aidé tout au long du projet.

B9CG Z è

- OpenTK 4.7.1
- ImGui.NET 1.87.3
- C# 10.0
- .NET 6
- Visual Studio 2022 17.1.5

- Zotero 6.0.4

BCH T ff è

Gregory, Jason. *Game Engine Architecture*. Third edition. Boca Raton: Taylor and Francis, CRC Press, 2018.

Marschner, Steve, et Peter Shirley. *Fundamentals of computer graphics*. Fifth edition. Boca Raton: CRC Press/Taylor & Francis Group, 2022.

Sarrafzadeh, M. « Basic Object-Oriented Programming and State Machines ». *ACM SIGDA Newsletter* 20, no 1 (juin 1990): 91. <https://doi.org/10.1145/378886.380416>.

Talks at Google. *A Philosophy of Software Design | John Ousterhout | Talks at Google*, 2018. <https://www.youtube.com/watch?v=bmSAYlu0NcY>.

The Book of Shaders. « The Book of Shaders ». Consulté le 12 avril 2022. <https://thebookofshaders.com/>.

« Triangle Edge Rendering – Sebastian Weiss ». Consulté le 27 avril 2022. <https://www.sebastian-weiss.eu/blog-triangle-edge-rendering/>.

Wright, Richard S., Graham Sellers, et Nicholas Haemel. *OpenGL Superbible: Comprehensive Tutorial and Reference*. Seventh edition. New York: Addison-Wesley, 2015.

ne

~~OB h~~ → E è

J'ai commencé la semaine par mettre en place le planning entier du diplôme. J'avais déjà listé les tâches nécessaires pour faire le projet sur GitHub, j'ai donc simplement repris cette liste.

Pour la tâche que je vais commencer demain, je vais avoir besoin d'importer un fichier via une fenêtre de dialogue. Plusieurs options s'offrent à moi :

- OpenFileDialog
 - Ce choix n'est pas crossplatform, je devrais refaire la même chose pour chaque plateforme si je souhaite un jour rendre le projet crossplatform
- Binder [nativefiledialog](#)
 - Outil complet, mais le binder en C# prendrait du temps
- [FilePicker.cs](#)
 - C'est un fichier que j'ai trouvé qui recréer une fenêtre de dialog en utilisant DearImGui

Sachant que c'est un problème secondaire à mon projet je vais choisir file picker car...

Je décide de faire le système de gestion de l'importation et de voir ce problème une fois cette tâche terminée.

Les [fichiers SVG](#) étant basé sur du [XML](#), je vais utiliser l'objet natif `XmlReader` pour la lecture.

Dans un fichier SVG, chaque élément (ex : `<circle cx="50" cy="50" r="50"/>`) possède des propriétés géométriques spécifiques (ex : `cx`, `cy`, `r`). Je pensais au départ faire un objet `Shape` qui contiendrait la liste des propriétés géométriques `List<Int>`. Cependant, il y a des propriétés géométriques qui ne sont pas de type `Int`, je dois donc manuellement décrire les propriétés de chaque élément, comme suit :

```
class SVGCircle : Shape
{
    private float _cx;
    private float _cy;
    private float _r;
```

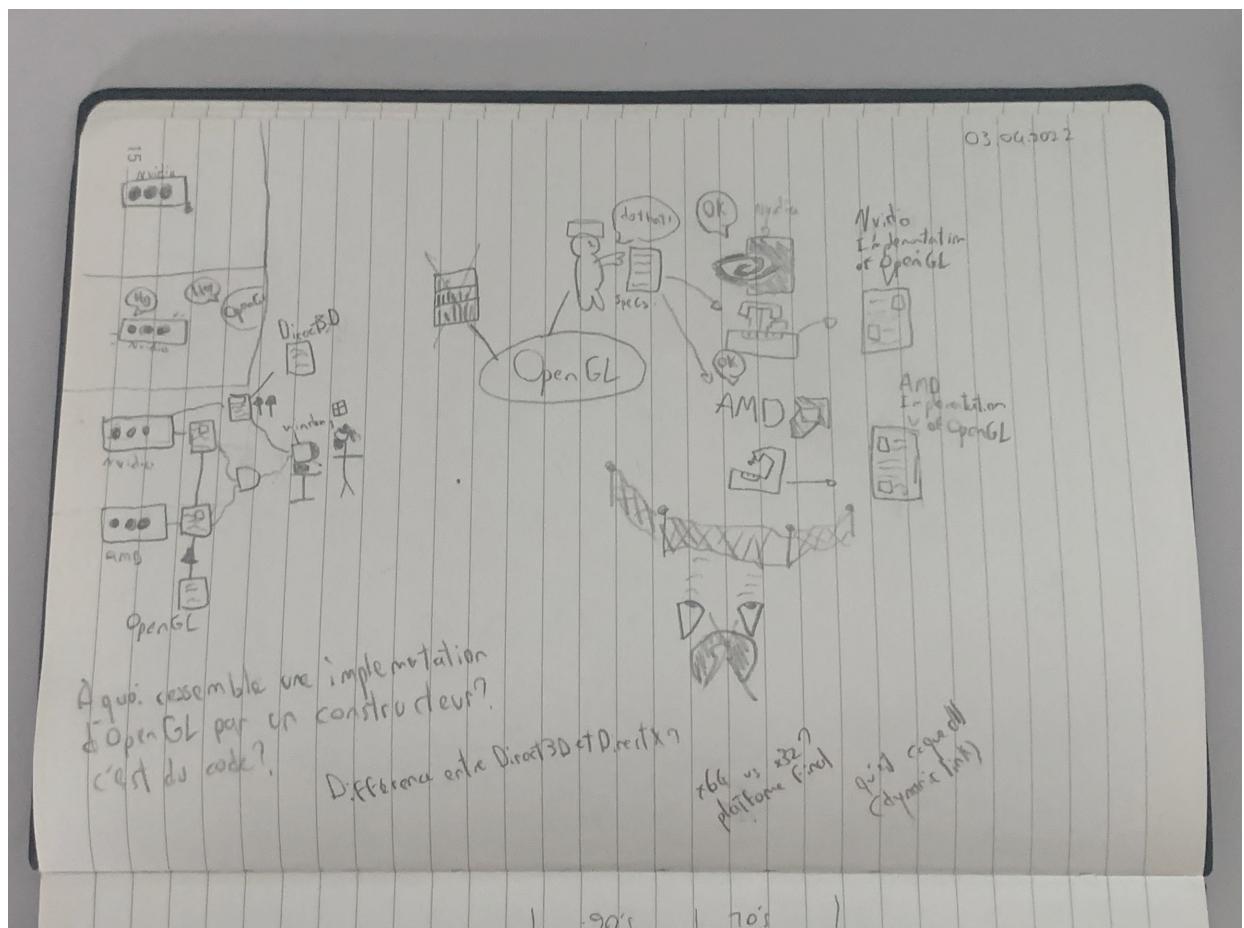
}

Le titre de mes tâches était basé sur la méthode agile (ex :

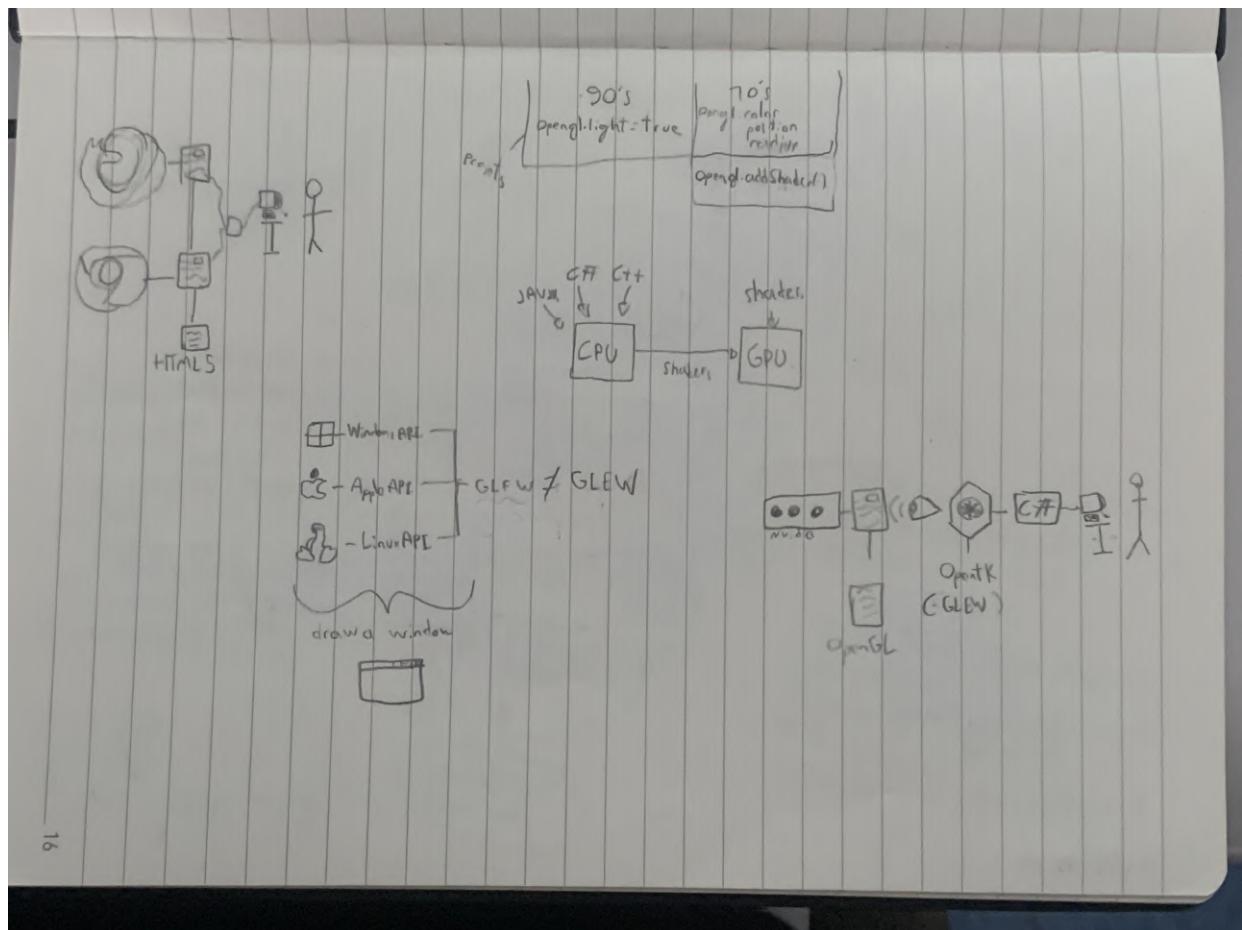
En tant qu'utilisateur je veux importer un fichier SVG). J'ai décidé de les changer en titre simple (ex : **Import de fichiers SVG**) pour que ce soit plus agréable à lire.

Idées de la journée :

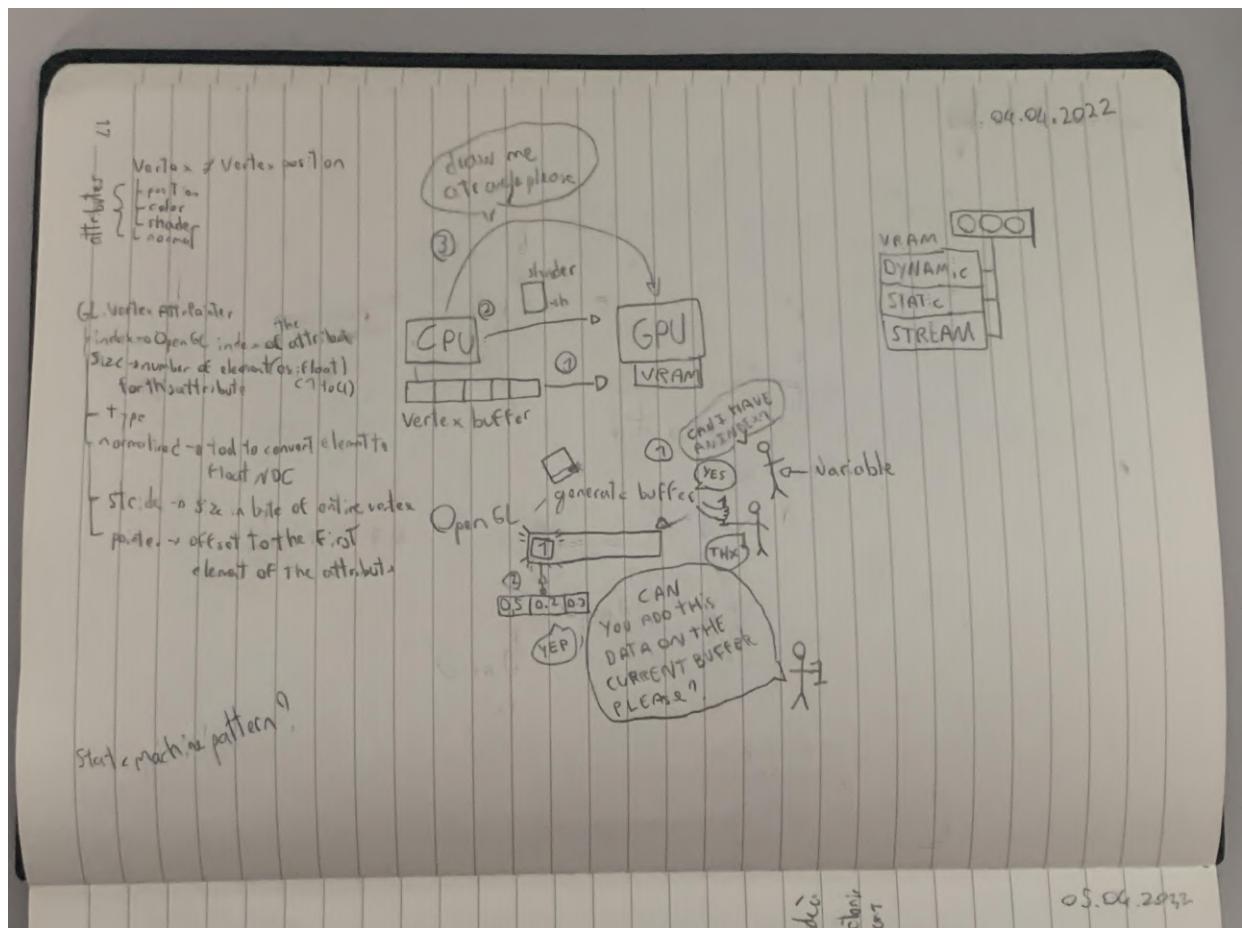
- À la fin de chaque semaine, faire une review de la semaine passée (voir ce qui s'est bien passé, mal passé). Et prévoir un planning pour la semaine suivante
- Écrire quelques objectifs SMART pour la semaine suivante
- Remplir une évaluation intermédiaire à la fin de chaque semaine



[Fig 56]:Journal papier 4 avril



[Fig 57]:Journal papier 4 avril 2



[Fig 58]:Journal papier 4 avril 3

OCCI è → F è

Aujourd'hui je dois terminer la liste exhaustive des éléments SVG compatibles. Il faudra que je déclenche une alerte si l'utilisateur envoie un fichier avec des balises pas compatibles.

Cette partie ne touche pas à de l'affichage, je vais en profiter pour écrire des tests unitaires. Je vais tester avec des fichiers :

- SVG corrompus ;
- SVG avec balises non compatibles ;
- SVG valide.

Le reste de ma classe d'importation `Importer` n'étant pas fini, les tests vont me permettre de savoir si je l'aurai bien implémenté plus tard.

J'ai vite enlevé `DataRow` de mes idées, car les lignes deviendraient longues et illisibles. J'ai préféré faire plusieurs méthodes de tests à la suite.

J'ai demandé l'avis de M. Bonvin qui m'a dit que c'est un bon choix. Sur le moment, il m'a donné deux idées :

- faire des tests de régression journaliers sur la vitesse d'exécution ;
- au lieu de reporter les tests de façon journalière utiliser des numéros de version ou des milestones sur GitHub.

Pour apprendre OpenGL, je regarde [la série de vidéos créée par la chaîne YouTube "The Cherno"](#) à ce sujet.

C90B m ffi → è

J'ai passé des semaines précédant mon diplôme pour trouver la meilleure méthode d'apprentissage.

Donc après avoir regardé la vidéo sur les [vertex buffer](#) et réaliser un mindmap, je commence à implémenter.

```
GL.BufferData(BufferTarget target, int size, IntPtr data, BufferUsageHint usage);
```

Cette fonction permet de définir la taille qu'il faut allouer pour le buffer (un buffer étant un endroit où l'on peut stocker des données, les sommets des objets (**en computer graphics, les triangles sont les formes préconisées. Gregory, Jason. Game Engine Architecture. Third edition. (Boca Raton: Taylor and Francis, CRC Press, 2018), 625.**) que je vais dessiner vont être stockés dans un `vertex buffer`).

Cependant, pour la taille (`size`) j'aimerais automatiser en regardant le nombre d'éléments dans mon tableau de sommet (qui forme un triangle).

```
readonly float[] _vertices =
{
    0.0f, -0.5f, 0.0f, // Top
    -0.5f, 0.5f, 0.0f, // Bottom-Left
    0.5f, 0.5f, 0.0f // Bottom-Right
};
```

Mon calcul serait quelque chose comme suit :

```
int size = _vertices.Length * sizeof(_vertices.GetType().GetElementType())
```

Mais ça ne fonctionne pas. Après plusieurs tentatives, je décide de poster [ma question sur StackOverflow](#).

La réponse est qu'en C# `sizeof()` est uniquement autorisé pour des types non gérés. Après avoir trouvé cette réponse, je conclus que je suis resté bloqué sur ce problème longtemps alors qu'il n'est pas si important. Je change de sujet et j'avance aussitôt.

Choses à faire :

- Tableau des tests exécutés.

Idées de la journée :

- Réaliser des tests de régression pour la vitesse d'exécution du programme OU allouer un jour pour améliorer les performances
- En plus d'écrire un peu de la documentation chaque jour, que je place des jours uniquement fait pour documenter
- Utiliser des `struct` pour la liste des *vertices* au lieu d'un tableau de `float`
- Mettre en place une pipeline GitHub CI pour mes tests unitaires
- Traduire mes tâches en anglais et passer le projet en public

OBDI ffi → Gè

Aujourd'hui je ne m'attends pas à écrire beaucoup de code, je vais principalement me documenter sur OpenGL.

Comme hier, je continue la série de vidéos sur OpenGL.

Je viens de penser, l'outil d'exportation Markdown vers PDF d'Obsidian ne fait pas de documents très beaux. Il faudra qu'à la fin de mon diplôme, que j'utilise Pandoc pour me permettre d'exporter mon fichier Markdown correctement.

Je suis tombé sur [docs.gl](#) qui est une excellente documentation pour OpenGL/GLSL.

Les shaders sont sous formes de fichier et pour tester le programme lorsque je modifie un shader je dois à chaque fois les dans le dossier `bin/Debug/net6.0`.

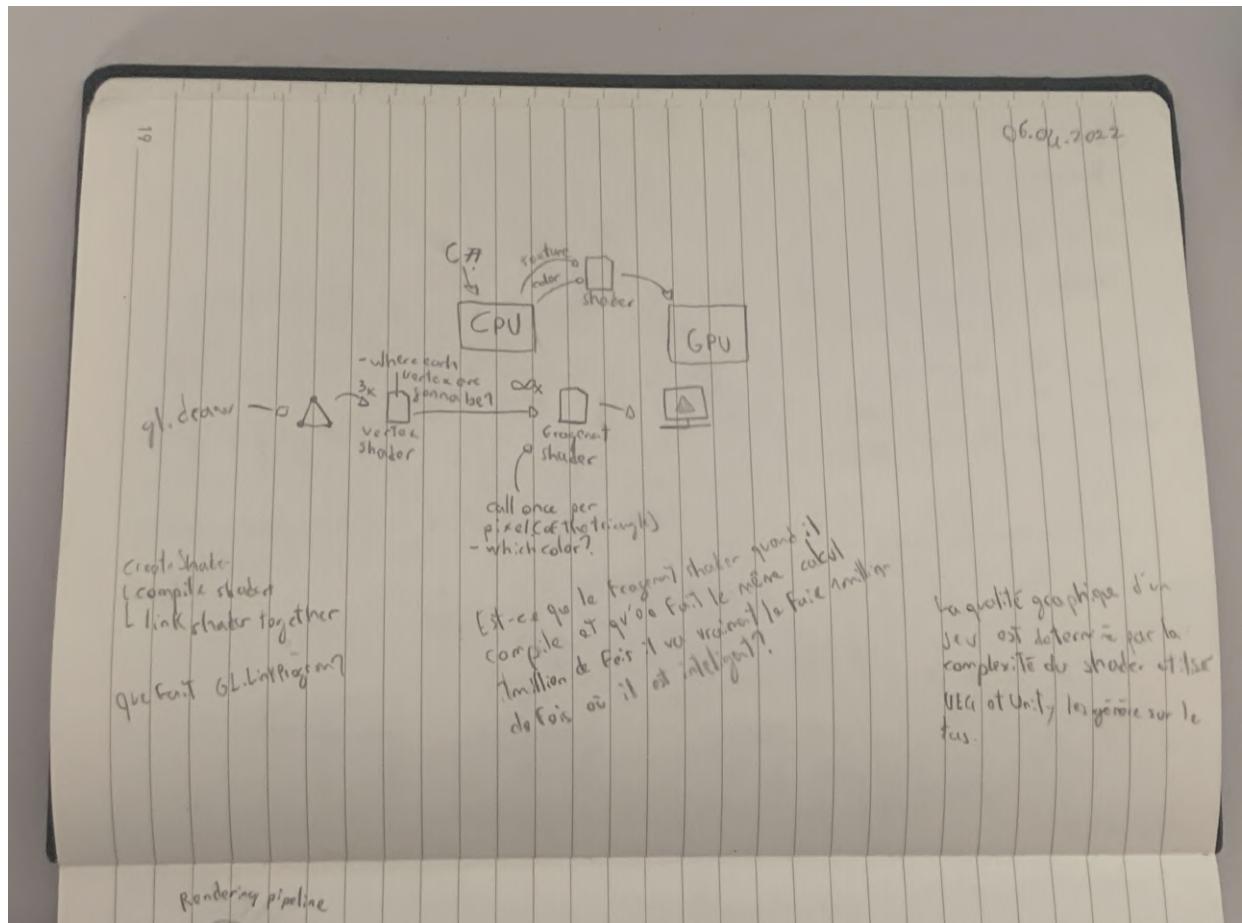
Je décide de faire une macro Visual Studio qui va copier pour moi les fichiers à chaque fois que je compile :

```
if not exist $(ProjectDir)$(OutDir)Shaders mkdir $(ProjectDir)$(OutDir)Shaders
copy "$(ProjectDir)Shaders\*" "$(ProjectDir)$(OutDir)Shaders\"
```

M. Schmid est passé pour voir si tout se passée bien.

Idées de la journée :

- Pour chaque diagramme fait, d'un côté mettre le diagramme et de l'autre faire une analogie au diagramme (ou un exemple)



[Fig 59]:Journal papier 6 avril

OEF → Hé

J'essaie d'afficher un rectangle grâce à deux triangles, seulement le programme plante au lancement.

Finalement, je remarque que mon erreur projet de ma liste de sommets. J'ai oublié une virgule :

```
public readonly float[] _vertices =
{
    0.5f, 0.5f, 0.0f, // Top-Right
    0.5f, -0.5f, 0.0f // <- ici // Bottom-Right
    -0.5f, -0.5f, 0.0f, // Bottom-Left
```

```
-0.5f, 0.5f, 0.0f, // Top-Left
};
```

Et vue que le float d'après a un signe - le compilateur crois que c'est un calcul.

Pour créer l'interface, je vais utiliser [Dear ImGui](#) (plus précisément [ImGui.NET](#)).

J'ai fini implémenter l'interface. ImGui oblige à faire du code spaghetti.

OF s → → | è

Pour l'importation de fichier, je dois lire les propriétés de chaque balise SVG. Pour ce faire, je vais utiliser du regex. (en utilisant le site [regex101](#))

Le fichier que j'utilisais pour tester mon programme n'était pas au bon format. Par exemple, pour la commande M, chaque position est définie comme suit (x, y). Mon fichier SVG lui était comme ça (x y).

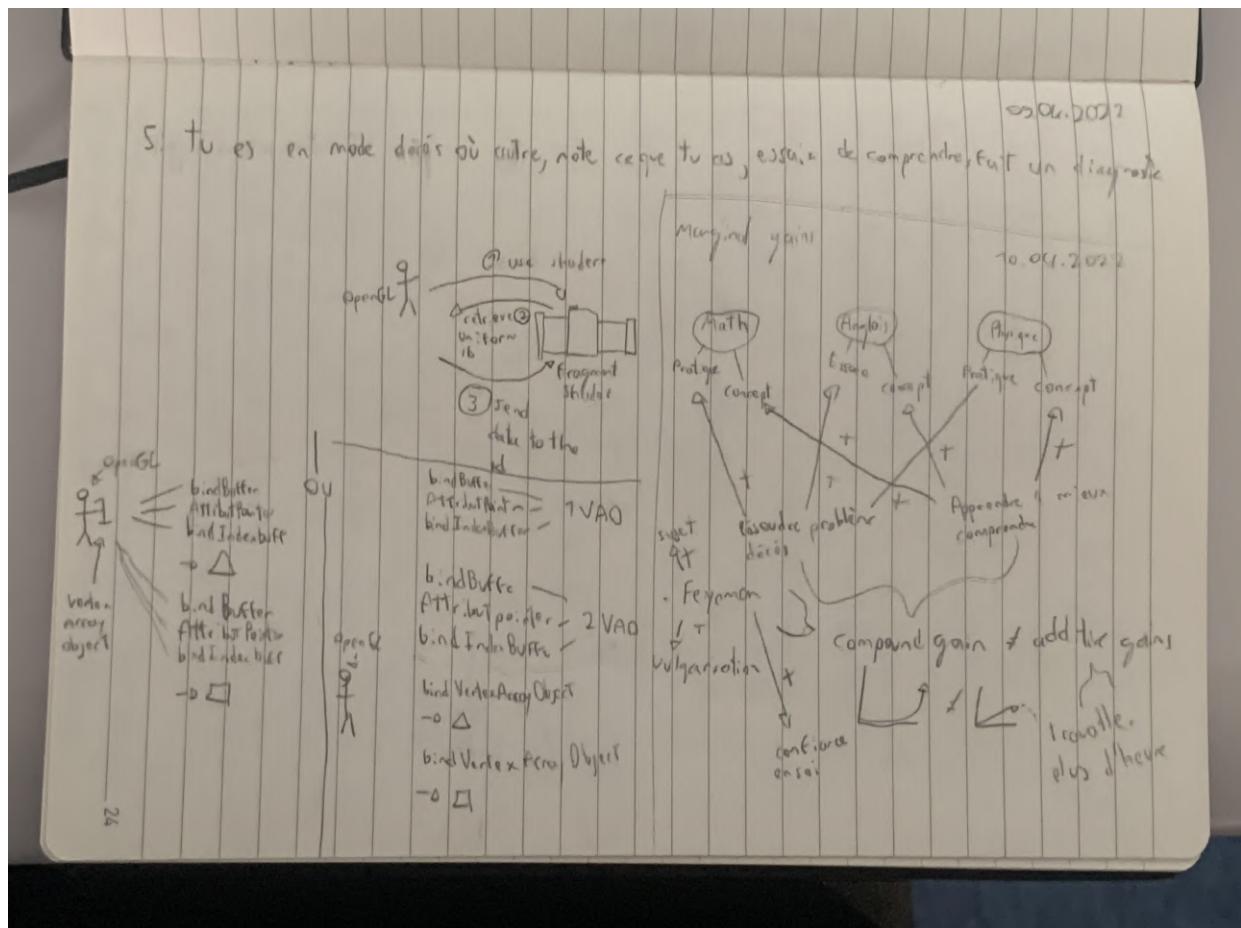
Je ne m'étais pas rendu compte, car le navigateur web que j'utilisais arrivais à quand même comprendre les coordonnées. Pour mon programme, je vais préférer utiliser la norme stricte.

J'ai décidé que tous les nombres des paramètres de propriétés (ex : H 10 dans <path d="H 10" />) seront considérés comme entier, je ne vais pas prendre en compte les [différentes unités](#), car ça va me prendre beaucoup de temps alors que ce n'est pas l'essentiel du travail.

Au départ pour l'implémentation de l'[attribut d](#) je voulais mettre les noms des paramètres en nom de classe. Sauf que je me suis retrouvé avec des classes M, H, V, L, ce qui n'est pas lisible. J'ai décidé de les changer pour mettre les noms, l'avantages et que c'est beaucoup plus agréable à lire où implémenter, le désavantages étant que si une personne souhaite reprendre mon code, il devra savoir que HorizontalLineTo correspond à H dans un fichier SVG. (les noms que j'ai utilisés sont ceux nommés officiellement dans [la documentation officielle](#)).

Idée de la journée :

- Faire une API SVG complète et la poster sur NuGet (et GitHub)



[Fig 60]:Journal papier 8 avril

CSGh → BB è

Récapitulatif de la première semaine :

- Le processus d'apprentissage que j'ai appliqué m'a énormément aidé à tout comprendre
 - J'ai pris du retard justement parce que je voulais comprendre un domaine que je ne connaissais pas (et que ce domaine est très conceptuel), mais je pense que ce retard va être rattrapé, car je ne vais pas avoir besoin de repasser sur ce que j'ai vu
 - Je n'ai pas bien avancé la documentation, j'aurais pu y passer plus de temps

Comme écrits dans mon mémoire, chaque fin de semaine, j'ai décidé de faire une analyse de la semaine.

J'ai :

- Relis le journal de bord
 - Regarder les tâches en cours et le planning

Ça me permet de savoir chaque semaine si je suis toujours dans le cadre du diplôme (au niveau du temps, qualité de travail, etc.).

Aujourd'hui je voulais commencer la journée par mettre en place une gestion des erreurs. Par défaut, OpenGL n'affiche pratiquement jamais des erreurs.

En cherchant un peu, je tombe sur deux méthodes :

- [glGetError](#)
- [glDebugMessageCallback](#)

La mise en place de [glGetError](#) est très rapide, en trois lignes, j'ai une gestion d'erreur fonctionnelle. Cependant, elle n'est pas agréable à utiliser. C'est au développeur de demander à OpenGL s'il y a une erreur. Par conséquent, si on veut déboguer, il faut appeler cette méthode après chaque utilisation de OpenGL.

[glDebugMessageCallback](#) vient régler ce problème, elle est plus récente et permet d'afficher des erreurs complètes. Ici on a simplement à lui donner une fonction de callback. Comme ça, c'est à OpenGL d'avertir le développeur s'il y a une erreur (ce qui est plus logique).

Dans l'article, l'auteur qu'il est possible d'ajouter un décorateur `[DebuggerStepThrough]` à la fonction pour que le debugger crash au niveau de la ligne de code qui a déclencher l'erreur OpenGL. (au lieu de l'exception dans la fonction de callback)

Pour la convention d'écriture de shaders je vais me baser sur [The Book of Shaders](#).

En computer graphics, les éléments sont formés à base de triangle. Pour afficher un triangle ou un carré, c'est simple, mais pour afficher un cercle, je dois avoir des centaines de triangles. Mon programme étant en 2D, pour un cercle, je peux simplement le gérer comme un rectangle et ce sera uniquement au niveau du shader que je vais afficher les bons pixels pour permettre son affichage. De plus, j'aurais directement les contours de sélections qui seront faits pour moi.

Il s'avère que le [FilePicker.cs](#) que j'avais trouvé le premier jour possède des problèmes d'affichages, les noms des dossiers et des fichiers ne s'affiche pas bien. En debuguant le code, je remarque que les méthodes `ImGui.PushStyleColor()` et `ImGui.PopStyleColor()` sont utilisé.

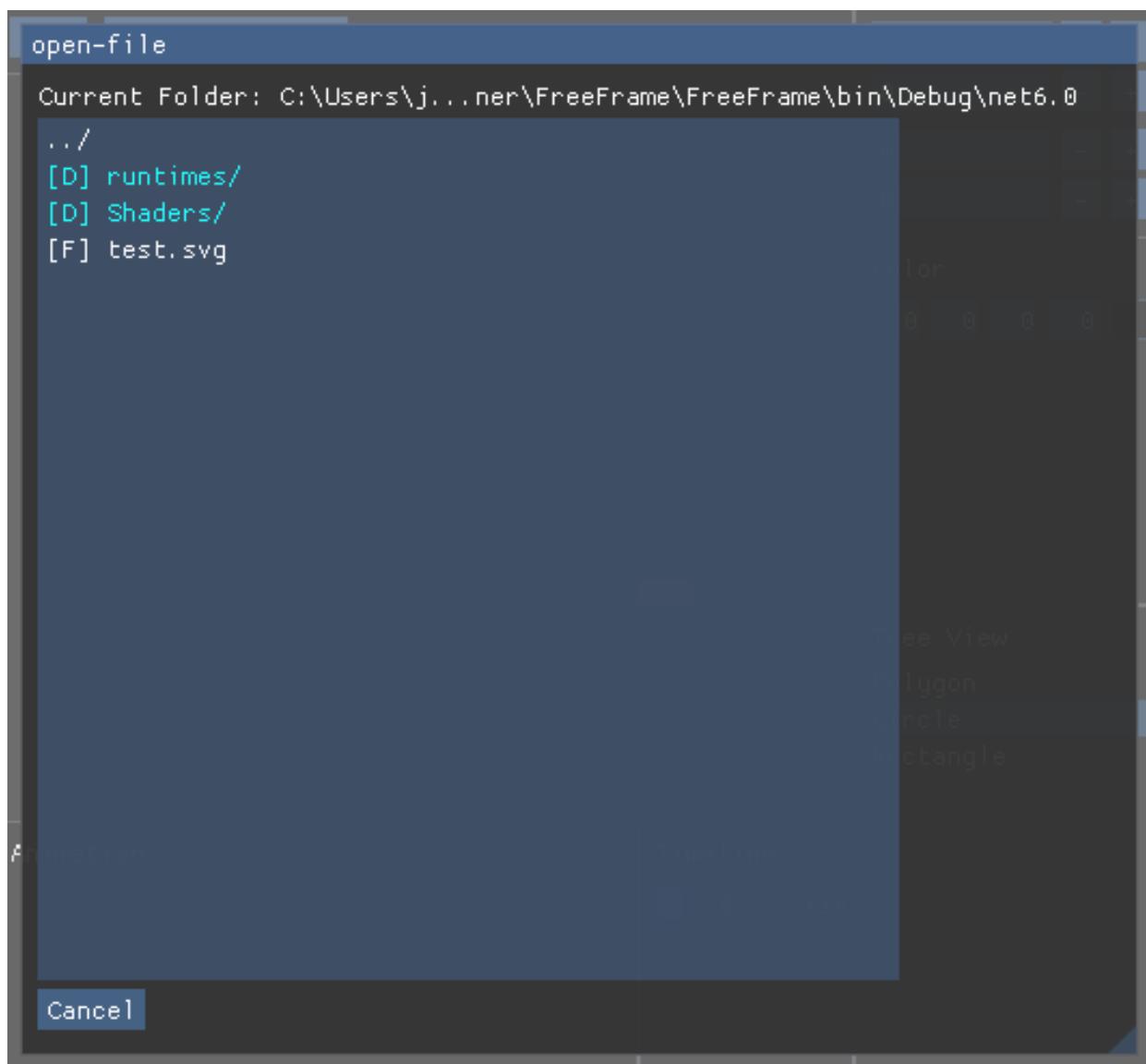
J'en ai également profité pour modifier quelques couleurs.

Avant

Après



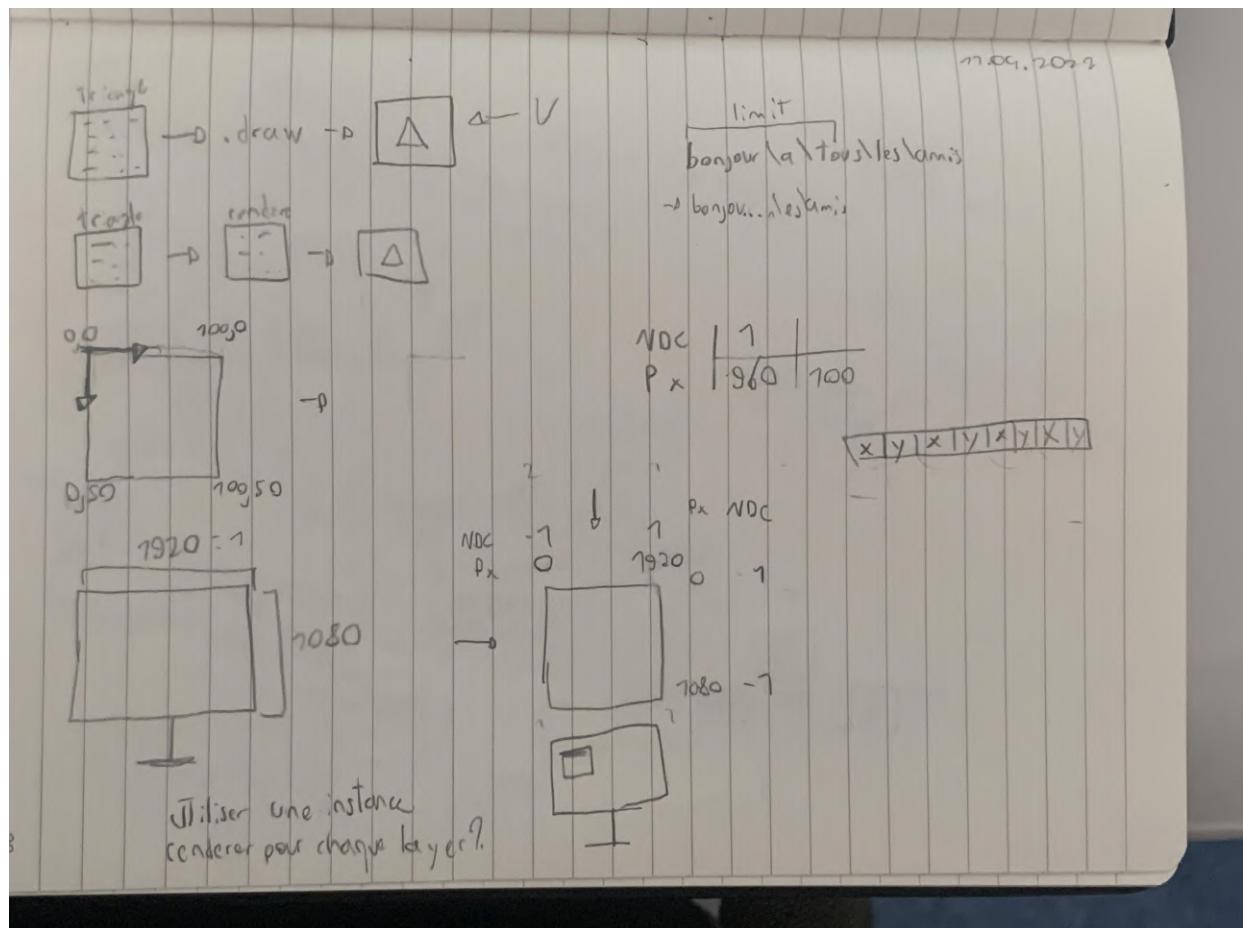
[Fig 61]:FileChooser.cs avant



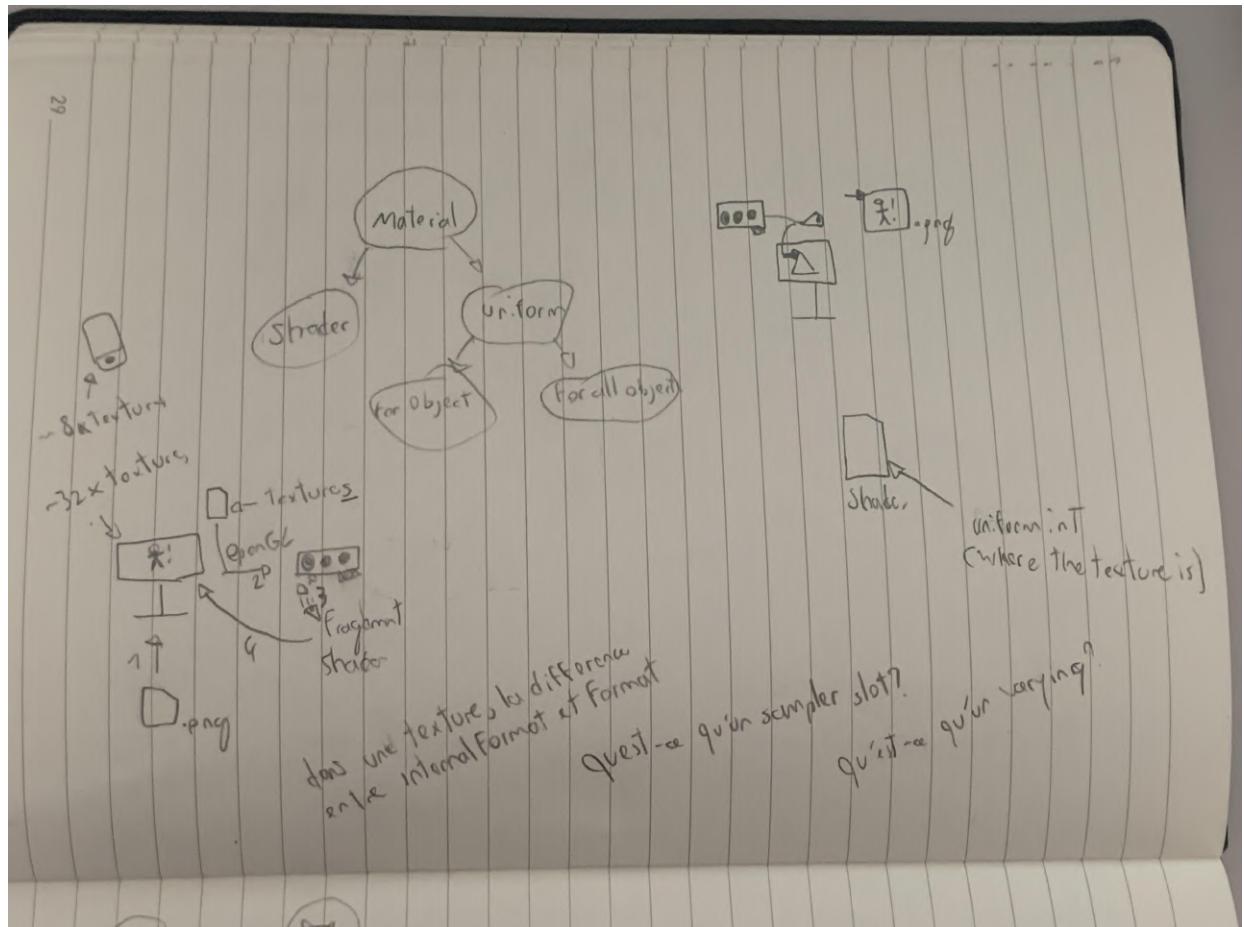
[Fig 62]:FileChooser.cs Après

Il était également impossible de revenir en arrière et le chemin du fichier actuel n'affiché pas les dossiers plus haut.

Mes tests ne passent plus, je suis en train d'investiguer. Le problème est que vue que maintenant les classes enfants de `Shape`, bind par défaut un VBO, VAO et IBO. Et que pour binder des éléments OpenGL il faut qu'un contexte d'OpenGL soit actif. Après beaucoup de recherche sur comment créer un contexte OpenGL rapidement, sans succès, je décide de simplement ajouter une nouvelle méthode pour binder à la classe abstraite `Shape`.



[Fig 63]:Journal papier 11 avril



[Fig 64]:Journal papier 11 avril 2

OHi è → BCè

Aujourd'hui j'ai décidé de principalement m'occuper de la documentation. J'aimerais commencer à écrire chaque partie de la documentation.

J'ai écrit presque dix pages, j'ai également essayé de chercher des tutoriels de professionnels sur comment écrire des documentations interne. La meilleure que j'ai pu trouver est celui sur le [site d'un développeur](#).

OHi ffi → BDè

Hier soir j'ai continué à regarder des vidéos sur OpenGL sur le blending et les matrices.

Aujourd'hui je dois commencer à travailler sur l'outil de modification d'éléments. Avant tout, j'aimerais changer comment je gère la conversion pixel vers NDC. J'avais créé une fonction qui le faisait :

```
public float[] ConvertToNDC(params int[] vertexPositions)
{
    float[] result = new float[vertexPositions.Length];
    for (int i = 0; i < vertexPositions.Length; i += 2)
    {
        result[i] = (float)vertexPositions[i] / ClientSize.X / 2;
        result[i + 1] = (float)vertexPositions[i + 1] / ClientSize.Y / 2;
    }
    return result;
}
```

Mais c'est en fait beaucoup plus logique d'utiliser des matrices. Pour utiliser les matrices, j'ai importé la librairie native `OpenTK.Mathematics` qui possède le struct `Matrix4`. Elle possède également la méthode `CreateOrthographicOffCenter(. . .)` qui va me permettre de faire la conversion en orthographic très facilement.

```
Matrix4 matrix = Matrix4.CreateOrthographicOffCenter(0, _window.ClientSize.X,
    _window.ClientSize.Y, 0, -1.0f, 1.0f);
```

J'ai bloqué quasiment une heure sur le fait que j'avais mis le calcul de la matrice à l'envers.

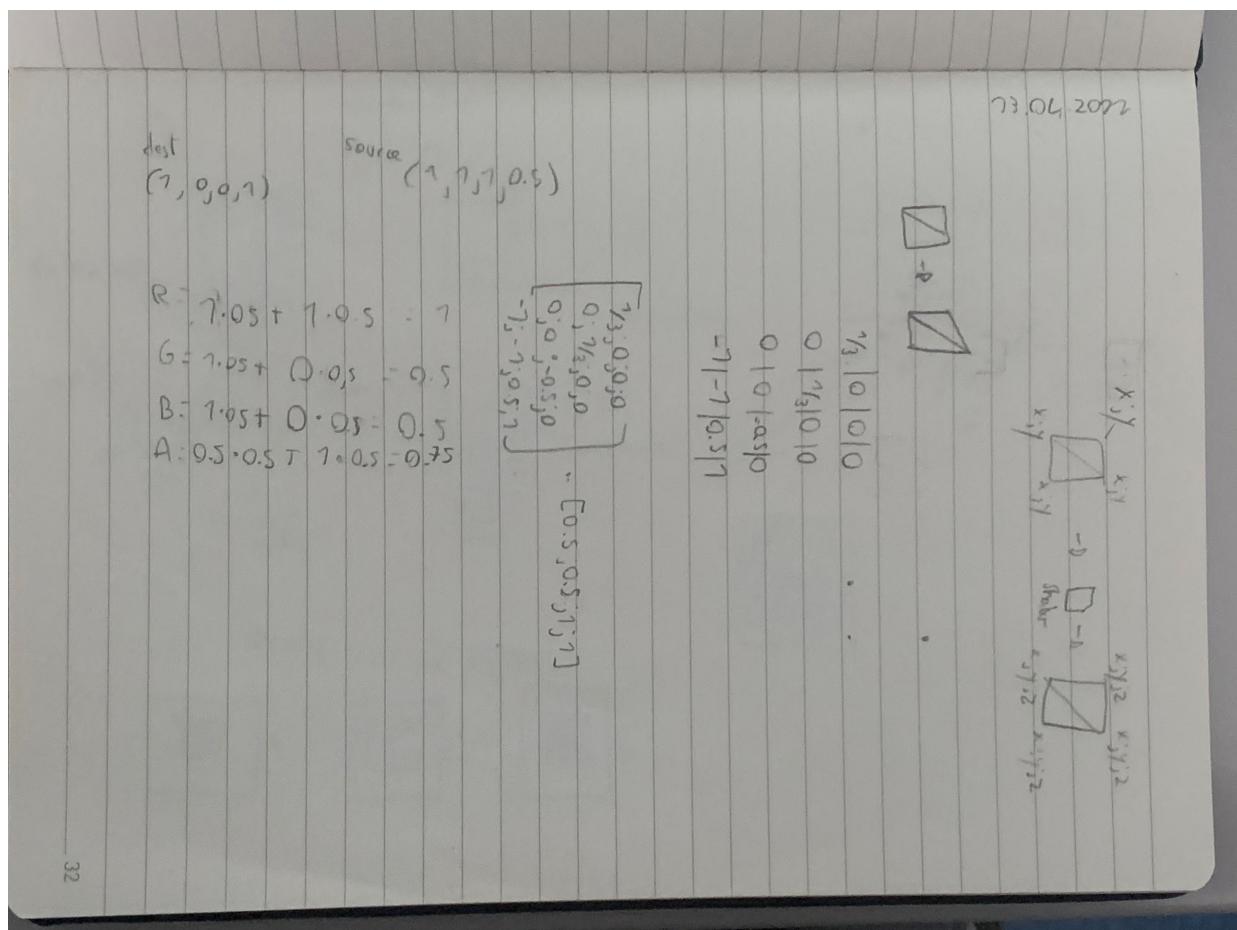
J'ai également découvert un nouveau bug, si je double-clique dans la fenêtre de choix de fichier. J'ai donc créé une nouvelle issue.

Résultat actuel de l'application

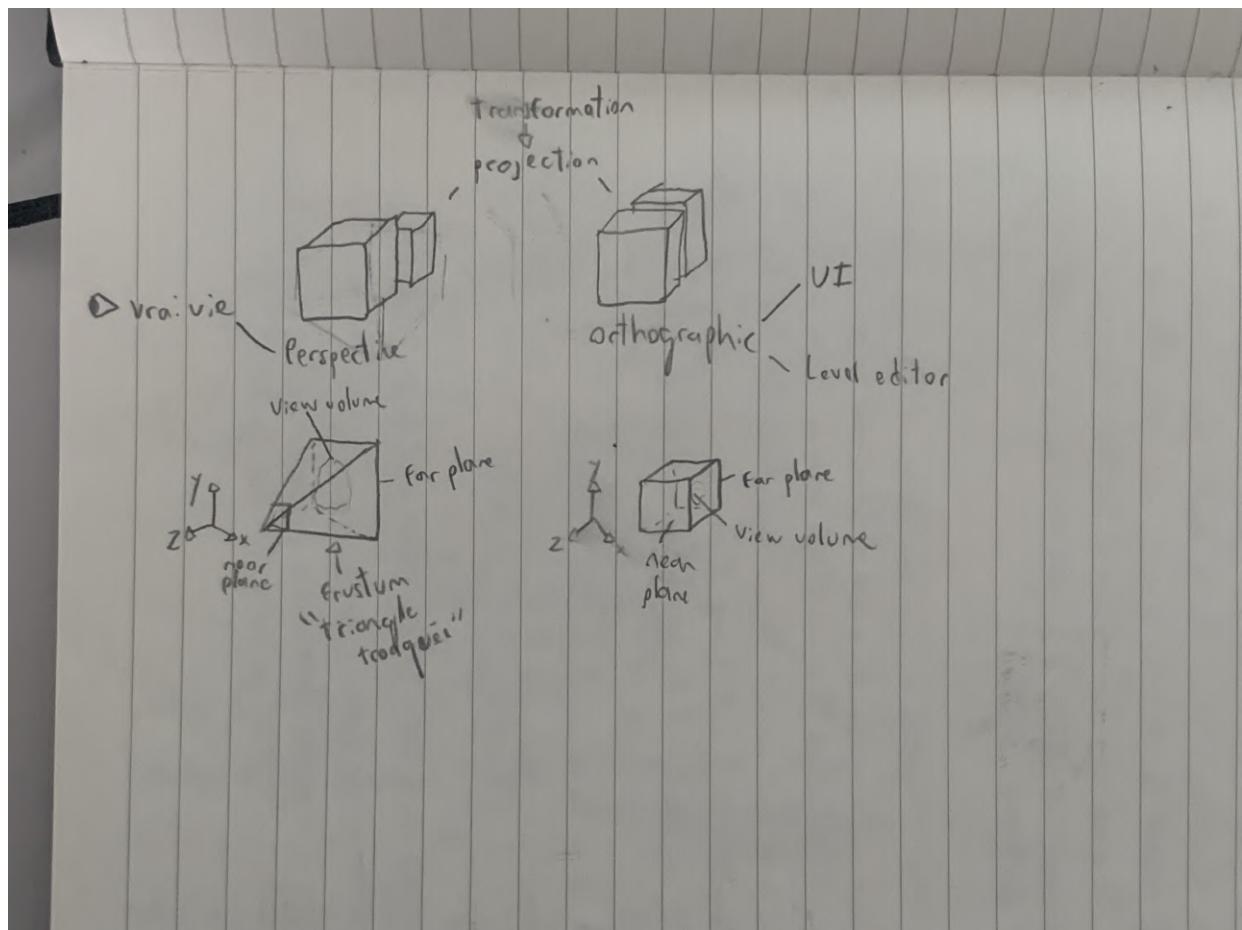
[Fig 65]:Résultat actuel de l'application

Je viens d'ajouter la fonctionnalité de pouvoir modifier les paramètres d'un élément depuis l'interface.

Maintenant je dois pouvoir rendre un élément cliquable. Pour un rectangle il n'y aura aucun problème, mais pour une ligne en diagonal ou un rond je vais comment pour savoir si l'utilisateur clique sur le rond affiché où le rectangle de gestion ? Étant donné qu'un rond est affiché à partir du shader je vais devoir implémenter une hit box pour chaque élément.



[Fig 66]:Journal papier 13 avril 1



[Fig 67]:Journal papier 13 avril 2

CJ h → CF è

Pour afficher des lignes, j'ai simplement à modifier la méthode d'affichage, soit :

```
GL.DrawElements(PrimitiveType.Lines, _indexCount, DrawElementsType.UnsignedInt,
0);
```

Au lieu de :

```
GL.DrawElements(PrimitiveType.Triangles, _indexCount,
DrawElementsType.UnsignedInt, 0);
```



[Fig 68]:Ligne sur l'espace

J'ai un problème, l'élément `<path>` permet à la fois d'afficher des formes pleines et creuse. Sauf les méthodes d'affichage ne seront pas les mêmes pour les formes pleines et creuse.

Autre problème, les éléments polygone peuvent se passer dessus :

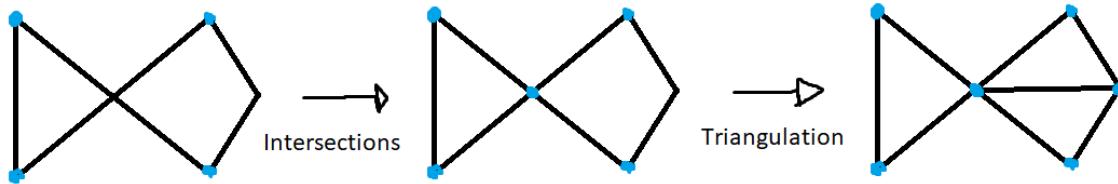


[Fig 69]:Polygone qui se passe dessus

Ici le point du milieu doit être calculé en regardant les lignes qui s'intercepte, je dois créer un algorithme qui le fait. À partir de ça, je peux ensuite facilement créer les différents triangles.

Seulement si le polygone plein possède plus de 3 points, je ne peux pas directement créer un triangle.

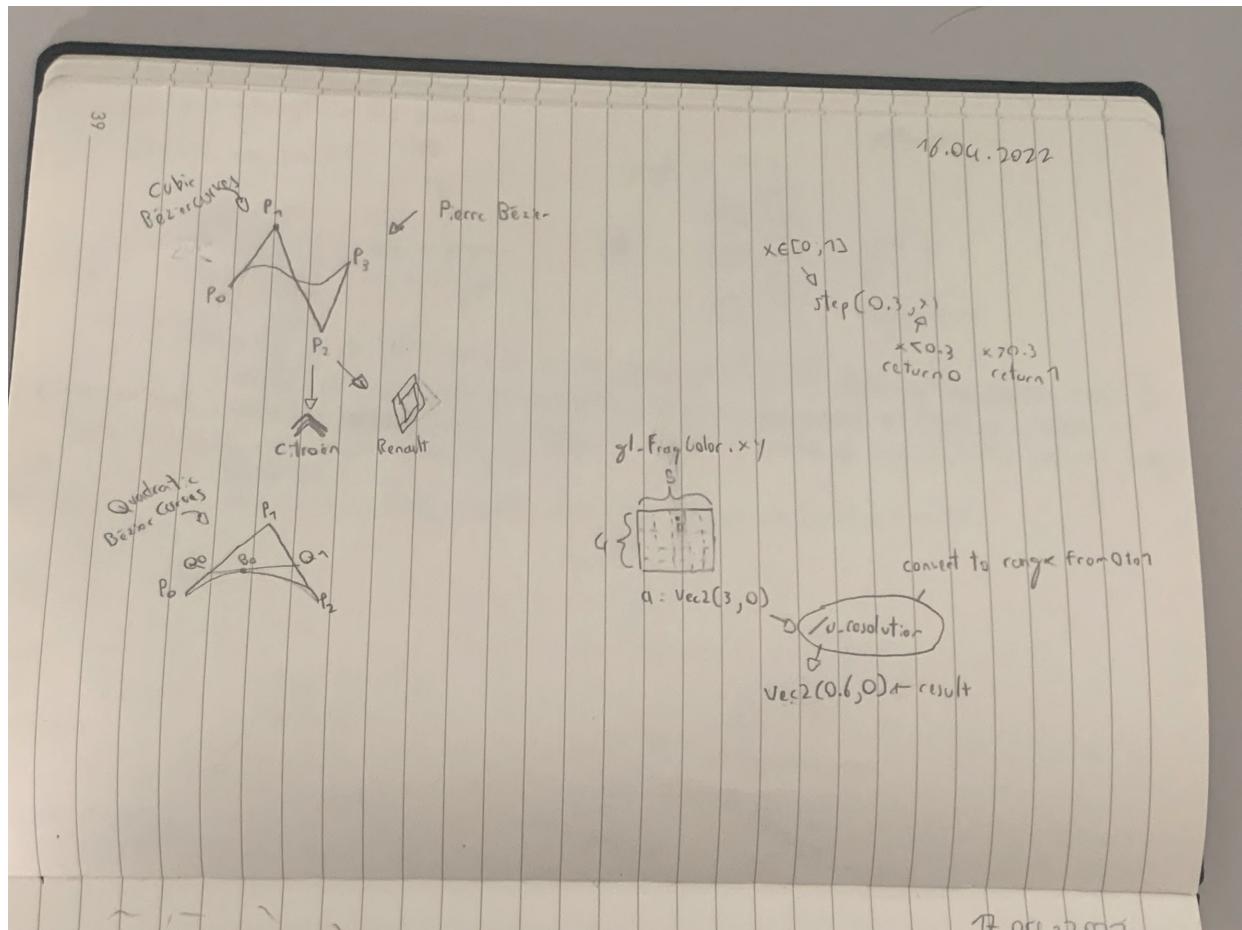
Je pense avoir trouvé une solution après beaucoup de recherche. Si le polygone est plein, je vais créer les points d'intersections qui manque et ensuite utiliser un algorithme de triangulation pour les polygones fermée (par exemple il y en a deux ci-dessous).



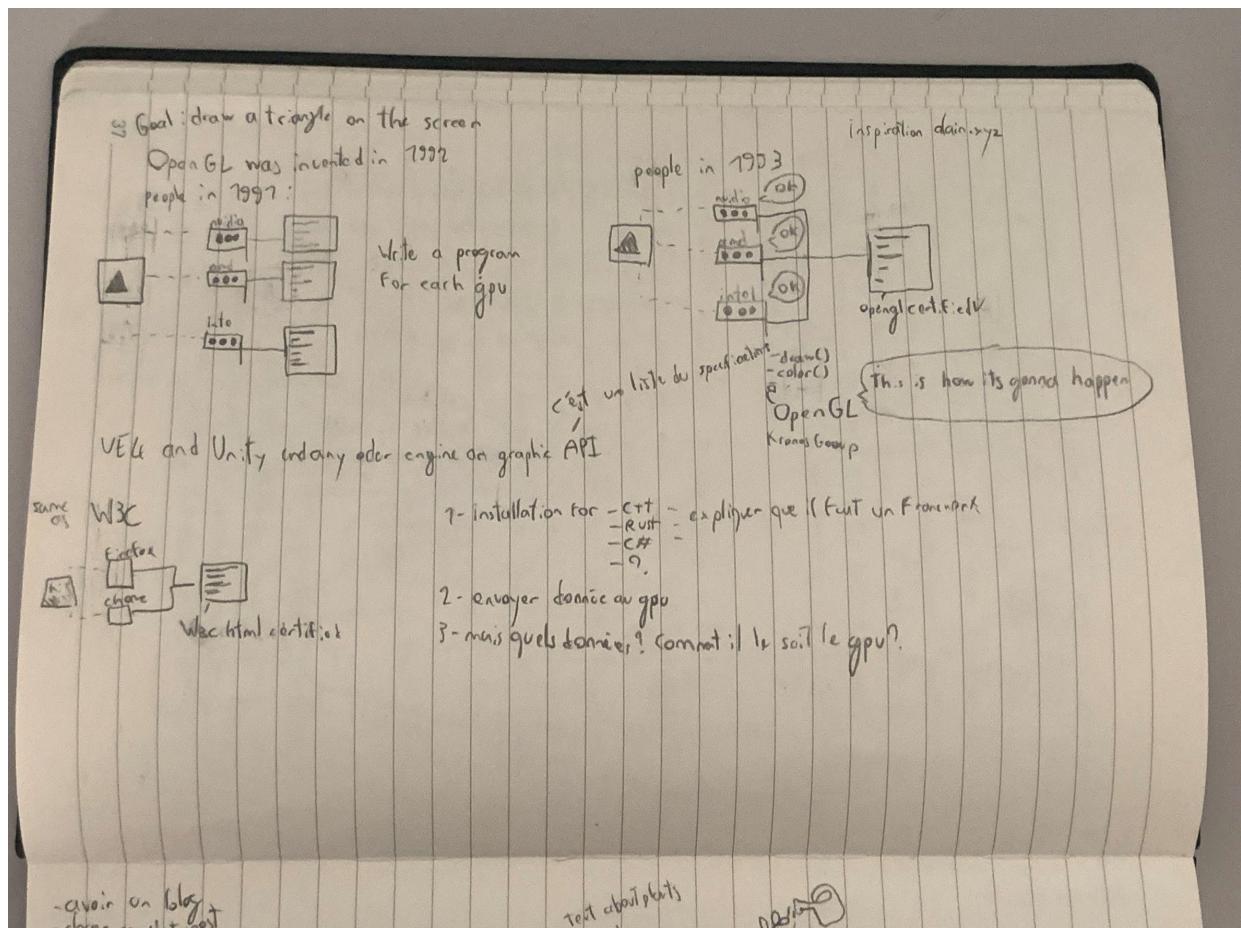
[Fig 70]:Deux étape pour convertir

Si le polygone est creux, je vais simplement afficher avec des lignes simples.

À noter que pendant les vacances qui viennent de se terminer, j'avais fait beaucoup de recherches et de tests pour afficher des polygones et courbes de Bézier à l'écran en utilisant uniquement le shader. Il s'est avéré que ce n'est pas pratique, car certain élément SVG dépendent d'un élément précédent et le fait que les informations de la courbe soit exclusivement décider au niveau du shader ne facilite pas la tâche.



[Fig 71]:Journal papier 25 avril



[Fig 72]:Journal papier 25 avril 2

C9BA i è → CGè

Je viens de remarquer que je n'ai pas pris en compte les valeurs négatives dans mon expression régulière pour l'élément `d` de `<path>`.

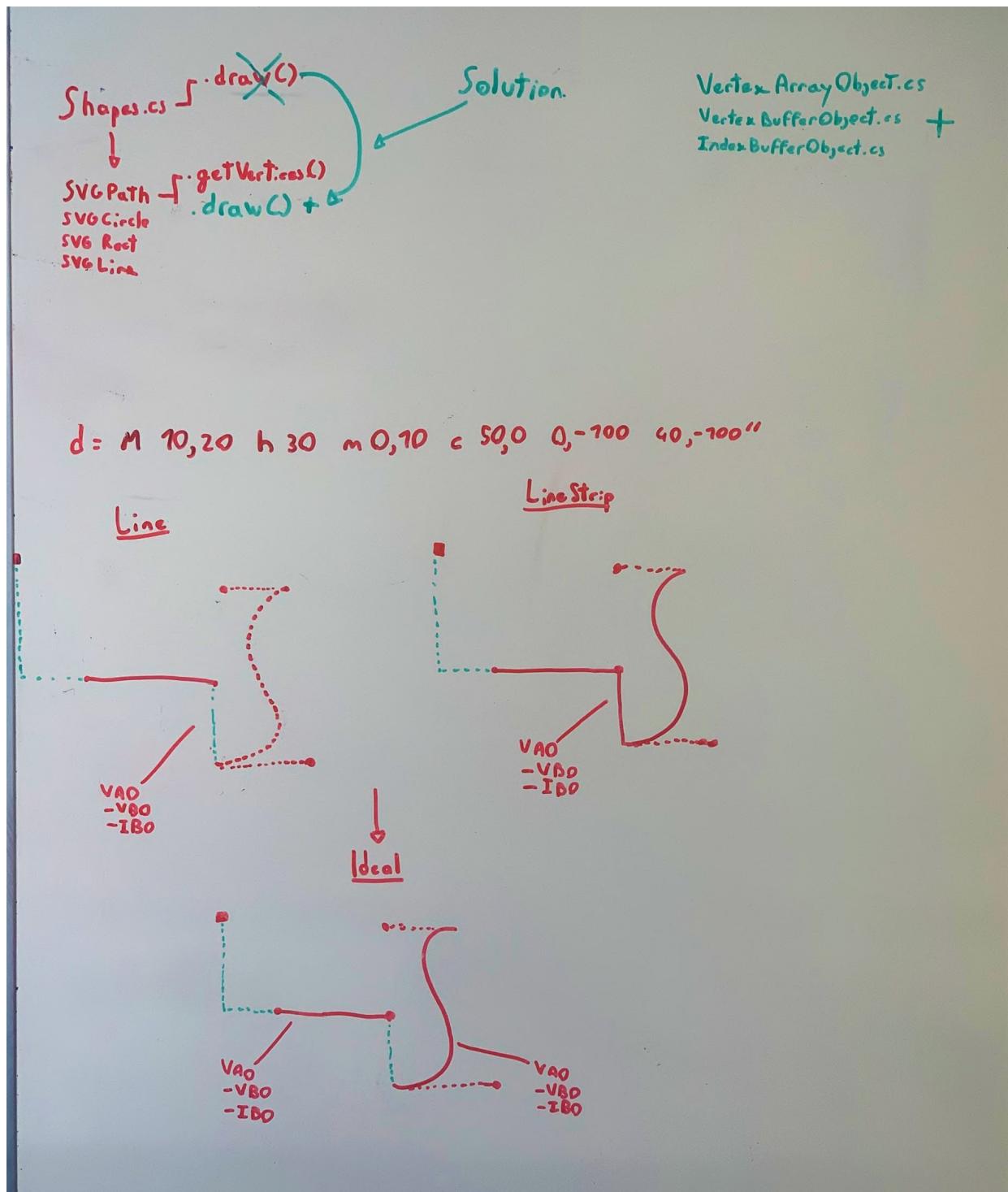
Je viens de tomber sur un problème, dans mon programme si l'utilisateur utilise l'action `M` dans l'élément `d` de `<path>`, le programme va dessiner les points entre chaque `M`, ce qui n'ai pas voulu car `M` est simplement le déplacement du curseur de dessin de `d`.

M. Bonvin est passé, j'en ai profité pour lui discuter des recherches que j'ai faites sur comment je vais détecter la sélection d'un élément lors d'une clique. Il m'a également fait découvrir **QCAD** qui est un outil de **CAD** très puissant.

Je remarque un autre problème, pour rendre des lignes, OpenGL propose trois modes d'affichages :

- [Lines](#)
- [LineStrip](#)
- [LineLoop](#)

Lines créer des lignes de deux points maximum alors que *LineStrip* connecte toutes les lignes ensemble (et *LineLoop* rajoute simplement une ligne entre le dernier et premier point). Le problème étant que pour ma structure de code actuelle il est impossible d'utiliser deux fois la méthode de rendu `Draw()` pour une forme. J'avais du mal à réfléchir donc j'ai préféré schématiser le problème pour avoir une vue d'ensemble.



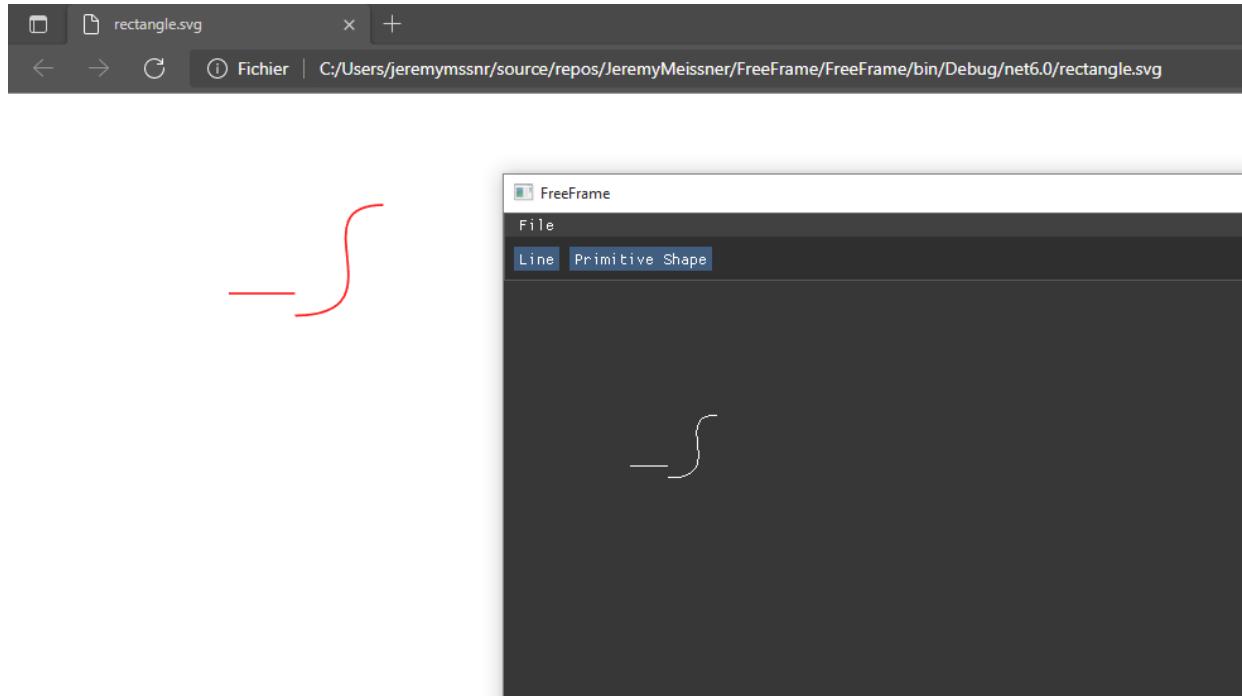
[Fig 73]: Schéma sur tableau blanc de mon problème

J'ai donc finalement trouvé que j'aller simplement mettre en `virtual` la méthode `Draw()` et que je vais ensuite gérer l'affichage de chaque forme séparément grâce à `override`.

Je vais également devoir créer des classes pour les différents objets d'OpenGL. Jusqu'à présent je n'en avais pas l'utilisé, car je les utilisais qu'à un seul endroit, mais maintenant je vais devoir en créer plusieurs instances indépendantes.

OBBI ffi → CHÈ

J'ai continué et résolu le problème d'hier.



[Fig 74]: Démonstration fonctionnement de l'application

Je vais maintenant implémenter la sélection d'un élément sur l'espace de travail.

Comme expliquer hier, j'ai implémenté un algorithme qui détecte le point d'une forme le plus proche du clique de l'utilisateur.

J'ai d'ailleurs décidé que je vais travailler en `int` dans mon plan de travail, pas en `float`. Je trouve ce choix cohérent, car ça m'évitera des problèmes de compatibilité.

Il est maintenant possible de modifier des rectangles depuis la fenêtre propriété de l'interface graphique (je l'avais déjà fait, mais suite à une restructuration du code constante ça ne fonctionnait plus). J'avais rencontré un petit problème, lorsque je cliquais pour modifier la valeur du champ d'une propriété, ça effectuait également un clique dans l'espace de travail et ça changeait par conséquent la forme sélectionnée. En cherchant je trouve [sur la faq de ImGui la solution](#). ImGui fournit un booléen qui permet de savoir si le clique de la souris a lieu sur une de ses fenêtres.

Je suis en train de me dire que les rectangles svg ne peuvent pas être que deux triangle, car si je rajoute des arrondis au niveau des angles je dois forcement faire de la triangulation a base partir des différents points.

OBCf → A è

En cherchant un peu je suis tombé sur un shader qui permet d'arrondir les angles d'un rectangle.

```
// from https://iquilezles.org/articles/distfunctions
float roundedBoxSDF(vec2 CenterPosition, vec2 Size, float Radius) {
    return length(max(abs(CenterPosition)-Size+Radius, 0.0))-Radius;
}
void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
    // The pixel space scale of the rectangle.
    vec2 size = vec2(300.0f, 300.0f);

    // the pixel space location of the rectangle.
    vec2 location = iMouse.xy;

    // How soft the edges should be (in pixels). Higher values could be used to
    // simulate a drop shadow.
    float edgeSoftness = 1.0f;

    // The radius of the corners (in pixels).
    float radius = (sin(iTime) + 1.0f) * 30.0f;

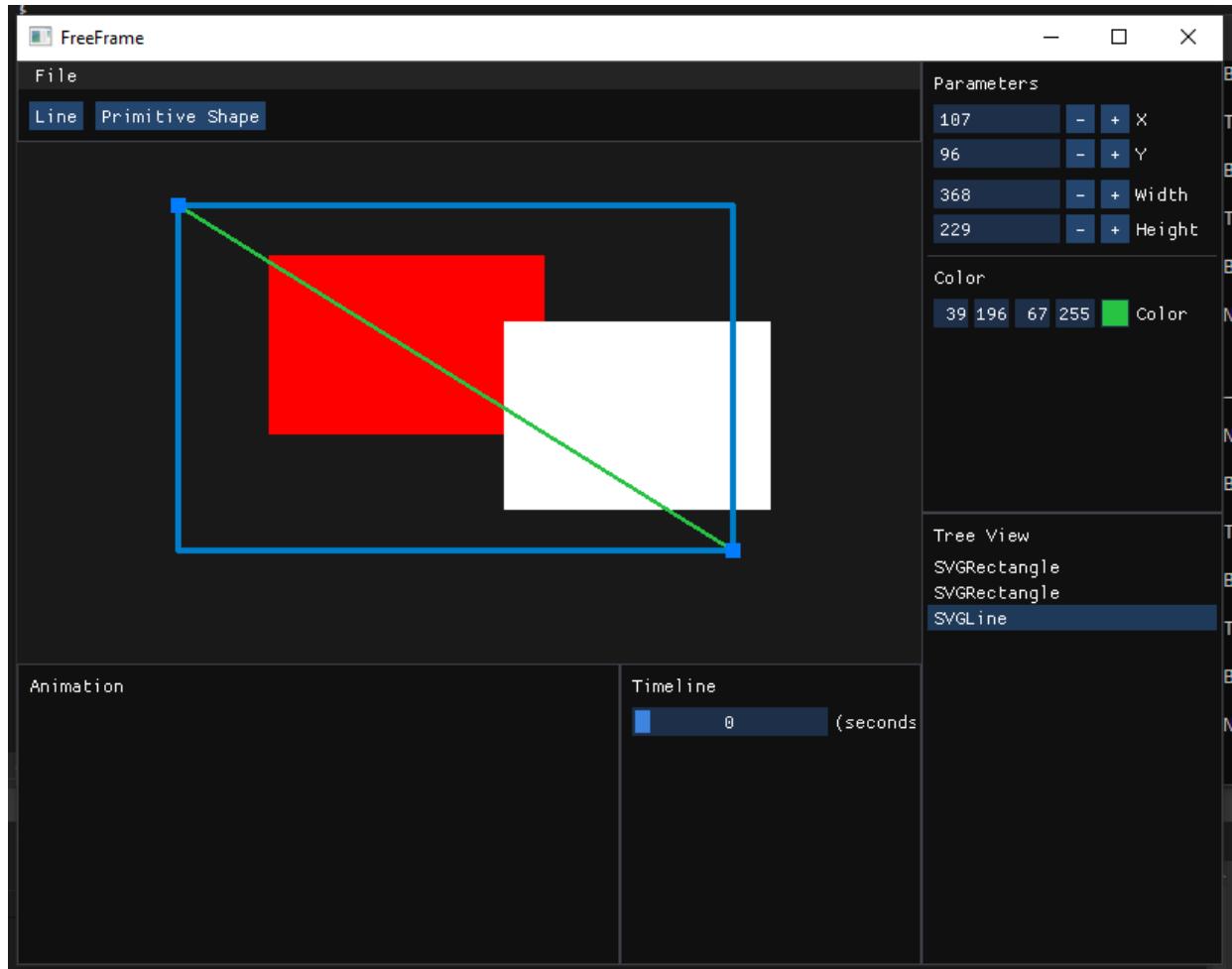
    // Calculate distance to edge.
    float distance = roundedBoxSDF(fragCoord.xy - location - (size/2.0f), size /
2.0f, radius);

    // Smooth the result (free antialiasing).
    float smoothedAlpha = 1.0f-smoothstep(0.0f, edgeSoftness * 2.0f,distance);

    // Return the resultant shape.
    vec4 quadColor = mix(vec4(1.0f, 1.0f, 1.0f, 1.0f), vec4(0.0f, 0.2f, 1.0f,
smoothedAlpha), smoothedAlpha);

    // Apply a drop shadow effect.
    fragColor = quadColor;
}
```

Je viens de rajouter la contrainte de symétrie lors de l'appui sur la touche SHIFT. J'ai d'ailleurs modifié la structure du code, j'ai mis en place trois méthodes `OnLeftMouseUp()`, `OnLeftMouseDown()` et `OnLeftMouseEnter()` qui vont être appelé lorsqu'une des trois conditions de la souris est remplie.



[Fig 75]:Démonstration fonctionnement système decouleurs

J'ai un problème, certains éléments `d` de `<path>` dépendant de l'élément précédent. Je suis en train de me dire que ce serait plus simplement de lorsque j'appelle n'importe quelle méthode d'une classe qui hérite de `DrawAttribute` je dois lui passer en paramètre le `DrawAttribute` qui le précède. Ça m'évitera de faire passer des variables `LastX` et `LastY` un peu partout sachant qu'elles sont uniquement modifiées depuis la classe `SVGPath`. Car dans tous les cas je parcours grâce à une liste les attributs dont je pourrais facilement récupérer le précédent.

OBDs → → CJ è

Aujourd'hui j'ai commencé à réaliser le logo et le poster.

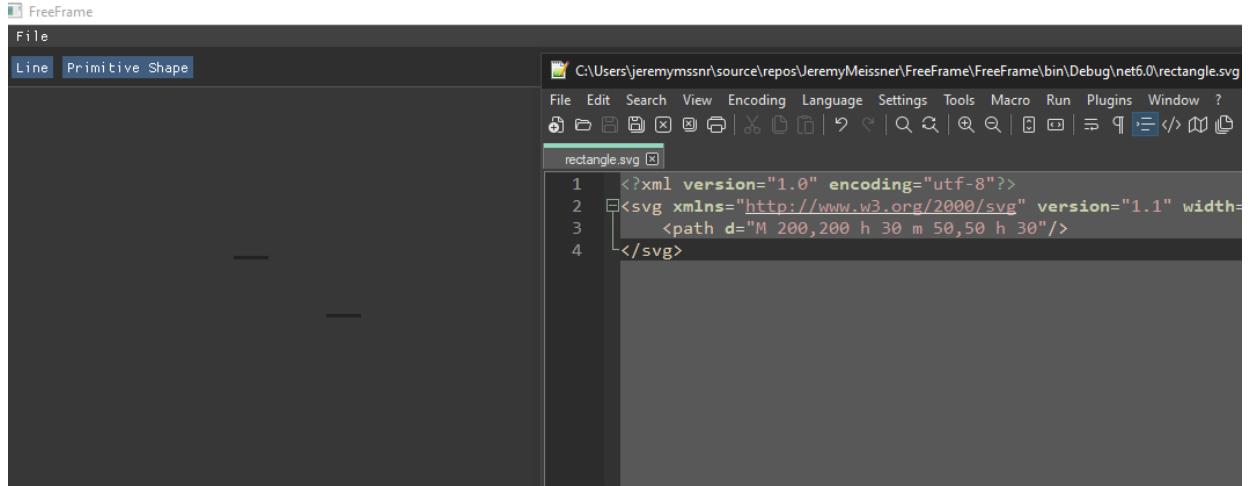
Je souhaite modifier l'interface pour mettre en place des fenêtres avec un [docking](#). Cependant, la librairie de portage que j'utilise [n'a pas encore implémenté la fonctionnalité](#).

OBE h → C è

Je suis en train de mettre en place une structure de code qui puisse modifier n'importe qu'elle forme.

J'ai découvert un nouveau bug que j'ai répertorié sur GitHub.

Je suis en train de bloquer sur la modification des éléments `<path>`.



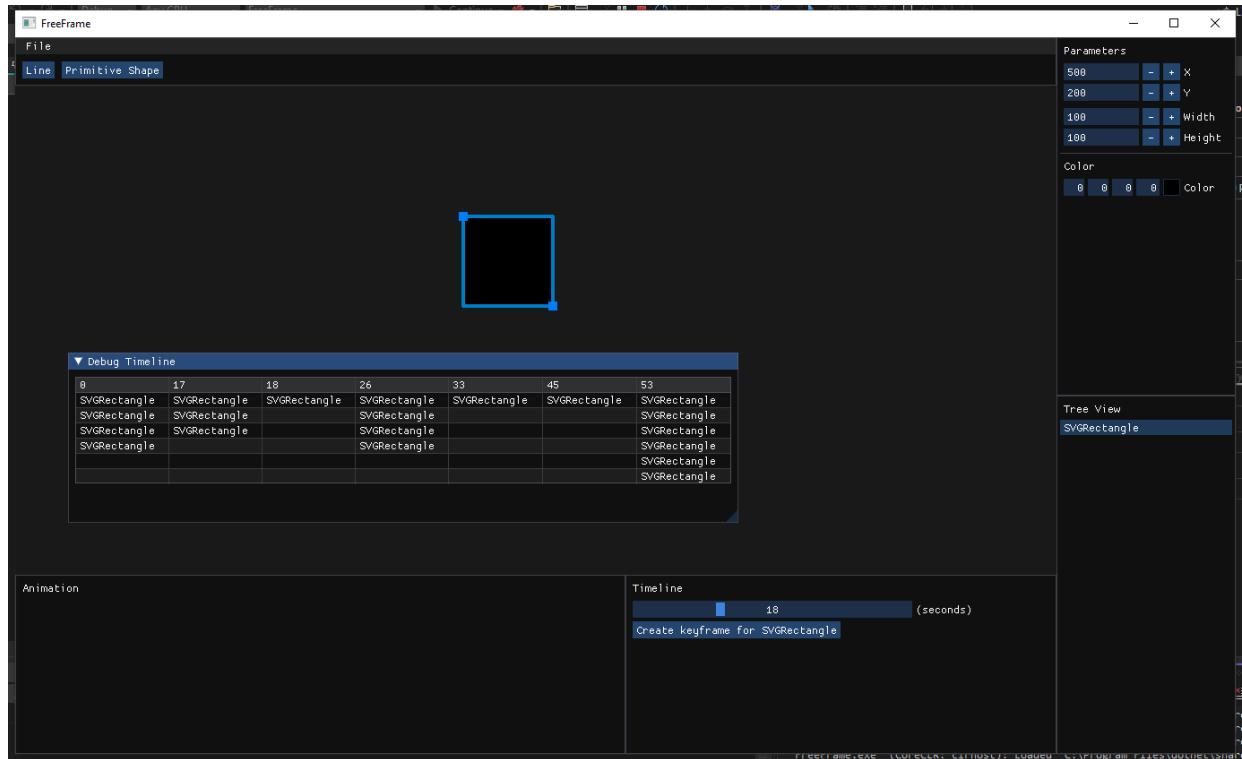
[Fig 76]:Démonstration fonctionnement de path

L'affichage fonctionne parfaitement, mais la redimension et le déplacement ne fonctionnent pas correctement.

OBF i è → D è

Je suis en train de commencer à implémenter la gestion d'animation.

Je viens de terminer un outil de debug qui me permet de savoir les éléments qui ont été modifier dans la timeline.

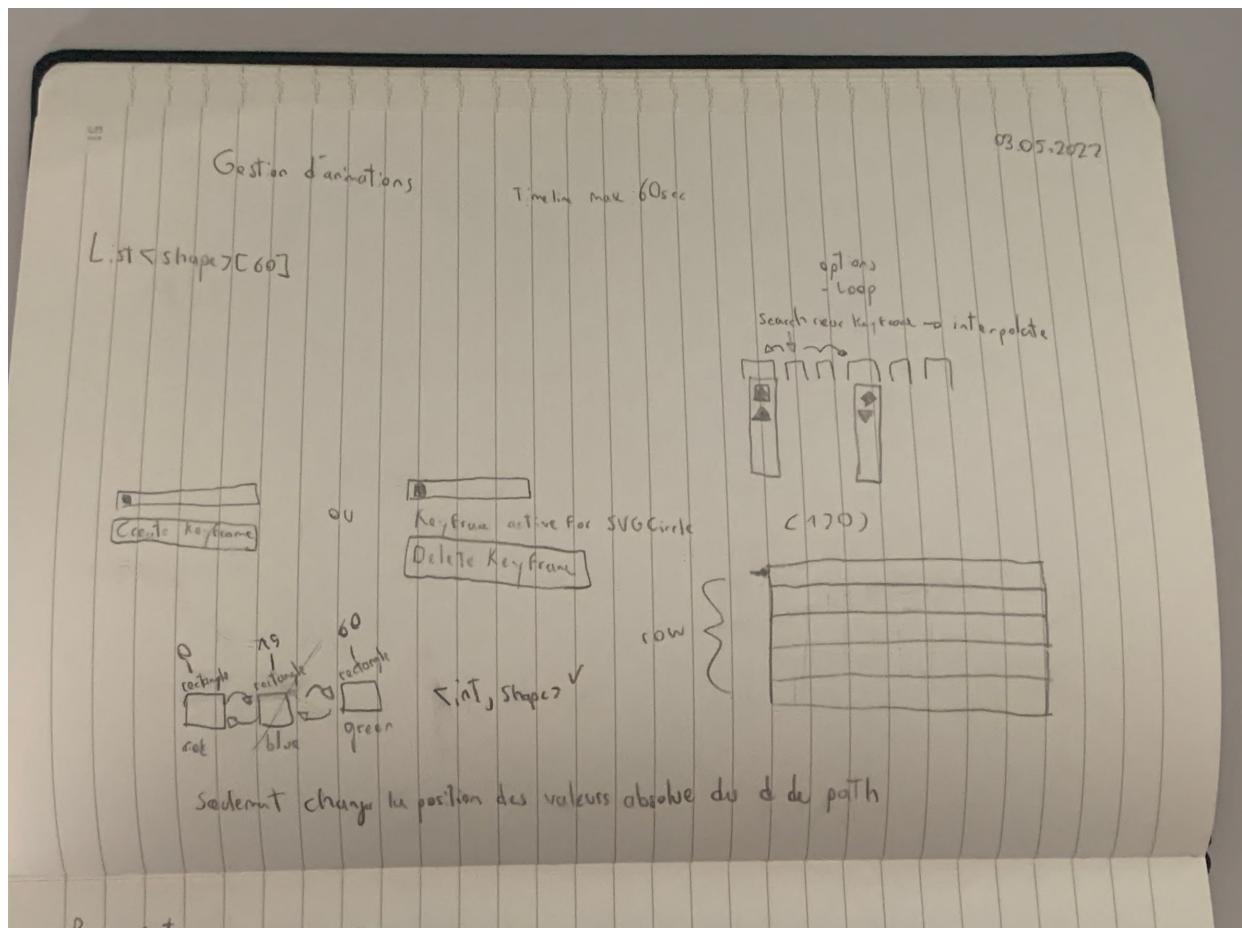


[Fig 77]:Outil de debug en création

L'api pour afficher des tables est mal fait, j'ai pris du temps pour bien le comprendre et l'implémenter correctement.

Je viens de discuter avec M. Schmid du choix que j'ai fait pour gérer les animations. Je souhaitais au départ partir sur une liste chaînée, mais on a finalement vu qu'utiliser un dictionnaire avec comme index le timestep, soit `Dictionary<int, Shape>`.

Je lui ai également parlé du problème des `<path>` que j'avais pour redimensionner. On a finalement conclu que j'avais uniquement besoin d'implémenter le déplacement pour les formes complexe comme `<path>` car il y a beaucoup de façon de le mettre en place et que ce n'était pas le but principal.

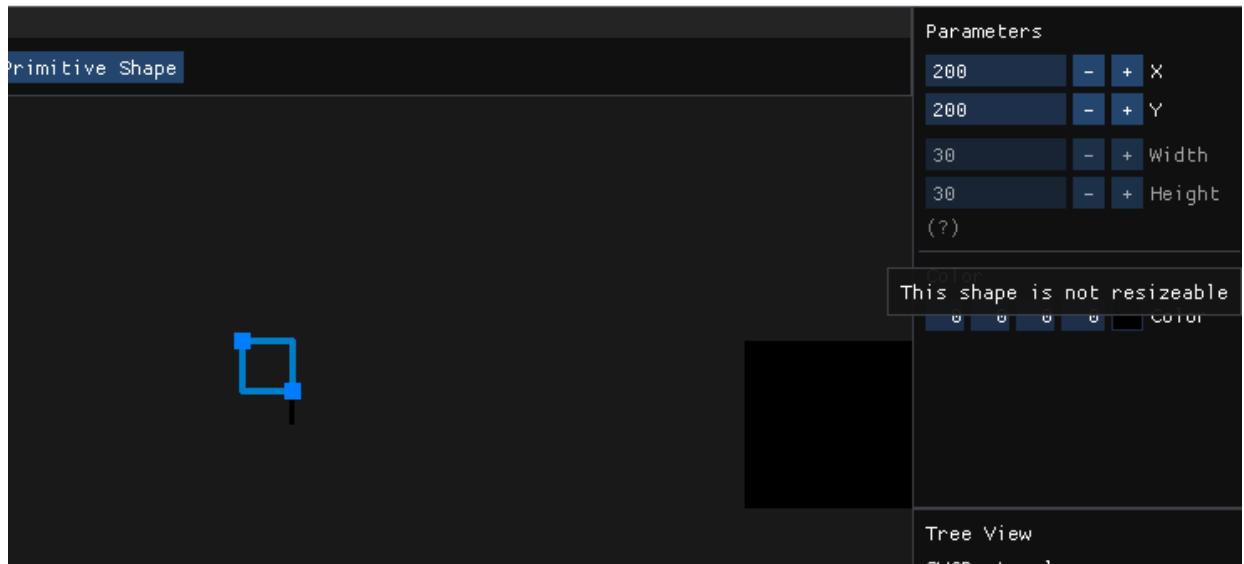


[Fig 78]:Journal papier 3 mai

OBGi ffi → E è

Je vais commencer la journée par régler le problème de déplacement du `<path>`.

J'ai donc désactivé les champs non utilisables et j'ai rajouté un message pour prévenir l'utilisateur que les éléments `<path>` ne sont pas redimensionnables.



[Fig 79]:Problème de path

Pour l'instant, je décide de ne pas afficher les sélecteurs des courbes de Béziers.

J'ai aussi mis à jour le poster.

M. Schmid est venu, il m'a dit qu'il était préférable d'avoir deux façons d'importer :

- Importer un fichier sur l'espace de travail actuel
 - Créer un nouveau projet en supprimant les éléments actuels
-

CSBHf → F è

Aujourd'hui j'ai commencé par implémenter les ronds. Pour cela je vais simplement utiliser un shader spécial qui va afficher les bons pixels afin de créer la forme d'un rond.

Rond compatibles

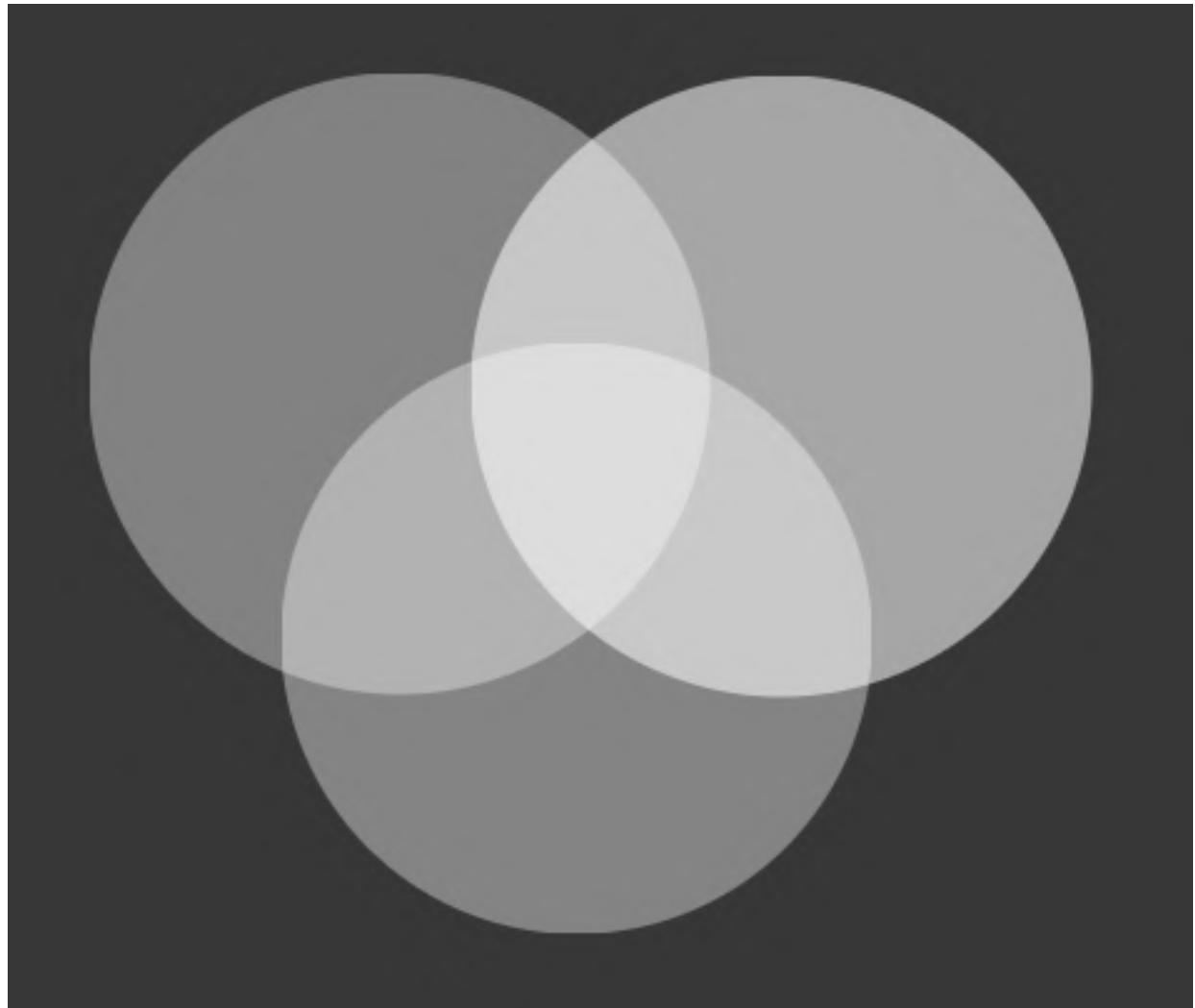
[Fig 80]:Rond compatibles

M. Schmid est passé, je n'avais aucun problème à signaler, je lui ai simplement présenté mon avancement de ce matin.

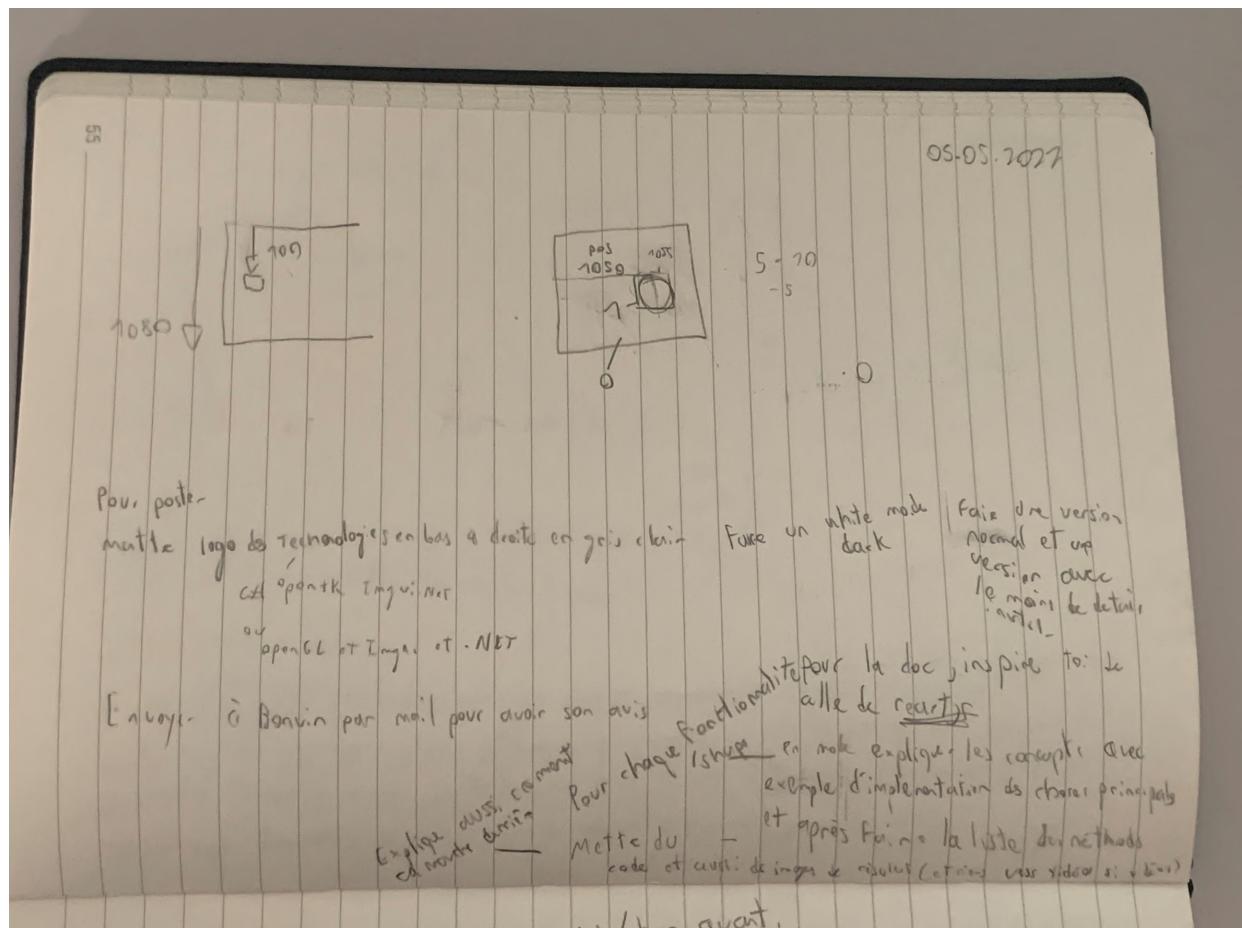
Je viens d'ajouter l'opacité sur tous les éléments

J'avais encore un bug de déplacement pour les `<path>`, j'ai décidé de consacrer du temps pour restructurer le code et pour me permettre normalement d'avoir moins de problème.

Je suis en train d'essayer d'implémenter les bords arrondis des rectangles. Je dois donc créer un shader.



[Fig 81]:Trois rond superposés coupé

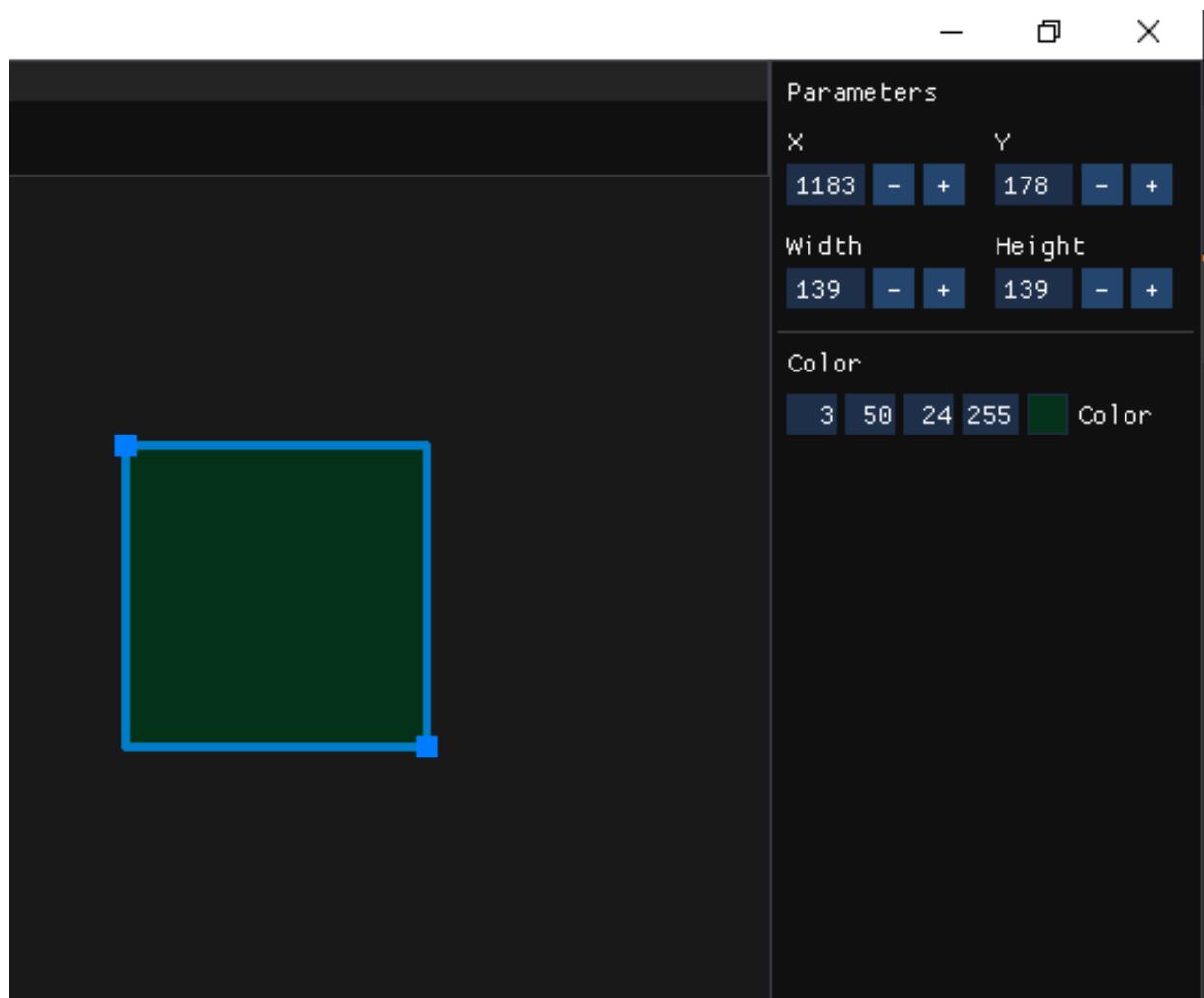


[Fig 82]:Journal papier 5 mai

OBJI s → → G è

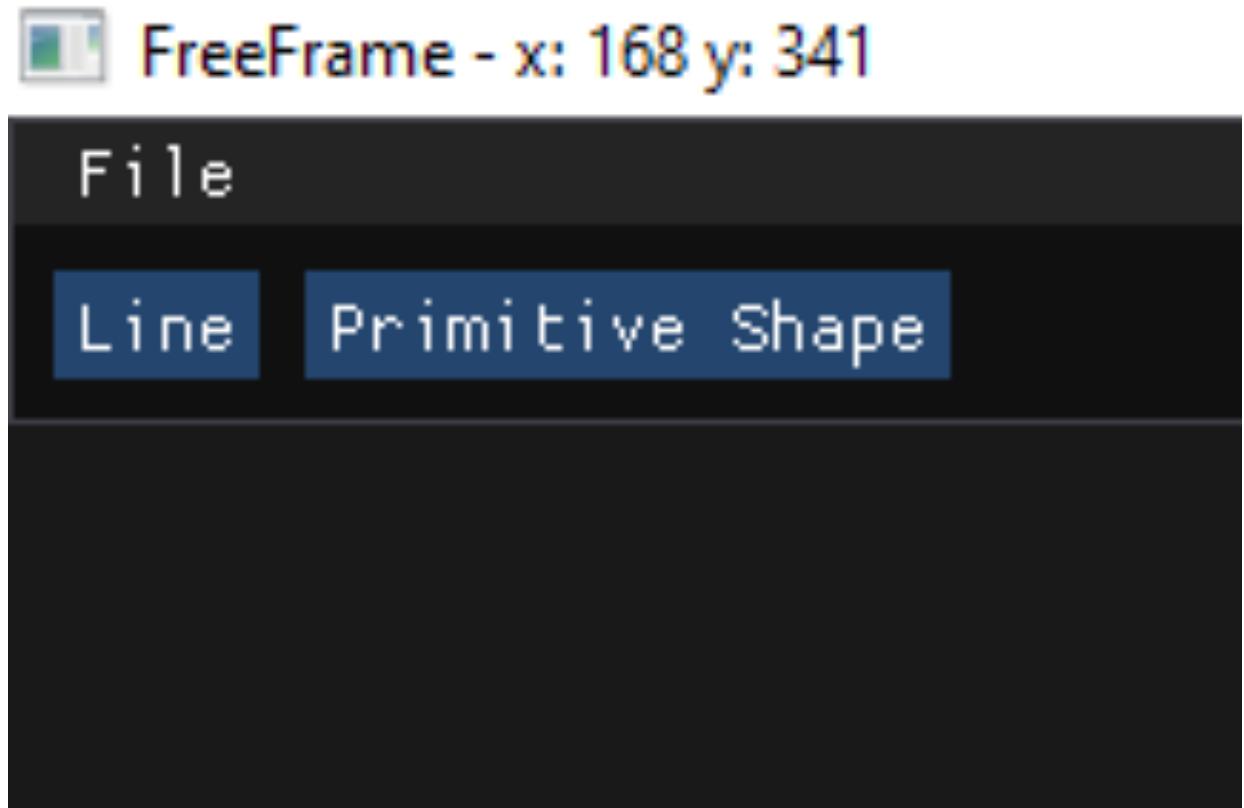
Je viens de terminer l'implémentation des rectangles arrondis il me faut ajouter dans l'interface la possibilité de modifier les valeurs d'arrondis et après je vais maintenant mettre à jour le poster.

J'ai modifier la disposition pour que l'ajout de nouveaux paramètres soit plus logique.



[Fig 83]:Démonstration disposition interface

J'ai également rajouté la position actuelle de la souris dans le titre de l'application.



[Fig 84]:Ajout position souris

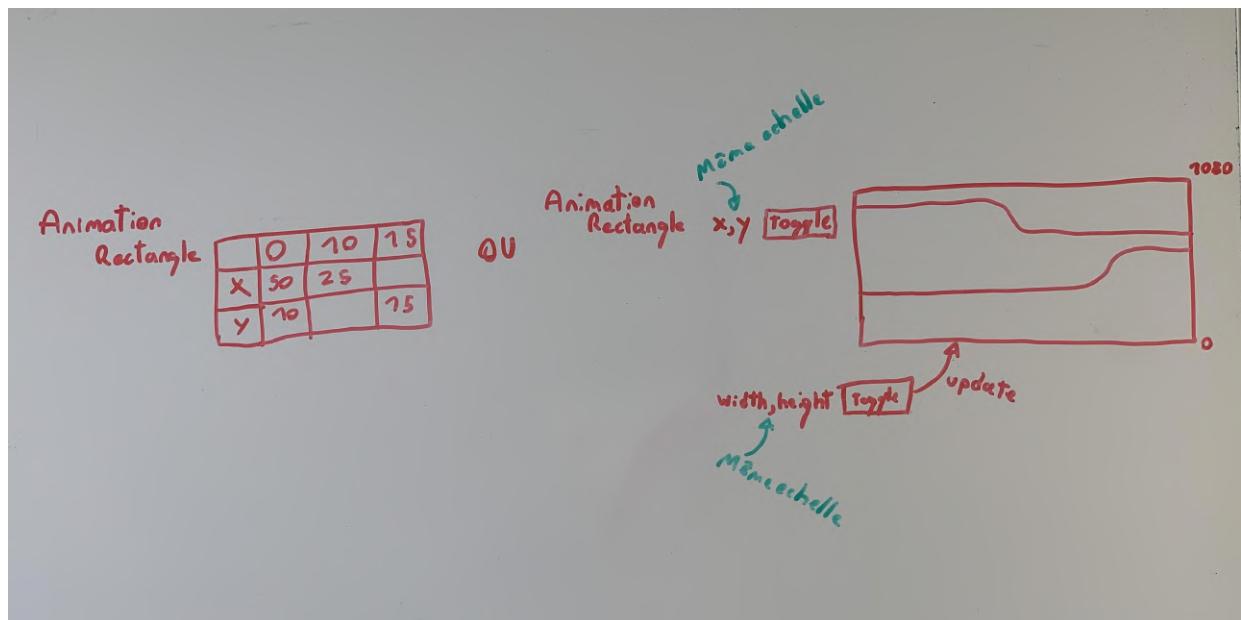
Je suis en train de me dire que ce serait bien de rajouter la sélection multiple, simplement avec Ctrl ou bien avec une sélection de zone avec la souris.

Il faudrait également que j'implémente le déplacement, par exemple si ont appuis sur la touche espace et qu'on clique avec la souris ça déplace. Ce serait bien de changer le curseur ?

J'ai décidé de mettre en commentaire les messages qui avertissent que l'on ne peut pas changer la taille de certains éléments. Je pense prendre du temps à afficher des messages de chaque contrainte appliquée plus tard.

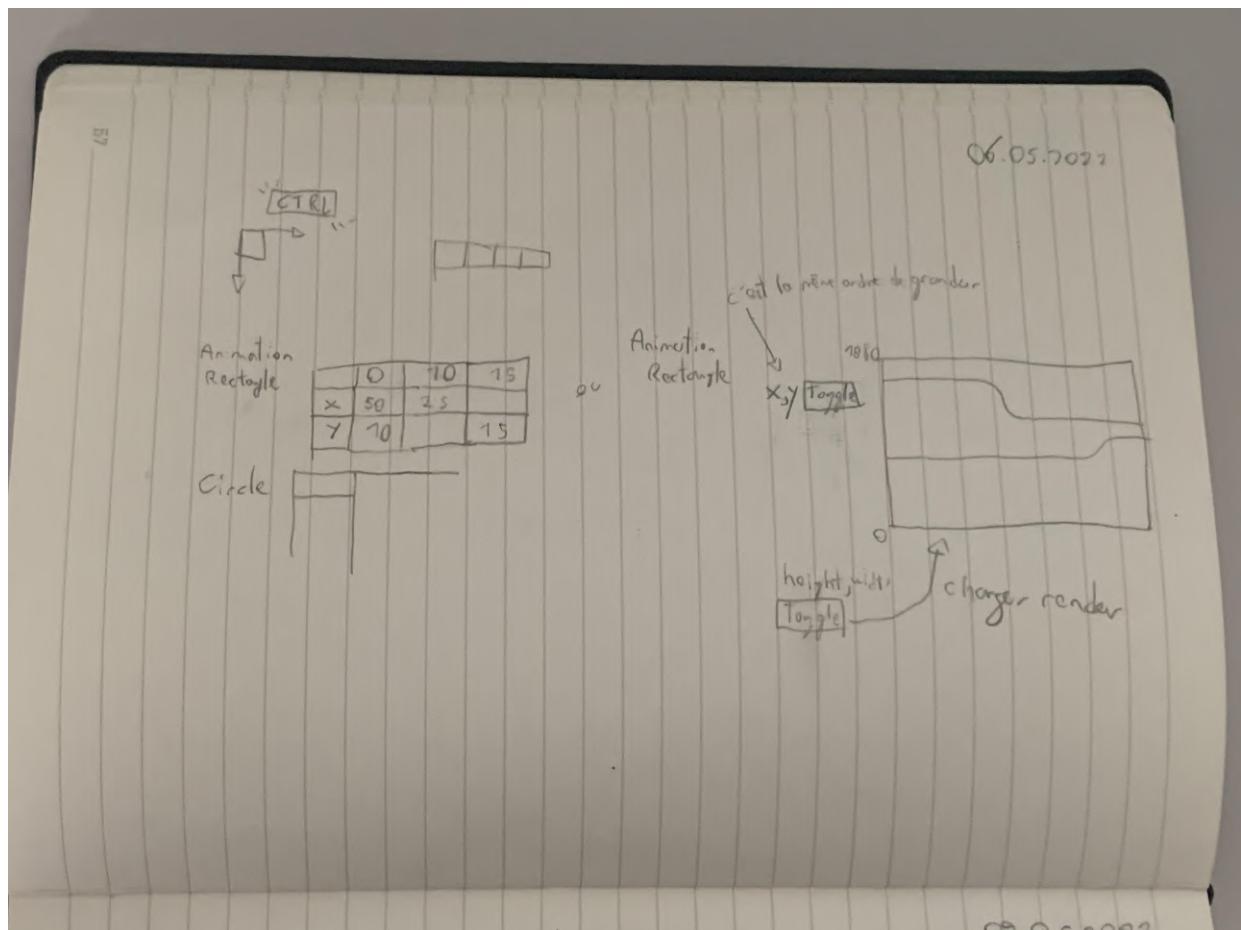
Je viens de penser qu'il serait intéressant d'appliquer une gestion de widget en simulant l'[API Docking](#).

Pour l'affichage des propriétés des éléments, j'ai deux choix :



[Fig 85]:Explication méthode pour panel d'animation

L'affichage avec un graphe est le beaucoup plus ergonomique, je vais donc d'abord implémenter la version avec le tableau.



[Fig 86]:Journal papier 6 mai

CBJ h → J è

Je vais commencer la journée par implémenter l'affichage de la timeline que j'avais imaginé la semaine dernière.

J'étais en train de me demander comment je vais mettre en production mon programme si j'ai des fichiers de shaders qui doivent toujours être présent dans la structure ? En cherchant je suis tombé sur une réponse <https://stackoverflow.com/questions/20443560/how-to-practically-ship-glsl-shaders-with-your-c-software> qui explique que c'est comme ça qu'il faut faire avec OpenGL. Cependant, ils disent qu'il est possible de déplacer à la compilation le texte dans une variable, comme ça il n'y a plus besoin de stocker les fichiers directement.

Je viens de penser qu'il serait bien de pouvoir changer la couleur du fond.

C9CA i → BA è

Je vais maintenant implémenter l'interpolation des éléments. Je vais simplement utiliser la [d'interpolation linéaire](#) disponible sur Wikipédia.

Pour l'instant j'ai un problème, l'interpolation ne marche pas lorsque que l'élément doit reculer en valeur pour avancer dans le temps.

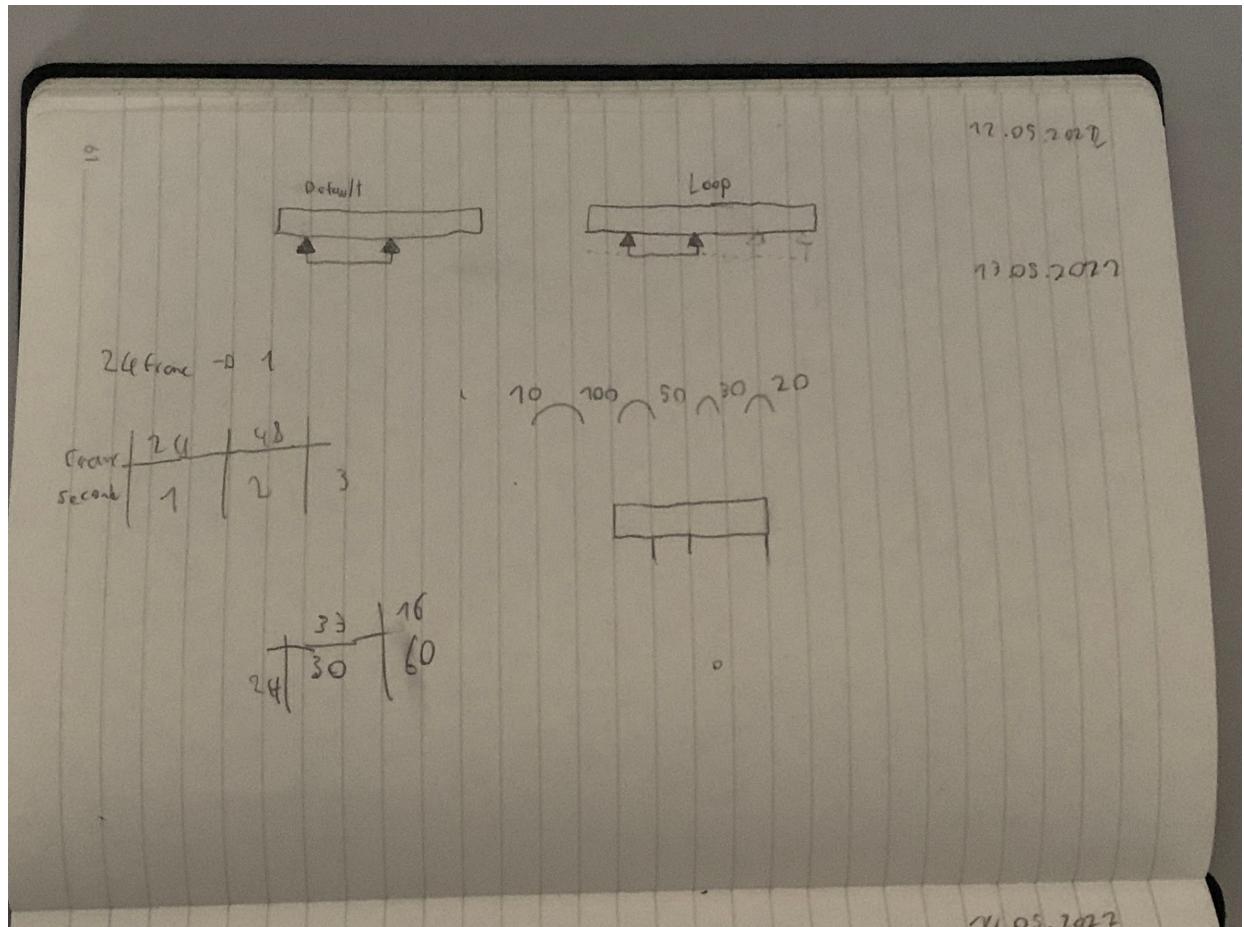
C9CB i ffi → BB è

J'ai fini d'implémenter la timeline. M. Schmid est venu pour me donner quelque conseils de structure de code.

Je viens de rajouter la possibilité d'enlever les keyframe. On peut également ajouter une option pour mettre en boucle la timeline.

C9CC f → BC è

Aujourd'hui, je vais régler le problème de la loop et implémenter des déplacements (avec la touche espace et drage and drop et zoom).



[Fig 87]:Journal papier 12 mai

OCDs → → BD è

Aujourd'hui, je vais commencer par implémenter l'exportation.

Comme écrit dans le cahier des charges, je dois en mp4 et svg et gif. Pour ce faire je vais utiliser [Emgu CV: OpenCV in .NET \(C#, VB, C++ and more\)](#) (wrapper .NET pour OpenCV). Il possède des méthodes simples pour créer des vidéos à partir d'images.

Pour exporter en mp4 je vais faire le rendu de chaque seconde dans la timeline et les combiner. Les méthodes d'Emgu permettent donc de gérer la vidéo de sortie.

Pour convertir en svg je vais simplement retranscrire les informations stockées dans le programme lors de la création de formes.

Pour convertir en gif je vais utiliser une [librairie](#) car les méthodes internet à .NET ne possède pas d'outil simple pour la création d'image gif.

M. Schmid est passé pour voir si tout ce passe bien.

Finalement la libairie utilisé [ne permet pas de mettre des delai courts entre les frames](#)

C9CE h → BG è

Je viens de penser qu'il faudrait que je passe la timeline en frame et que l'utilisateur puisse choisir directement le nombre de fps qu'il souhaite.

J'ai décidé d'enlever la loop et le reverse pour l'instant car ils seront intéressant si je les implémente au maximum. De plus que ce n'était pas écrit de faire ça dans le cahier des charges.

Je viens de refactor une énorme partie de la window. J'ai créé une classe timeline qui permet de gérer et d'afficher les éléments de la timeline.

Je dois implémenter l'anti aliasing et le bouton play.

C9CF i è → BH è

Je vais faire en sorte qu'il n'ait pas possible de redimensionner négativement un élément.

Je vais également ajouter l'option de pouvoir changer la couleur de fond de l'espace de travail.

Je dois demander à M. Schmid si je me lance dans la triangulation ou bien j'implémente l'élément `<animate>`. Également, je dois lui parler de la sécurité.

C90Gi ffi → BI è

Aujourd'hui, je vais régler le problème de déplacement des courbes de bézier de l'élément `<path>`.

- Régler problème export
 - Faire triangle
 - Sérialiser json/animate
 - Message erreur si problème fichier svg
 - Régler problème de position pour les paths
 - Implémenter Elliptical Arc Curve (A, a)
 - Implémenter (Z, z)
 - Problème rotation angles
 - Vérifier mettre en place sécurité pour la compatibilité Linux/Windows
-

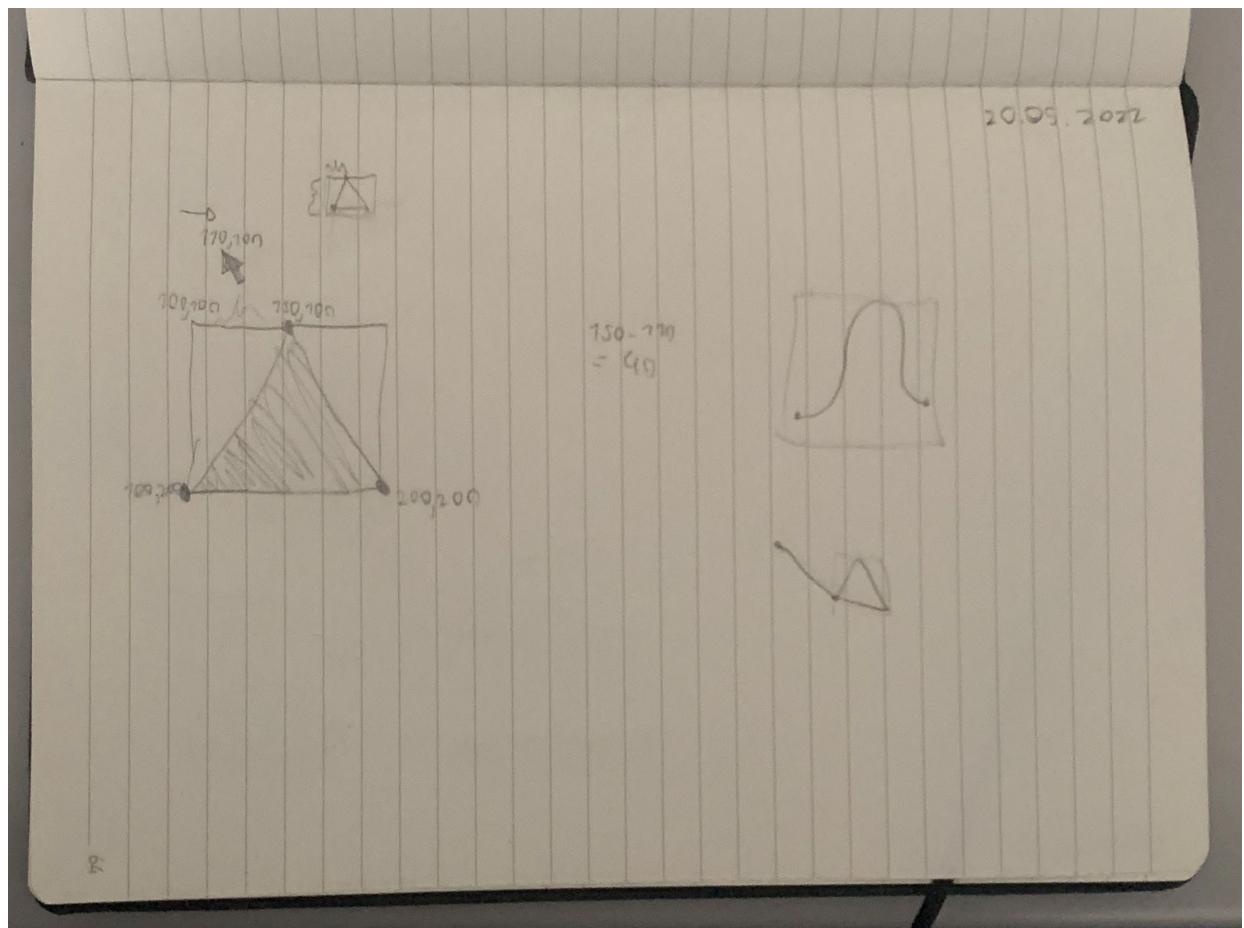
C90Hf → BJ è

Aujourd'hui, j'ai avancé de la doc, j'ai ajouté des schémas que j'ai réalisés sur Figma. Je viens de résoudre le problème de rotation à l'export.

C91 s → → CA è

Pour les polygons j'ai décidé que les points sont stockés en relatif par rapport aux variables X et Y. Comme ça pour le déplacement, j'ai simplement à modifier X et Y et le reste se modifie tout seul.

Je viens de me rendre compte qu'il faut que je prenne en compte la rotation lorsque j'exporte un SVG.



[Fig 88]:Journal papier 20 mai

O9CJ h → CD è

Aujourd'hui j'ai réussi à finir l'implémentation les triangles en utilisant `<polygon>`. Je commence également par essayer de sérialiser l'espace de travail au format SVG.

C9DA i è → CE è

J'ai continué à travailler sur la sérialisation des fichiers SVG en utilisant `transform` et `animateTransform`.

C9DB i ffi → CF è

Aujourd'hui j'ai travaillé sur la documentation et j'ai fait les mises à jour que M. Schmid m'avait demandé de faire lors de la réunion précédente.

C9DC s → → CH è

J'ai avancé la documentation et fait des schémas. J'ai également résolu le problème d'export (GIF, MP4) qui mettait à l'envers.

C9DD h → DA è

Aujourd'hui j'ai mis à jour le journal de bord à partir des notes que j'avais prises sur papier. J'ai également rajouté les tests unitaires, une conclusion partielle et les problèmes rencontrés sur la documentation.

C9DE i è → DB è

Ce matin, M. Garcia est passé pour voir si tout se passait bien. J'ai mis en place l'environnement de MkDocs pour me permettre par la suite de rapidement convertir mon document markdown en pdf. J'ai ensuite continué la documentation.

C9DF i ffi → B

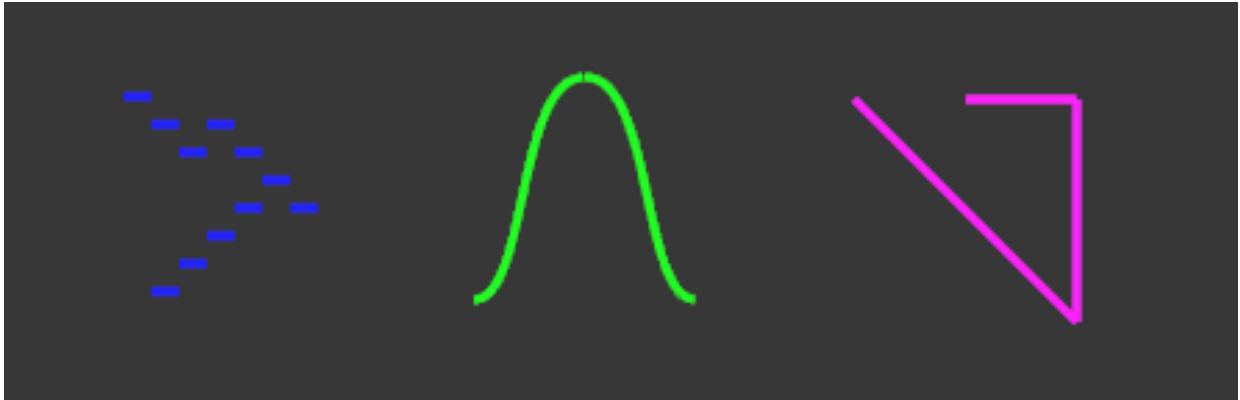
Comme M. Schmid m'avait dit que j'avais pris du retard sur la documentation, je continue à l'avancée, je pense avoir le temps de régler le reste des problèmes la semaine prochaine une fois la documentation terminée.

M. Schmid est passé pour me conseiller d'expliquer en détail comment je lis et j'implémente OpenGL et d'éviter de juste survoler.

Il m'a également dit qu'il ne fallait pas que je mette trop de code dans ma documentation.

O9Gf → C

Aujourd'hui j'ai réglé le problème d'affichage de `<path>`. (j'avais fait de mauvais calculs pour la position).



[Fig 89]: Démonstration fonctionnement de path

Je suis en train de travailler sur la sérialisation/désérialisation et je rencontre beaucoup de problèmes. J'ai décidé d'utiliser NewtonsoftJSON.

Le problème que j'ai que Newtonsoft appelle les constructeurs au moment de désérialiser, du coup ça modifie tous les id de mes shapes.

Ok le problème était que mes propriétés étaient en `protected` et par conséquent Newtonsoft appelle automatiquement le constructeur pour essayer de le remplir par ça.

M. Schmid est passé, il m'a validé que j'allais maintenant intégrer un explorateur de fichier pour la sauvegarde et régler le problème de rotation.

O9Hs → → D

Aujourd'hui, je voulais m'attaquer au problème de gestion des inputs pour ImGui. J'ai une erreur qui fait que mes entrées du clavier ne sont pas lus par ImGui.

Après beaucoup de recherche sur différents forums, je décide de [poster mon problème](#). J'espère recevoir une réponse rapide, car il me reste seulement une semaine de travail.

CDI i è → H

M. Garcia est passé dans la classe pour nous briefer sur comment va se dérouler la fin du diplôme.

Après ses explications, je commence à travailler sur la documentation.

J'ai également factorisé du code.

CDJ i ffi → I

J'ai commencé la journée par encore factorisé le code. Je viens de régler le problème de l'importateur de fichiers qui faisait que ça ne mettait pas à jour le filtres des extensions.

Questions pour M. Schmid :

- Entête des fichiers
- Manuel utilisateur

M. Schmid est passé, je lui est posé mes questions et il m'a fait un retour sur la documentation.

CEA f → J

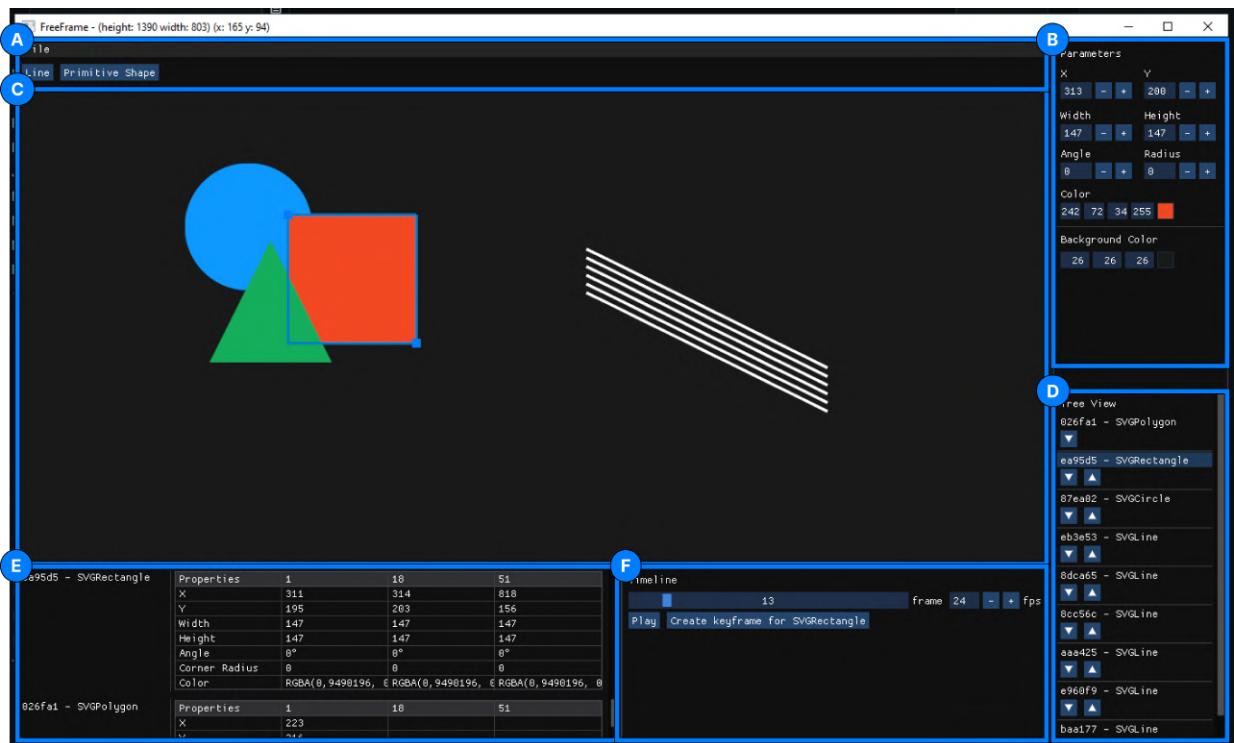
Je continue à avancer la documentation, car il me reste plus qu'un jour et j'ai terminé de faire le manuel utilisateur.

C9EB s → → BA

J'ai préparé le rendu et rendu tous les fichiers.

oe

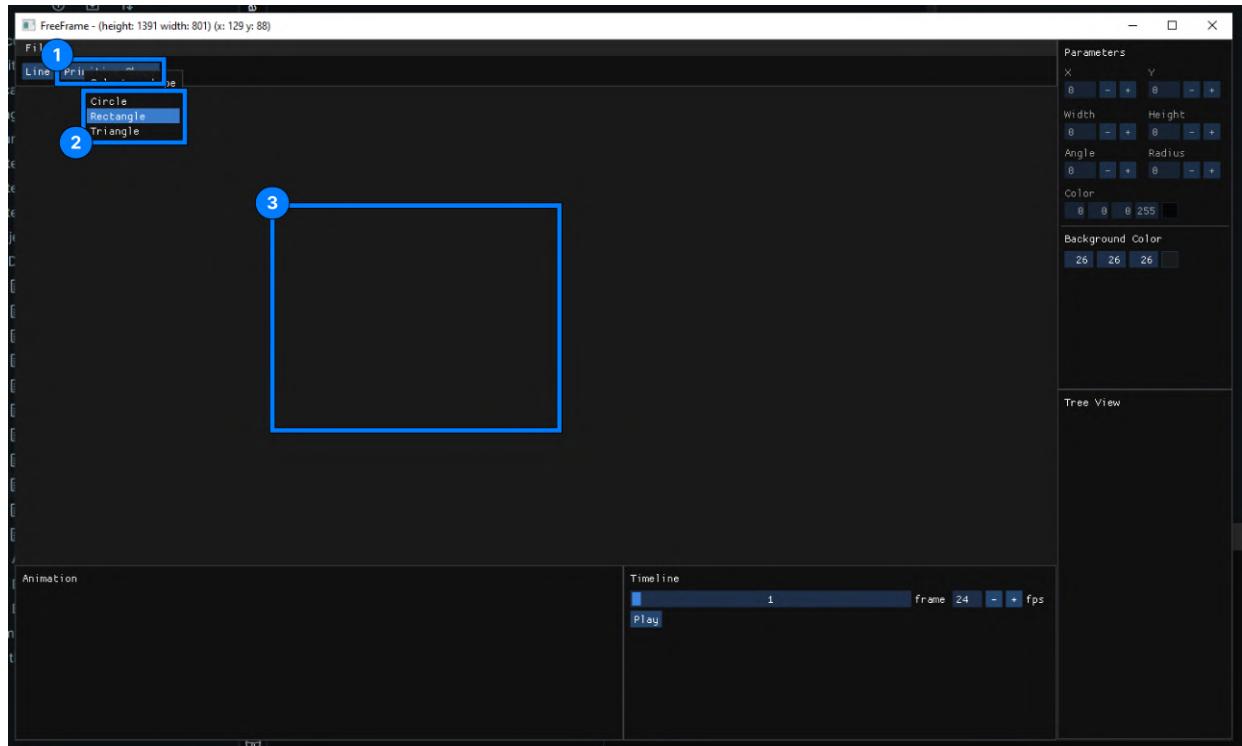
D9B Z ffi → èffi



[Fig 90]:Description de l'interface

- A -> Barre de navigation
- B Espace des paramètres
- C Espace de travail
- D Hiérarchie des formes
- E -> Informations sur les animations
- F -> Gestion de la timeline

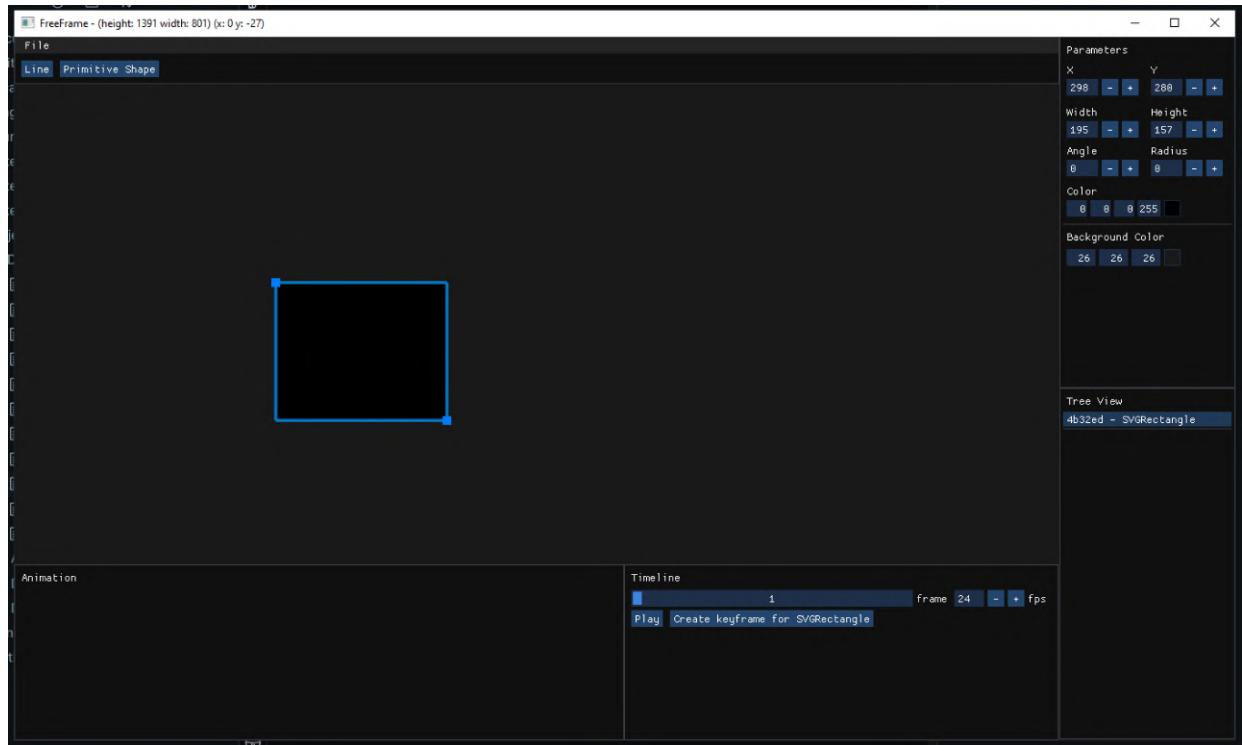
D9C V



[Fig 91]:Créer une forme 1

Pour créer une forme :

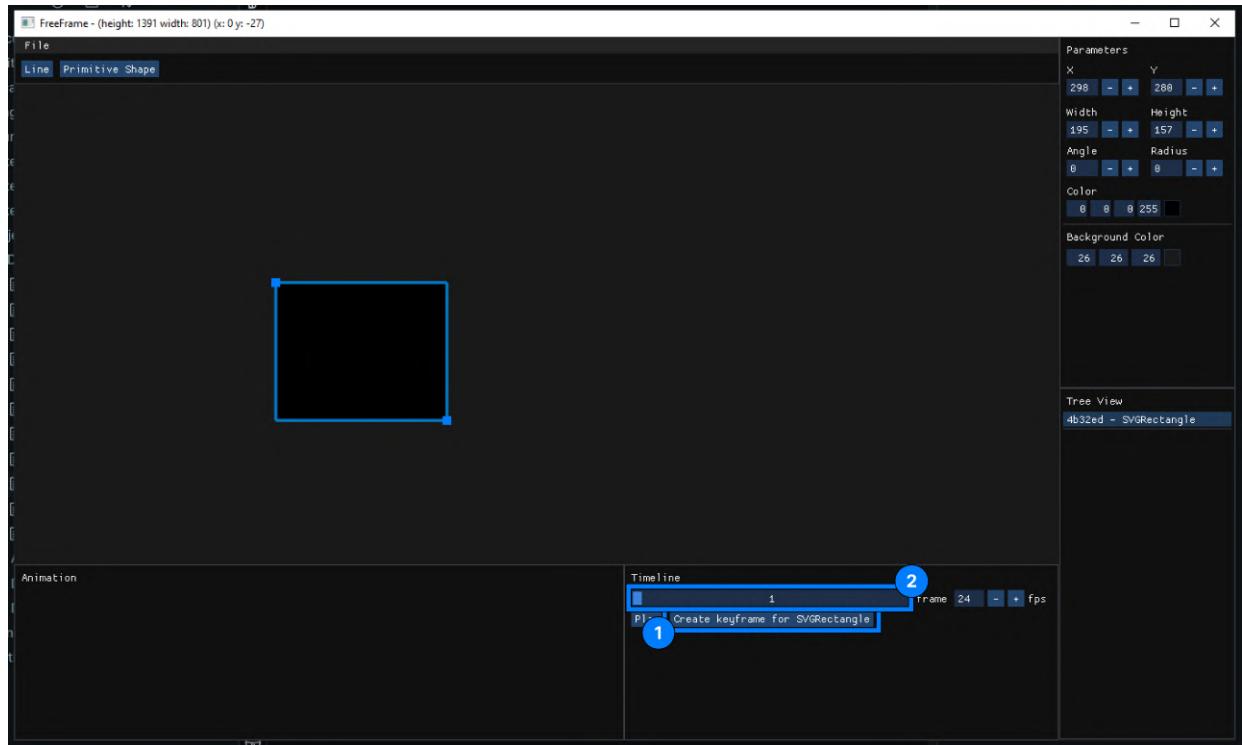
1. Cliquer sur le bouton *Primitive Shape*
2. Choisir la forme *Rectangle*
3. Cliquer sur l'espace de travail, puis en restant appuyé, glisser la souris



[Fig 92]:Créer une forme résultat

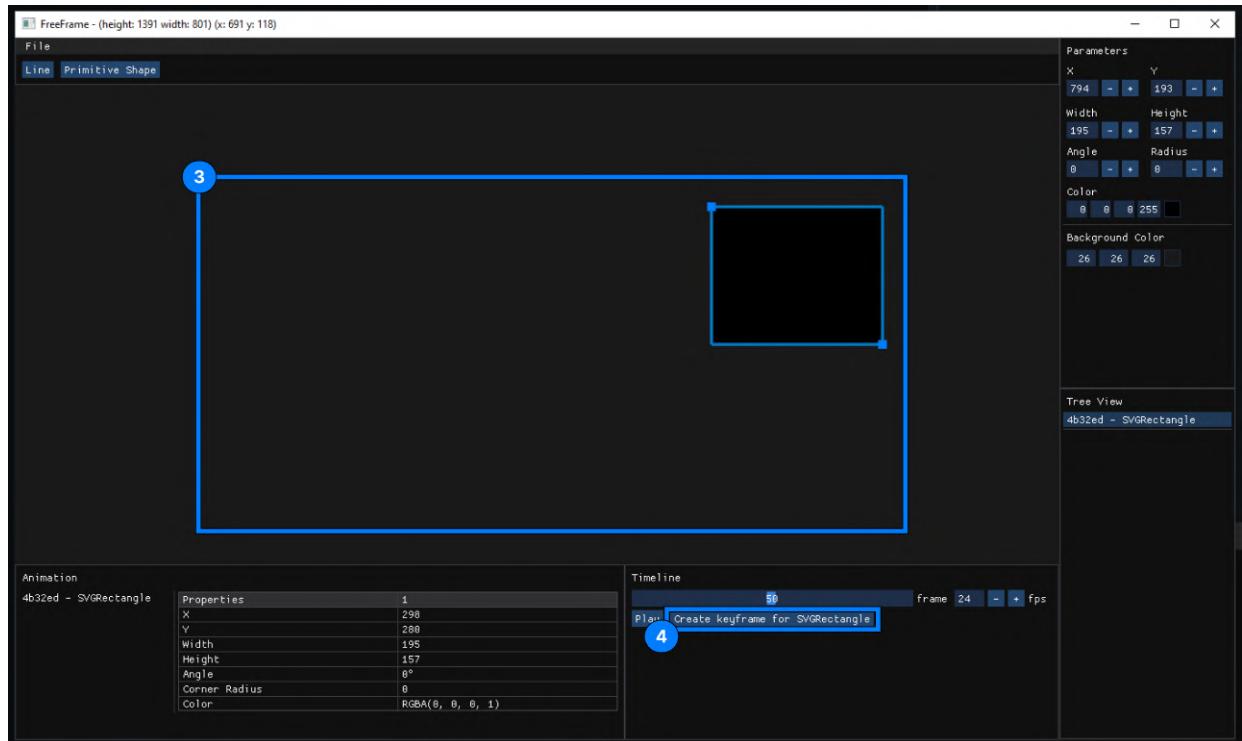
D9S

Premièrement, il faut vérifier qu'une forme est bien sélectionnée.



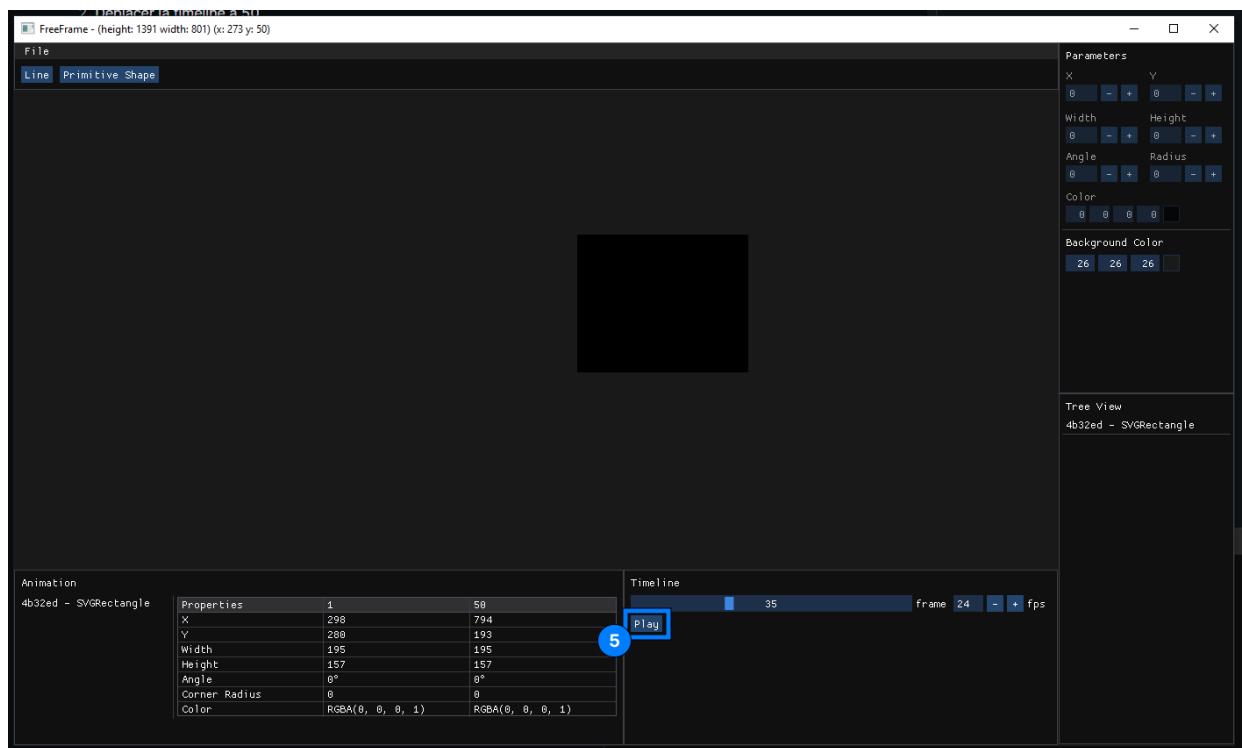
[Fig 93]:Animer une forme 1

1. Cliquer sur le bouton *Create keyframe for SVGRectangle* pour sauvegarder l'état actuel de la forme
2. Déplacer la timeline à 50



[Fig 94]:Animer une forme 2

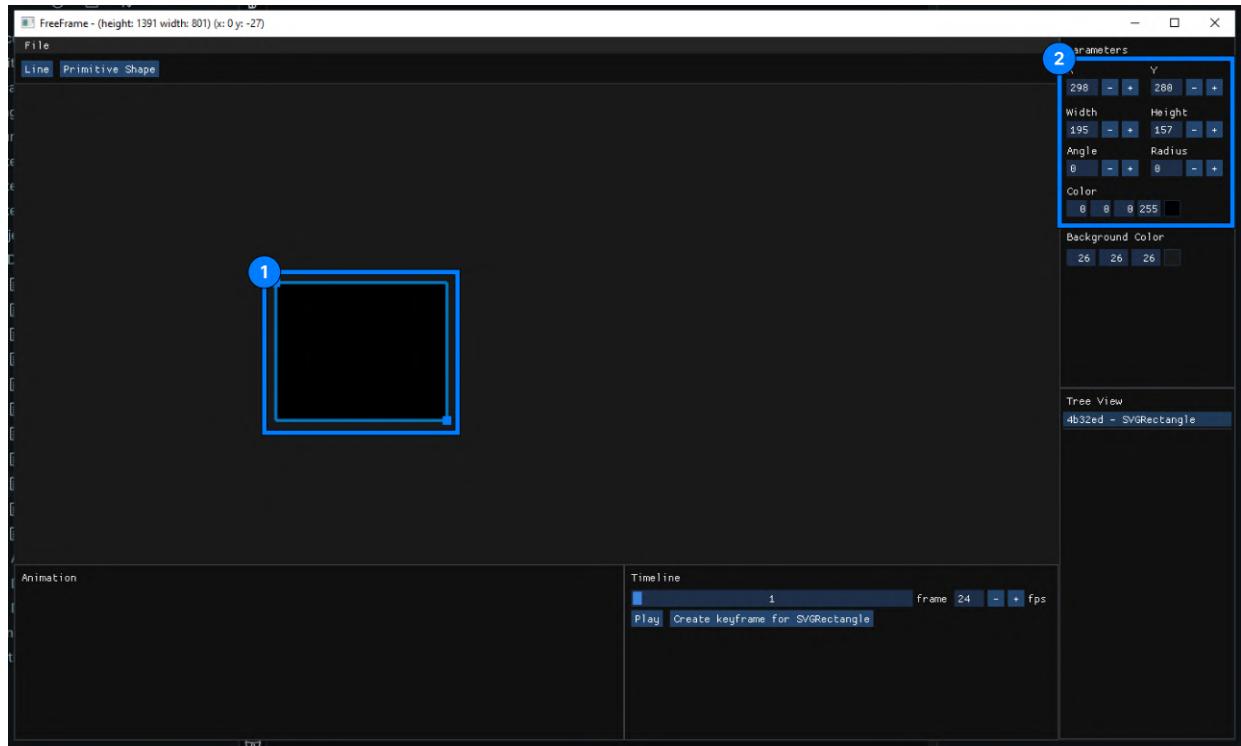
1. Changer l'état de la forme, par exemple en la déplaçant
2. Cliquer sur le bouton *Create keyframe for SVGRectangle* pour sauvegarder le nouvel état de la forme



[Fig 95]:Animer une forme résultat

1. Vous pouvez maintenant cliquer sur le bouton *Play* et voir la forme se déplacer de gauche à droite

DE i →

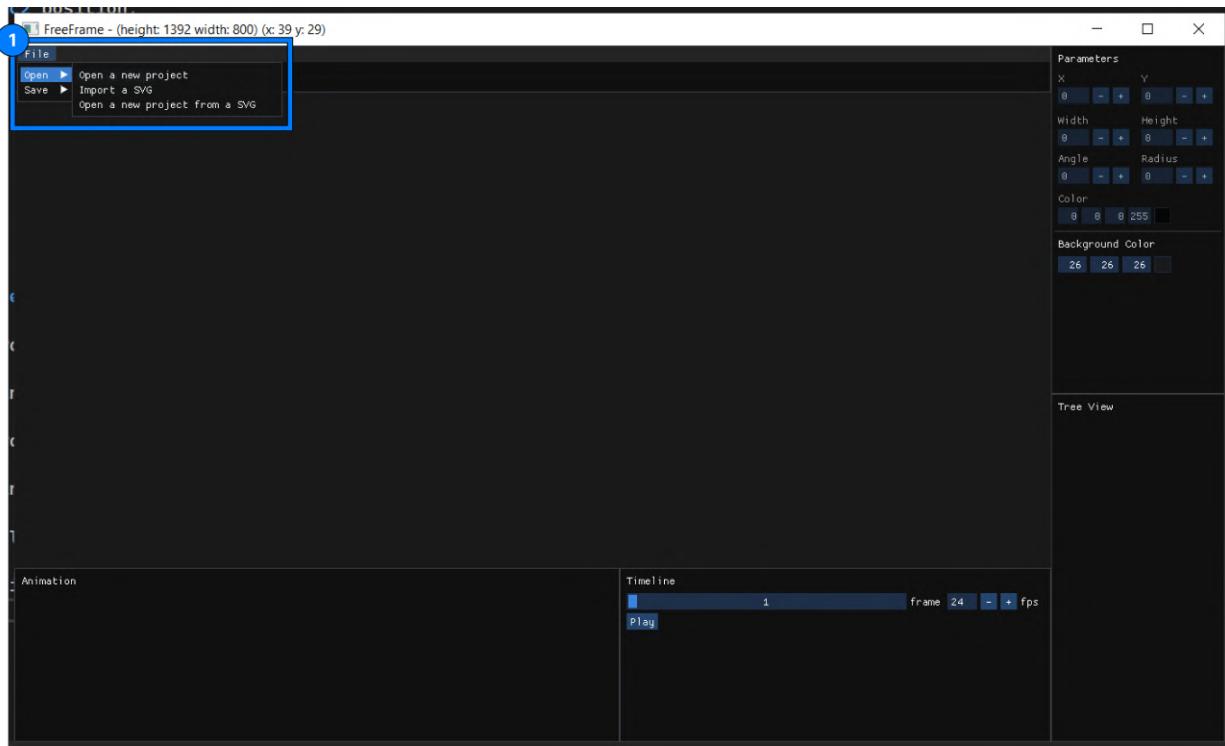


[Fig 96]:Modifier une forme 1

Pour modifier une forme :

1. Vérifier qu'une forme est bien sélectionnée
2. Modifier les paramètres de la forme via l'interface

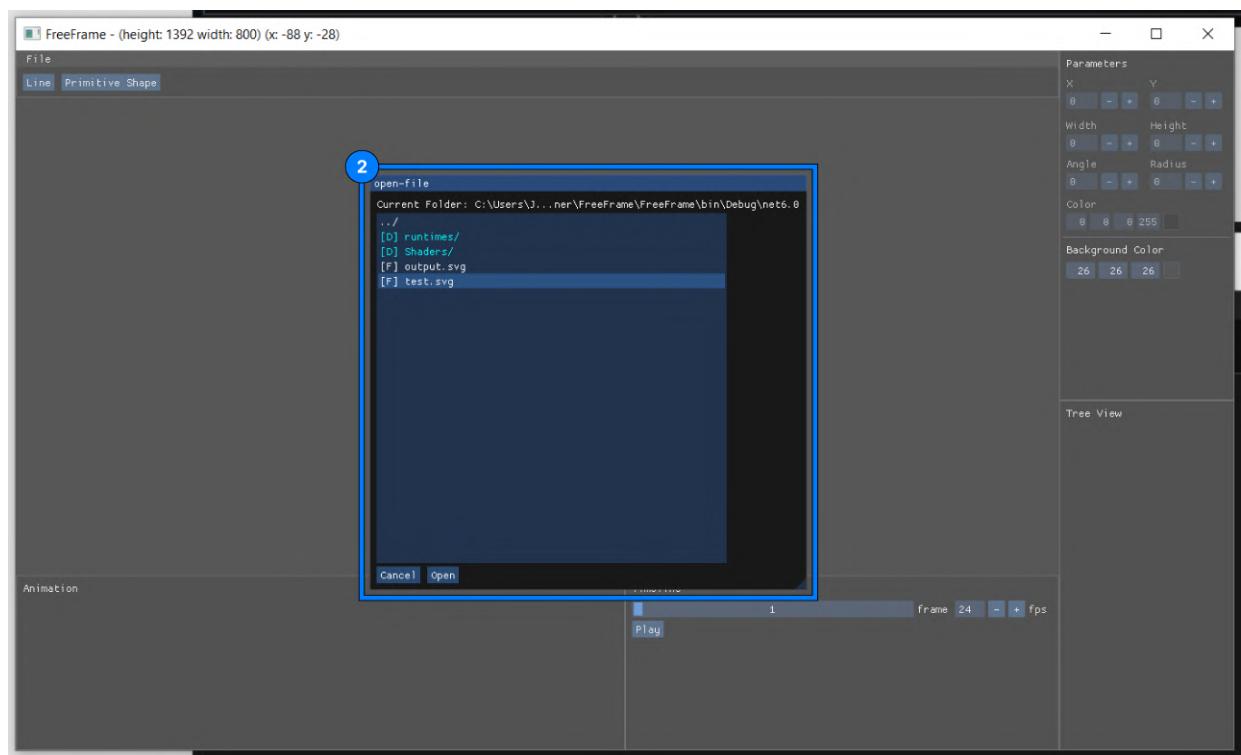
Définition



[Fig 97]: Importer un fichier 1

Pour importer un fichier :

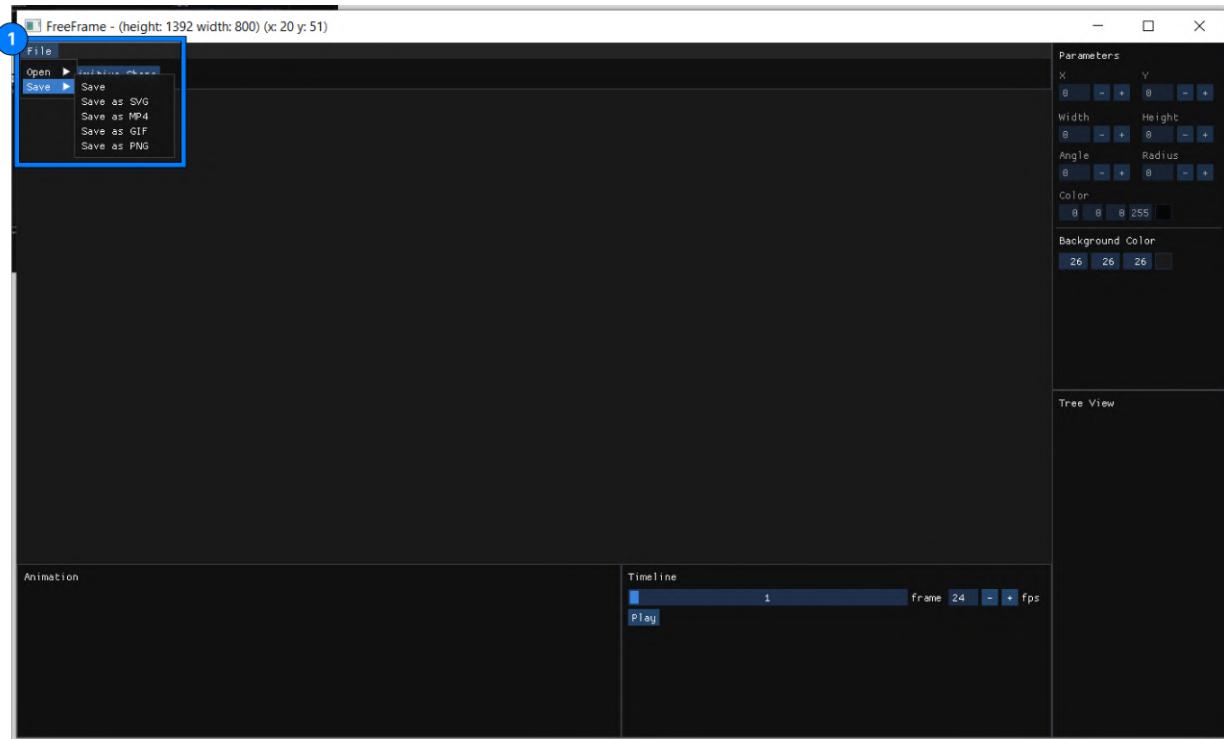
1. Cliquer sur *File* puis *Open* ensuite choisissez parmi les trois options d'import possibles



[Fig 98]: Importer un fichier 2

1. Choisir le fichier à importer puis appuyer sur le bouton *Open*

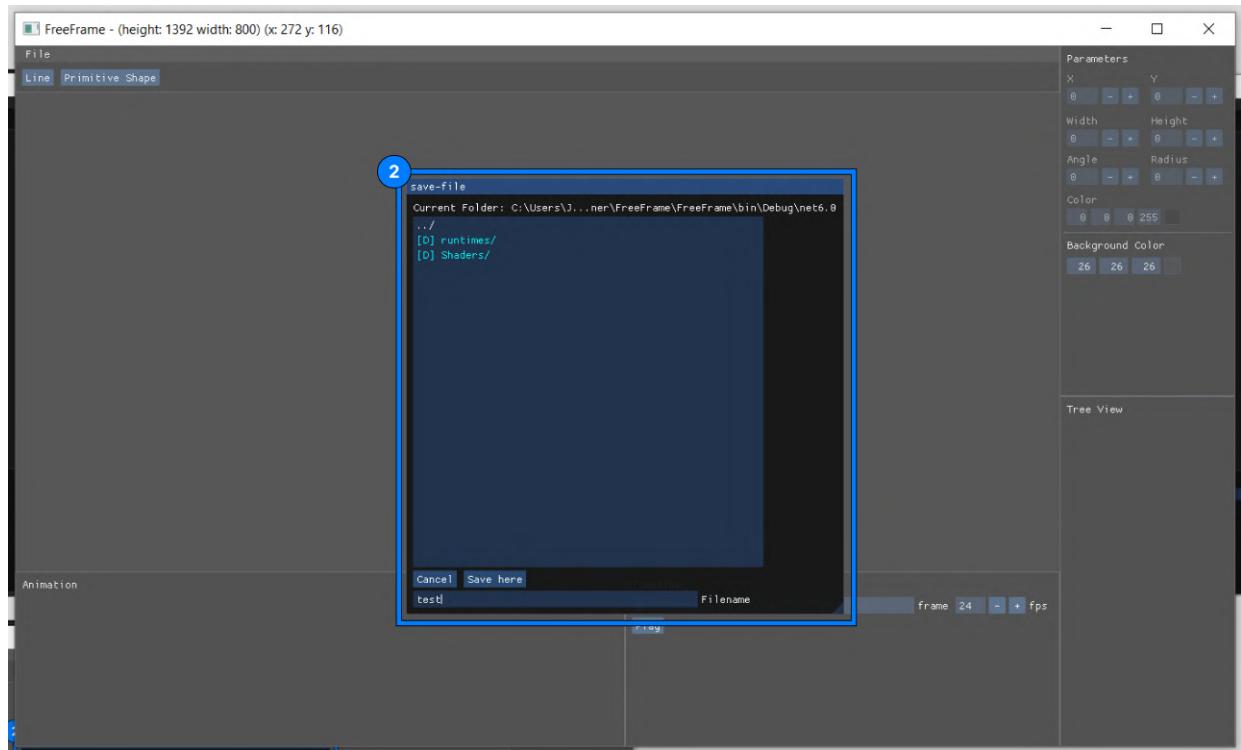
D9Ga



[Fig 99]:Exporter un fichier 1

Pour exporter :

1. Cliquer sur *File* puis *Save* ensuite choisissez parmi les cinq options d'export possibles



[Fig 100]: Exporter un fichier 2

1. Choisir l'emplacement de sauvegarde du fichier ainsi que son nom, puis cliquer sur le bouton *Save here*

D9H qèff →

[\[Fig 1\]](#): Maquette de l'application

[\[Fig 2\]](#): Affiches

[\[Fig 3\]](#): Maquette de l'interface de l'application

[\[Fig 4\]](#): Structure interne de l'application

[\[Fig 5\]](#): Outil Cavalry. Source : cavalry.scenegroup.co

[\[Fig 6\]](#): Outil After Effects. Source : adobe.com

[\[Fig 7\]](#): Logo OpenGL. Source : khronos.org

[\[Fig 8\]](#): Workflow GitHub Projects

[\[Fig 9\]](#): Espace de travail Zotero

[\[Fig 10\]](#): Diagramme1

[Fig 11]: Diagramme1

[Fig 12]: Différence Triangle et Square

[Fig 13]: Triangle Lists. Source : Game engine architecture - Jason Gregory

[Fig 14]: Indexed Triangle Lists. Source : Game engine architecture - Jason Gregory

[Fig 15]: Pipeline de rendu dans OpenGL. Source : learnopengl.com

[Fig 16]: Trois ronds superposés

[Fig 17]: Représentation du système solaire

[Fig 18]: Quatre rectangles colorés superposés, ordonné

[Fig 19]: Quatre rectangles colorés superposés, désordonné

[Fig 20]: Trois ronds superposés, un sélectionné

[Fig 21]: Trois ronds superposés, un décalé

[Fig 22]: Déplacement d'un élément

[Fig 23]: Trois ronds superposés, un grand

[Fig 24]: Redimension d'élément

[Fig 25]: Deux triangles, un avec une rotation

[Fig 26]: Un rectangle arrondis

[Fig 27]: Trois ronds superposés, un rouge

[Fig 28]: Création d'un rectangle, étape une

[Fig 29]: Création d'un rectangle, étape deux

[Fig 30]: Création d'un rectangle, étape trois

[Fig 31]: Outil de debug

[Fig 32]: Architecture de l'application

[Fig 33]: Diagramme de classe

[Fig 34]: Courbe de Bézier cubique

[Fig 35]: Courbe de Bézier cubique dépendante

- [Fig 36]: Courbe de Bézier quadratique
- [Fig 37]: Courbe de Bézier quadratique dépendante
- [Fig 38]: Architecture d'import d'un fichier
- [Fig 39]: Architecture des classes de IDrawable
- [Fig 40]: Architecture d'appel d'OpenGL
- [Fig 41]: Architecture de Shape
- [Fig 42]: Courbe de Bézier cubique précisions
- [Fig 43]: Rotation origine comparaison
- [Fig 44]: Conversion model view to NDC
- [Fig 45]: Schéma, problème d'export PNG
- [Fig 46]: Architecture sélection d'une forme
- [Fig 47]: Formes et leurs points de sélections
- [Fig 48]: Points de sélections choisis lors d'un clic
- [Fig 49]: Schéma calcul d'hypoténuse
- [Fig 50]: Architecture modification d'une forme
- [Fig 51]: Schéma interpolation linéaire
- [Fig 52]: Interface de l'application
- [Fig 53]: Hello world dans ImGui
- [Fig 54]: Planification prévisionnelle
- [Fig 55]: Planification effective
- [Fig 56]: Journal papier 4 avril
- [Fig 57]: Journal papier 4 avril 2
- [Fig 58]: Journal papier 4 avril 3
- [Fig 59]: Journal papier 6 avril
- [Fig 60]: Journal papier 8 avril

- [Fig 61]: FilePicker.cs avant
- [Fig 62]: FilePicker.cs Après
- [Fig 63]: Journal papier 11 avril
- [Fig 64]: Journal papier 11 avril 2
- [Fig 65]: Résultat actuel de l'application
- [Fig 66]: Journal papier 13 avril 1
- [Fig 67]: Journal papier 13 avril 2
- [Fig 68]: Ligne sur l'espace
- [Fig 69]: Polygone qui se passe dessus
- [Fig 70]: Deux étape pour convertir
- [Fig 71]: Journal papier 25 avril
- [Fig 72]: Journal papier 25 avril 2
- [Fig 73]: Schéma sur tableau blanc de mon problème
- [Fig 74]: Démonstration fonctionnement de l'application
- [Fig 75]: Démonstration fonctionnement système decouleurs
- [Fig 76]: Démonstration fonctionnement de path
- [Fig 77]: Outil de debug en création
- [Fig 78]: Journal papier 3 mai
- [Fig 79]: Problème de path
- [Fig 80]: Rond compatibles
- [Fig 81]: Trois rond superposés coupé
- [Fig 82]: Journal papier 5 mai
- [Fig 83]: Démonstration disposition interface
- [Fig 84]: Ajout position souris
- [Fig 85]: Explication méthode pour panel d'animation

[Fig 86]: Journal papier 6 mai

[Fig 87]: Journal papier 12 mai

[Fig 88]: Journal papier 20 mai

[Fig 89]: Démonstration fonctionnement de path

[Fig 90]: Description de l'interface

[Fig 91]: Créer une forme 1

[Fig 92]: Créer une forme résultat

[Fig 93]: Animer une forme 1

[Fig 94]: Animer une forme 2

[Fig 95]: Animer une forme résultat

[Fig 96]: Modifier une forme 1

[Fig 97]: Importer un fichier 1

[Fig 98]: Importer un fichier 2

[Fig 99]: Exporter un fichier 1

[Fig 100]: Exporter un fichier 2

Code source - FreeFrame

Jeremy Meissner
Travail de diplôme 2022
École d'informatique (CFPT-I)

10 Juin 2022

Table des matières

1 FreeFrame	2
1.1 C#	2
1.1.1 Program.cs	2
1.1.2 Window.cs	2
1.1.3 Importer.cs	24
1.1.4 Timeline.cs	26
1.1.5 Helper.cs	30
1.1.6 IDrawable.cs	31
1.1.7 Shape.cs	32
1.1.8 Selector.cs	34
1.1.9 Renderer.cs	36
1.1.10 SVGLine.cs	39
1.1.11 SVGCircle.cs	41
1.1.12 SVGRectangle.cs	43
1.1.13 SVGPolygon.cs	44
1.1.14 SVGPath.cs	47
1.1.15 DrawAttribute.cs	51
1.1.16 Shader.cs	64
1.1.17 FilePicker.cs	65
1.1.18 FileSaver.cs	68
1.1.19 ImGuiUtil.cs	70
1.1.20 ImGuiController.cs	72
1.1.21 ImGuiShader.cs	78
1.1.22 ImGuiTexture.cs	80
1.2 Shaders	84
1.2.1 shader.vert	84
1.2.2 shader.frag	84
1.2.3 circle.frag	84
1.2.4 rectangle.frag	85

Chapitre 1

FreeFrame

1.1 C#

1.1.1 Program.cs

```
1 ﻿using OpenTK.Mathematics;
2  using OpenTK.Windowing.Desktop;
3
4  namespace FreeFrame
5  {
6      class Program
7      {
8          static void Main()
9          {
10              NativeWindowSettings nativeWindowSettings = new()
11              {
12                  Size = new Vector2i(800, 600),
13                  Title = "FreeFrame",
14                  NumberOfSamples = 8, // MSAA
15              };
16              using Window window = new(GameWindowSettings.Default, ←
17                  nativeWindowSettings); // Create window context (GLFW, ←
18                  OpenGL)
19
20              window.Run();
21
22      }
23  }
```

Listing 1.1 – ./source/csharp/Program.cs

1.1.2 Window.cs

```
1  using AnimatedGif;
2  using Emgu.CV;
3  using FreeFrame.Components;
4  using FreeFrame.Components.Shapes;
5  using FreeFrame.Lib.FileExplorer;
6  using FreeFrame.Lib.ImGuiTools;
7  using ImGuiNET;
8  using Newtonsoft.Json;
9  using OpenTK.Graphics.OpenGL4;
10 using OpenTK.Mathematics;
11 using OpenTK.Windowing.Common;
12 using OpenTK.Windowing.Desktop;
13 using OpenTK.Windowing.GraphicsLibraryFramework;
14 using System.Drawing;
15 using System.Drawing.Imaging;
16 using static FreeFrame.Selector;
17
```

```

18 namespace FreeFrame
19 {
20     public class Window : GameWindow
21     {
22         const int DEFAULT_IO_INT = 0;
23         struct Workspace
24         {
25             public List<Shape> Shapes { get; set; }
26             public SortedDictionary<int, List<Shape>> SortedTimeline { ←
27                 get; set; }
28             public System.Numerics.Vector3 BgColor { get; set; }
29         }
30         enum UserMode
31         {
32             Idle,
33             Edit,
34             Create
35         }
36         enum ImportMode
37         {
38             Workspace,
39             Add,
40             Override
41         }
42         enum ExportMode
43         {
44             Workspace,
45             GIF,
46             MP4,
47             PNG,
48             SVG
49         }
50         enum CreateMode
51         {
52             Line,
53             Rectangle,
54             Circle,
55             Triangle
56         }
57         int _ioAngle;
58         int _ioCornerRadius;
59         int _ioX;
60         int _ioY;
61         int _ioWidth;
62         int _ioHeight;
63         int _ioTimeline;
64         int _ioFps;
65         System.Numerics.Vector4 _ioColor;
66
67         System.Numerics.Vector3 _ioBgColor;
68         bool _dialogFilePicker = false;
69         bool _dialogCompatibility = false;
70         bool _dialogFileSaver = false;
71         bool _dialogError = false;
72
73         Vector2i _mouseOriginalState;
74
75         Timeline _timeline;
76
77         SelectorType _selectorType = SelectorType.None;
78
79         Selector _selector;
80         Shape? _selectedShape;
81         Shape? _selectedShapeBefore;
82
83         UserMode _userMode;
84         CreateMode _createMode;
85         ImportMode _importMode;
86         ExportMode _exportMode;
87
88         ImGuiController _ImGuiController;

```

```

89     List<Shape> _shapes;
90
91     public List<Shape> Shapes { get => _shapes; set => _shapes = value; }
92
93     public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWindowSettings) : base(gameWindowSettings, nativeWindowSettings) { }
94
95     protected override void OnLoad()
96     {
97         base.OnLoad();
98
99         //Helper.EnableDebugMode();
100
101        _userManager = UserManager.Idle;
102        _createMode = CreateMode.Rectangle;
103
104        Shapes = new List<Shape>();
105        _selector = new Selector();
106        _mouseOriginalState = new Vector2i(0, 0);
107        _ImGuiController = new ImGuiController(ClientSize.X, ClientSize.Y);
108        _timeline = new Timeline();
109
110        // Input/Output
111        _ioAngle = DEFAULT_IO_INT;
112        _ioCornerRadius = DEFAULT_IO_INT;
113        _ioX = DEFAULT_IO_INT;
114        _ioY = DEFAULT_IO_INT;
115        _ioWidth = DEFAULT_IO_INT;
116        _ioHeight = DEFAULT_IO_INT;
117        _ioTimeline = DEFAULT_IO_INT;
118        _ioFps = Timeline.DEFAULT_FPS;
119        _ioColor = new System.Numerics.Vector4(0f, 0f, 0f, 1f);
120        _ioBgColor = new System.Numerics.Vector3(0.1f, 0.1f, 0.1f);
121        GL.ClearColor(_ioBgColor.X, _ioBgColor.Y, _ioBgColor.Z, 1.0f);
122
123        GL.Enable(EnableCap.Multisample);
124    }
125
126    protected override void OnResize(EventArgs e)
127    {
128        base.OnResize(e);
129
130        GL.Viewport(0, 0, ClientSize.X, ClientSize.Y);
131
132        _ImGuiController.WindowResized(ClientSize.X, ClientSize.Y);
133    }
134    /// <summary>
135    /// Triggered at a fixed interval. (Logic, etc.)
136    /// </summary>
137    /// <param name="e"></param>
138    protected override void OnUpdateFrame(FrameEventArgs e)
139    {
140        base.OnUpdateFrame(e);
141
142        Title = String.Format("FreeFrame - (height:{0} width:{1}) - ({2},{3})", ClientSize.X, ClientSize.Y, MouseState.X, MouseState.Y);
143
144        if (_userManager == UserManager.Edit && _selectorType == SelectorType.Move)
145        {
146            float x = MouseState.X, y = MouseState.Y;
147            if (KeyboardState.IsKeyDown(Keys.LeftShift))
148            {
149                if (Math.Abs(_mouseOriginalState.X - x) > Math.Abs(_mouseOriginalState.Y - y))
150                    y = _mouseOriginalState.Y;
151                else
152                    x = _mouseOriginalState.X;

```

```

153         }
154         Title += String.Format("(deltaX:{0} deltaY:{1})", ←
155             x - _mouseOriginalState.X, y - _mouseOriginalState.Y);
156     }
157 }
158 /// <summary>
159 /// Triggered as often as possible (fps). (Drawing, etc.)
160 /// </summary>
161 /// <param name="e"></param>
162 protected override void OnRenderFrame(FrameEventArgs e)
163 {
164     base.OnRenderFrame(e);
165     GL.Clear(ClearBufferMask.ColorBufferBit); // Clear the color
166
167     _timeline.OnRenderFrame(e, this);
168
169     _ioTimeline = _timeline.TimelineIndex;
170
171     // Reset selection
172     if (KeyboardState.IsKeyDown(Keys.Escape))
173         ResetSelection();
174
175     // Delete shape
176     if (KeyboardState.IsKeyDown(Keys.Delete))
177         if (_selectedShape != null)
178             DeleteShape(_selectedShape);
179
180     // Duplicate shape
181     if (KeyboardState.IsKeyPressed(Keys.D) && ←
182         KeyboardState.IsKeyDown(Keys.LeftControl))
183         if (_selectedShape != null)
184             DuplicateShape(_selectedShape);
185
186     // Mouse actions
187     if (MouseState.WasButtonDown(MouseButton.Left) == false && ←
188         MouseState.IsButtonDown(MouseButton.Left) == true) // ←
189         First left click
190         OnLeftMouseDown();
191     else if (MouseState.WasButtonDown(MouseButton.Left) == true && ←
192         MouseState.IsButtonDown(MouseButton.Left) == true) // ←
193         Long left click
194         OnLeftMouseEnter();
195     else if (MouseState.WasButtonDown(MouseButton.Left) == true && ←
196         MouseState.IsButtonDown(MouseButton.Left) == false) ←
197         // Release left click
198         OnLeftMouseUp();
199
200     //GL.PolygonMode(MaterialFace.FrontAndBack, PolygonMode.Line);
201     foreach (Shape shape in Shapes)
202         shape.Draw(ClientSize);
203
204     if (_userMode == UserMode.Edit)
205         _selector.Draw(ClientSize); // Only draw selector on ←
206         edit mode
207     //GL.PolygonMode(MaterialFace.FrontAndBack, PolygonMode.Fill);
208
209     if (_selectedShape != null)
210     {
211         if (_selectedShape != _selectedShapeBefore) // New shape
212         {
213             UpdateIO_UI();
214             _selectedShape.ImplementObject();
215             _selectedShapeBefore = _selectedShape;
216         }
217         else
218         {
219             if (_selectedShape.X != _ioX ||
220                 _selectedShape.Y != _ioY ||
221                 _selectedShape.Width != _ioWidth ||
222                 _selectedShape.Height != _ioHeight ||

```

```

215             _selectedShape.Color != new Color4(_ioColor.X, ←
216                 _ioColor.Y, _ioColor.Z, _ioColor.W) ||
217             _selectedShape.Angle != _ioAngle ||
218             _selectedShape.CornerRadius != _ioCornerRadius) ←
219             // If a property needs to be updated
220         {
221             _selectedShape.X = _ioX;
222             _selectedShape.Y = _ioY;
223             _selectedShape.Width = _ioWidth;
224             _selectedShape.Height = _ioHeight;
225             _selectedShape.Color = new Color4(_ioColor.X, ←
226                 _ioColor.Y, _ioColor.Z, _ioColor.W);
227             _selectedShape.Angle = _ioAngle;
228             _selectedShape.CornerRadius = _ioCornerRadius;
229
230             // Update the shape in the timeline
231             _timeline.UpdateShapeInTimeline(_selectedShape);
232
233         }
234     }
235
236     _ImGuiController.Update(this, (float)e.Time);
237     ShowUI();
238
239     //ImGui.ShowDemoWindow();
240     //ShowUIDebug();
241
242     _ImGuiController.Render(); // Render ImGui elements
243
244     SwapBuffers();
245 }
246 protected override void OnUnload()
247 {
248     base.OnUnload();
249
250     Console.WriteLine("Program stops");
251
252     GL.BindBuffer(BufferTarget.ArrayBuffer, 0);
253     GL.BindBuffer(BufferTarget.ElementArrayBuffer, 0);
254     GL.BindVertexArray(0);
255
256     Shapes.ForEach(shape => shape.DeleteObjects());
257 }
258
259 private void OnLeftMouseDown()
260 {
261     if (_userMode != UserMode.Create) // If it's not to create
262     {
263         if (ImGui.GetIO().WantCaptureMouse == false) // If it's ←
264             not ImGui click
265         {
266             _mouseOriginalState.X = (int)MouseState.X;
267             _mouseOriginalState.Y = (int)MouseState.Y;
268
269             // Retrieve the nearest shape
270             Shape? nearestShape = GetNearestShape(new ←
271                 Vector2i((int)MouseState.X, (int)MouseState.Y));
272             if (nearestShape != null)
273             {
274                 if (nearestShape != _selectedShape)
275                 {
276                     _userMode = UserMode.Edit;
277                     _selector.Select(nearestShape);
278                     _selectedShape = nearestShape;
279                 }
280
281                 (bool click, SelectorType? type) = ←
282                     _selector.HitBox(new Vector2i((int)MouseState.X, ←

```

```

281             (int)MouseState.Y));
282
283         // If the click is in a selector
284         if (click)
285         {
286             _selectorType = type ?? SelectorType.None;
287
288             if (_selectorType == SelectorType.Move && ←
289                 _selectedShape != null)
290             {
291                 // Save the original point to calculate the ←
292                 // delta when the user move the mouse
293                 _mouseOriginalState.X = _selectedShape.X;
294                 _mouseOriginalState.Y = _selectedShape.Y;
295             }
296         }
297     }
298
299     private void OnLeftMouseEnter()
300     {
301         switch (_userMode)
302         {
303             case UserMode.Create: // Create mode
304                 _selectedShape = _createMode switch
305                 {
306                     CreateMode.Line => new ←
307                         SVGLine((int)MouseState.X, ←
308                             (int)MouseState.Y, (int)MouseState.X, ←
309                             (int)MouseState.Y, Shape.DefaultColor),
310                     CreateMode.Rectangle => new SVGRectangle(0, 0, ←
311                         (int)MouseState.X, (int)MouseState.Y),
312                     CreateMode.Circle => new SVGCircle(0, ←
313                         (int)MouseState.X, (int)MouseState.Y, ←
314                         Shape.DefaultColor),
315                     CreateMode.Triangle => new ←
316                         SVGPolygon((int)MouseState.X, ←
317                             (int)MouseState.Y, 0, 0, Shape.DefaultColor),
318                     _ => throw new Exception("A create mode need to ←
319                         be selected"),
320                 };
321                 Shapes.Add(_selectedShape);
322                 _selectorType = SelectorType.Resize;
323                 _userMode = UserMode.Edit;
324                 OnLeftMouseEnter(); // Change user mode and call ←
325                     same function in order to switch to edit mode
326                 break;
327
328             case UserMode.Edit: // Edit mode
329                 if (_selectedShape != null)
330                 {
331                     switch (_selectorType)
332                     {
333                         // Move the current shape
334                         case SelectorType.Move:
335                             if (_selectedShape.IsMoveable)
336                             {
337                                 float x = MouseState.X, y = ←
338                                     MouseState.Y;
339                                 if ←
340                                     (KeyboardState.IsKeyDown(Keys.LeftShift)) ←
341                                         // SHIFT
342                                 {
343                                     if ←
344                                         (Math.Abs(_mouseOriginalState.X ←
345                                             - x) > ←
346                                             Math.Abs(_mouseOriginalState.Y ←
347                                                 - y))
348                                         y = _mouseOriginalState.Y;
349                                     else
350                                         x = _mouseOriginalState.X;
351                                 }
352                             }
353                         }
354                     }
355                 }
356             }
357         }
358     }

```

```

333
334         }
335         _selectedShape.Move(new ←
336             Vector2i((int)x, (int)y));
337
338         // Update shape in timeline
339         _timeline.UpdateShapeInTimeline(_selectedShape);
340
341         _selector.Select(_selectedShape);
342         UpdateIO_UI();
343     }
344     break;
345
346     // Resize the current shape
347     case SelectorType.Resize:
348         if (_selectedShape.IsResizeable)
349         {
350             float width, height;
351             width = MouseState.X - ←
352                 _selectedShape.X;
353             height = MouseState.Y - ←
354                 _selectedShape.Y;
355             if ←
356                 (KeyboardState.IsKeyDown(Keys.LeftShift) ←
357                  || _selectedShape.GetType() == ←
358                  typeof(SVGCircle)) // SHIFT
359             {
360                 if (width > height)
361                     height = width;
362                 else
363                     width = height;
364             }
365             _selectedShape.Resize(new ←
366                 Vector2i((int)width, (int)height));
367
368             // Update shape in timeline
369             _timeline.UpdateShapeInTimeline(_selectedShape);
370
371             _selector.Select(_selectedShape);
372             UpdateIO_UI();
373         }
374         break;
375     default:
376         break;
377     }
378 }
379
380 private void OnLeftMouseUp()
381 {
382     _selectorType = SelectorType.None;
383 }
384
385 /// <summary>
386 /// Update shape properties windows using shape properties
387 /// </summary>
388 public void UpdateIO_UI()
389 {
390     if (_selectedShape != null)
391     {
392         _ioX = _selectedShape.X;
393         _ioY = _selectedShape.Y;
394         _ioWidth = _selectedShape.Width;
395         _ioHeight = _selectedShape.Height;
396         _ioColor = new ←
397             System.Numerics.Vector4(_selectedShape.Color.R, ←
398                 _selectedShape.Color.G, _selectedShape.Color.B, ←
399                 _selectedShape.Color.A);
400         _ioAngle = _selectedShape.Angle;
401     }
402 }

```

```

395         _ioCornerRadius = _selectedShape.CornerRadius;
396     }
397 }
398
399 /// <summary>
400 /// Reset the selection
401 /// </summary>
402 public void ResetSelection()
403 {
404     _userMode = UserMode.Idle;
405     _selectorType = SelectorType.None;
406     _selectedShape = null;
407     _selectedShapeBefore = null;
408
409     _ioX = DEFAULT_IO_INT;
410     _ioY = DEFAULT_IO_INT;
411     _ioWidth = DEFAULT_IO_INT;
412     _ioHeight = DEFAULT_IO_INT;
413     _ioColor = new System.Numerics.Vector4(DEFAULT_IO_INT);
414     _ioAngle = DEFAULT_IO_INT;
415     _ioCornerRadius = DEFAULT_IO_INT;
416 }
417
418 /// <summary>
419 /// Retrieve the nearest shape base on a given point
420 /// </summary>
421 /// <param name="currentLocation">point where the shape ←
422 /// distance is going to be calculated</param>
423 /// <returns>Nearest shape</returns>
424 public Shape? GetNearestShape(Vector2i currentLocation)
425 {
426     (Shape? shape, double pythagore) nearest = (null, ←
427         double.MaxValue);
428
429     foreach (Shape shape in Shapes)
430     {
431         List<Vector2i> points = shape.GetSelectablePoints();
432
433         foreach (Vector2i point in points)
434         {
435             double pythagore = Math.Sqrt(Math.Pow(point.X - ←
436                 currentLocation.X, 2) + Math.Pow(point.Y - ←
437                 currentLocation.Y, 2));
438
439             if (pythagore < nearest.pythagore) // Get the ←
440                 nearest pythagore value
441                 nearest = (shape, pythagore);
442         }
443     }
444     return nearest.shape;
445 }
446
447 /// <summary>
448 /// Draw the debug UI
449 /// </summary>
450 private void ShowUIDebug()
451 {
452     ImGui.Begin("Debug");
453     ImGui.Text("SelectedShape:");
454     if (_selectedShape != null)
455     {
456         ImGui.Text(string.Format("Name:{0}", ←
457             _selectedShape.GetType().Name));
458         ImGui.Text(string.Format("UID:{0}", _selectedShape.Id));
459         ImGui.Text(string.Format("Hash:{0}", ←
460             _selectedShape.GetHashCode()));
461     }
462     else
463         ImGui.Text("none");
464
465     ImGui.Spinning();
466     ImGui.Separator();

```

```

460     ImGui.Spacing();
461
462     if (_selectedShape != null)
463     {
464         foreach (Renderer vao in _selectedShape.Renderers)
465         {
466             ImGui.Text(String.Format("VAO:{0}", ←
467                         vao.VertexArrayObjectID));
468             ImGui.Text(String.Format("VBO:{0}", ←
469                         vao.VertexBufferObjectID));
470             ImGui.Text(String.Format("IBO:{0}", ←
471                         vao.IndexBufferObjectID));
472         }
473     }
474     else
475         ImGui.Text("none");
476
477     ImGui.Spacing();
478     ImGui.Separator();
479     ImGui.Spacing();
480
481     ImGui.Text("Context:");
482     ImGui.Text(String.Format("UserMode:{0}", ←
483                         _userMode.ToString()));
484     ImGui.Text(String.Format("CreateMode:{0}", ←
485                         _createMode.ToString()));
486
487     ImGui.Spacing();
488     ImGui.Separator();
489     ImGui.Spacing();
490     ImGui.Text("Timeline");
491     int numberColumns = 0, numberOfRows = 0;
492
493     if (_timeline.SortedTimeline != null)
494     {
495         numberColumns = _timeline.SortedTimeline.Count;
496         if (_timeline.SortedTimeline.Count > 0)
497             numberOfRows = _timeline.SortedTimeline.Max(i => ←
498                         i.Value != null ? i.Value.Count : 0);
499     }
500
501     if (numberColumns > 0)
502     {
503         if (ImGui.BeginTable("elements", numberColumns, ←
504                         ImGuiTableFlags.RowBg | ImGuiTableFlags.Borders))
505         {
506             foreach (KeyValuePair<int, List<Shape>> shapes in ←
507                         _timeline.SortedTimeline)
508                 if (shapes.Value != null)
509                     ImGui.TableSetupColumn(shapes.Key.ToString());
510             ImGui.TableHeadersRow();
511
512             for (int row = 0; row < numberOfRows; row++)
513             {
514                 int columnIndex = 0;
515                 ImGui.TableNextRow();
516                 foreach (KeyValuePair<int, List<Shape>> shapes ←
517                         in _timeline.SortedTimeline)
518                 {
519                     if (shapes.Value != null)
520                     {
521                         if (shapes.Value.Count > row)
522                         {
523                             ImGui.TableSetColumnIndex(columnIndex);
524                             ImGui.Text(string.Format("{0}", ←
525                                         shapes.Value[row].GetType().Name));
526                             ImGui.Text(string.Format("{0}", ←
527                                         shapes.Value[row].Id));
528                             ImGui.Text(string.Format("{0}", ←
529                                         shapes.Value[row].GetHashCode()));
530                         foreach (Renderer vao in ←
531                                         shapes.Value[row].Renderers)
532                         {
533                             ImGui.TableSetColumnIndex(columnIndex);
534                             ImGui.Text(string.Format("{0}", ←
535                                         vao.VertexArrayObjectID));
536                             ImGui.Text(string.Format("{0}", ←
537                                         vao.VertexBufferObjectID));
538                             ImGui.Text(string.Format("{0}", ←
539                                         vao.IndexBufferObjectID));
540                         }
541                     }
542                 }
543             }
544         }
545     }
546 
```

```

519
520             {
521                 ImGui.Text(String.Format("VAO: ↵
522                                     {0}", ↵
523                                     vao.VertexArrayObjectID));
524                 ImGui.Text(String.Format("VBO: ↵
525                                     {0}", ↵
526                                     vao.VertexBufferObjectID));
527                 ImGui.Text(String.Format("IBO: ↵
528                                     {0}", ↵
529                                     vao.IndexBufferObjectID));
530             }
531         }
532     }
533 }
534
535 /// <summary>
536 /// Draw the UI
537 /// </summary>
538 private void ShowUI()
539 {
540     // ImGui settings
541     ImGui.GetStyle().WindowRounding = 0.0f;
542     ImGui.GetStyle().ScrollbarRounding = 0.0f;
543     ImGui.GetStyle().LogSliderDeadzone = 0.0f;
544     ImGui.GetStyle().TabRounding = 0.0f;
545
546     // Parameters side
547     ImGui.Begin("Parameters", ImGuiWindowFlags.NoCollapse | ↵
548                 ImGuiWindowFlags.NoResize | ImGuiWindowFlags.NoMove | ↵
549                 ImGuiWindowFlags.NoTitleBar);
550     ImGui.SetWindowSize(new System.Numerics.Vector2(200, ↵
551                             ClientSize.Y / 2));
552     ImGui.SetWindowPos(new System.Numerics.Vector2(ClientSize.X ↵
553                               - ImGui.GetWindowWidth(), 0));
554
555     ImGui.Text("Parameters");
556     ImGui.Spacing();
557     if (_selectedShape == null || _selectedShape.IsMoveable == ↵
558         false)
559         ImGui.BeginDisabled();
560
561     // Position parameters
562     if (ImGui.BeginTable("Position", 2))
563     {
564         ImGui.TableNextRow();
565         ImGui.TableSetColumnIndex(0);
566         ImGui.Text("X");
567         ImGui.SameLine();
568         if (_selectedShape == null || _selectedShape.IsMoveable == ↵
569             false)
570         {
571             ImGui.EndDisabled();
572             //HelpMarker("This shape is not moveable");
573             ImGui.BeginDisabled();
574         }
575         ImGui.TableSetColumnIndex(1);
576         ImGui.Text("Y");
577         ImGui.TableNextRow();
578         ImGui.TableSetColumnIndex(0);
579         ImGui.PushItemWidth(82);
580         ImGui.InputInt("##X", ref _ioX);
581         ImGui.TableSetColumnIndex(1);
582         ImGui.PushItemWidth(82);
583         ImGui.InputInt("##Y", ref _ioY);
584     }
585 }

```

```

579         ImGui.EndTable();
580     }
581     if (_selectedShape == null || _selectedShape.IsMoveable == false)
582         ImGui.EndDisabled();
583
584     ImGui.Spacing();
585     if (_selectedShape == null || _selectedShape.IsResizeable == false)
586         ImGui.BeginDisabled();
587
588     // Size parameters
589     if (ImGui.BeginTable("Size", 2))
590     {
591         ImGui.TableNextRow();
592         ImGui.TableSetColumnIndex(0);
593         ImGui.Text("Width");
594         ImGui.SameLine();
595         if (_selectedShape == null || !_selectedShape.IsResizeable)
596         {
597             ImGui.EndDisabled();
598             //HelpMarker("This shape is not resizeable");
599             ImGui.BeginDisabled();
600         }
601         ImGui.TableSetColumnIndex(1);
602         ImGui.Text("Height");
603         ImGui.TableNextRow();
604         ImGui.TableSetColumnIndex(0);
605         ImGui.PushItemWidth(82);
606         if (ImGui.InputInt("##Width", ref _ioWidth))
607         {
608             if (_selectedShape != null) // Constraint verification
609                 if (_selectedShape.GetType() == typeof(SVGCircle))
610                     _ioHeight = _ioWidth;
611         }
612         ImGui.TableSetColumnIndex(1);
613         ImGui.PushItemWidth(82);
614         if (ImGui.InputInt("##Height", ref _ioHeight))
615         {
616             if (_selectedShape != null) // Constraint verification
617                 if (_selectedShape.GetType() == typeof(SVGCircle))
618                     _ioWidth = _ioHeight;
619         }
620
621         ImGui.EndTable();
622     }
623     if (_selectedShape == null || _selectedShape.IsResizeable == false)
624         ImGui.EndDisabled();
625
626     // Other parameters
627     if (ImGui.BeginTable("Other", 2))
628     {
629         ImGui.TableNextRow();
630         ImGui.TableSetColumnIndex(0);
631         if (_selectedShape == null || !_selectedShape.IsAngleChangeable == false)
632             ImGui.BeginDisabled();
633         ImGui.Text("Angle");
634         if (_selectedShape == null || !_selectedShape.IsAngleChangeable == false)
635             ImGui.EndDisabled();
636
637
638         ImGui.TableSetColumnIndex(1);
639         if (_selectedShape == null || !_selectedShape.IsCornerRadiusChangeable == false)
640             ImGui.BeginDisabled();
641         ImGui.Text("Radius");
642         if (_selectedShape == null || !_selectedShape.IsCornerRadiusChangeable == false)

```

```

643         ImGui.EndDisabled();
644
645
646         ImGui.TableNextRow();
647
648
649         ImGui.TableSetColumnIndex(0);
650         ImGui.PushItemWidth(82);
651         if (_selectedShape == null || ←
652             _selectedShape.IsAngleChangeable == false)
653             ImGui.BeginDisabled();
654         ImGui.InputInt("##Angle", ref _ioAngle);
655         if (_selectedShape == null || ←
656             _selectedShape.IsAngleChangeable == false)
657             ImGui.EndDisabled();
658
659
660         ImGui.TableSetColumnIndex(1);
661         ImGui.PushItemWidth(82);
662         if (_selectedShape == null || ←
663             _selectedShape.IsCornerRadiusChangeable == false)
664             ImGui.BeginDisabled();
665         ImGui.InputInt("##Radius", ref _ioCornerRadius);
666         _ioCornerRadius = Math.Max(_ioCornerRadius, 0);
667         if (_selectedShape == null || ←
668             _selectedShape.IsCornerRadiusChangeable == false)
669             ImGui.EndDisabled();
670
671         ImGui.EndTable();
672     }
673
674     ImGui.Spacing();
675
676     if (_selectedShape == null)
677         ImGui.BeginDisabled();
678     ImGui.Text("Color");
679     ImGui.ColorEdit4("##Color", ref _ioColor);
680     if (_selectedShape == null)
681         ImGui.EndDisabled();
682
683     ImGui.Spacing();
684     ImGui.Separator();
685     ImGui.Spacing();
686
687     ImGui.Text("Background Color");
688     ImGui.Spacing();
689     if (ImGui.ColorEdit3("##BgColor", ref _ioBgColor))
690         GL.ClearColor(_ioBgColor.X, _ioBgColor.Y, _ioBgColor.Z, ←
691                         1.0f);
692     ImGui.End();
693
694     // Tree view side
695     ImGui.Begin("Treeview", ImGuiWindowFlags.NoCollapse | ←
696                 ImGuiWindowFlags.NoResize | ImGuiWindowFlags.NoMove | ←
697                 ImGuiWindowFlags.NoTitleBar);
698     ImGui.SetWindowSize(new System.Numerics.Vector2(200, ←
699                         ClientSize.Y / 2));
700     ImGui.SetWindowPos(new System.Numerics.Vector2(ClientSize.X ←
701                         - ImGui.GetScreenWidth(), ClientSize.Y / 2));
702     ImGui.Text("TreeView");
703     ImGui.Spacing();
704     for (int i = Shapes.Count - 1; i >= 0; i--)
705     {
706         if (ImGui.Selectable(String.Format("{0} - {1}##{2}", ←
707             Shapes[i].ShortId, Shapes[i].GetType().Name, ←
708             Shapes[i].GetHashCode()), _selectedShape == Shapes[i]))
709         {
710             _selectedShape = Shapes[i];
711             _selector.Select(Shapes[i]);
712             _userMode = UserMode.Edit;
713         }
714         if (i - 1 >= 0)

```

```

704        {
705            if (ImGui.ArrowButton(String.Format("Down##d{0}", i), ImGuiDir.Down))
706            {
707                InvertShape(i, i - 1);
708                SelectShape(Shapes[i - 1]);
709            }
710        }
711        if (i + 1 < Shapes.Count)
712        {
713            if (i - 1 >= 0)
714                ImGui.SameLine();
715            if (ImGui.ArrowButton(String.Format("Up##u{0}", i), ImGuiDir.Up))
716            {
717                InvertShape(i, i + 1);
718                SelectShape(Shapes[i + 1]);
719            }
720        }
721        ImGui.Separator();
722    }
723    ImGui.End();
724
725    // Animation side
726    ImGui.Begin("Animation", ImGuiWindowFlags.NoCollapse | ←
727        ImGuiWindowFlags.NoResize | ImGuiWindowFlags.NoMove | ←
728        ImGuiWindowFlags.NoTitleBar);
729    ImGui.SetWindowSize(new ←
730        System.Numerics.Vector2(ClientSize.X / 2, 200));
731    ImGui.SetWindowPos(new System.Numerics.Vector2(0, ←
732        ClientSize.Y - ImGui.GetWindowHeight()));
733    ImGui.Text("Animation");
734    ImGui.Spacing();
735
736    int numberColumns = _timeline.SortedTimeline.Count;
737    if (numberColumns > 0)
738    {
739        foreach (Shape shape in Shapes)
740        {
741            if (ImGui.BeginTable("shape", 2, ←
742                ImGuiTableFlags.Resizable))
743            {
744                ImGui.TableNextRow();
745                ImGui.TableSetColumnIndex(0);
746                ImGui.Text(string.Format("{0} - {1}", ←
747                    shape.ShortId, shape.GetType().Name));
748                ImGui.TableSetColumnIndex(1);
749                if ←
750                    (ImGui.BeginTable(String.Format("elements##{0}", ←
751                        shape.Id), numberColumns + 1, ←
752                        ImGuiTableFlags.Borders))
753                {
754                    ImGui.TableSetupColumn("Properties");
755                    foreach (KeyValuePair<int, List<Shape>> ←
756                        shapes in _timeline.SortedTimeline)
757                        ImGui.TableSetupColumn(shapes.Key.ToString());
758                    ImGui.TableHeadersRow();
759
760                    int i = 0;
761
762                    if (shape.IsMoveable)
763                    {
764                        i = 0;
765                        ImGui.TableNextRow();
766                        ImGui.TableSetColumnIndex(i);
767                        ImGui.Text("X");
768                        foreach (KeyValuePair<int, List<Shape>> ←
769                            timeline in _timeline.SortedTimeline)
770                        {
771                            i++;
772                        }
773                    }
774                }
775            }
776        }
777    }

```

```

762             Shape? sibling = ←
763                 timeline.Value.Find(x => x.Id == ←
764                     shape.Id);
765             if (sibling != null)
766             {
767                 ImGui.TableSetColumnIndex(i);
768                 ImGui.Text(sibling.X.ToString());
769             }
770         }
771         i = 0;
772         ImGui.TableNextRow();
773         ImGui.TableSetColumnIndex(i);
774         ImGui.Text("Y");
775         foreach (KeyValuePair<int, List<Shape>> ←
776                         timeline in _timeline.SortedTimeline)
777         {
778             i++;
779             Shape? sibling = ←
780                 timeline.Value.Find(x => x.Id == ←
781                     shape.Id);
782             if (sibling != null)
783             {
784                 ImGui.TableSetColumnIndex(i);
785                 ImGui.Text(sibling.Y.ToString());
786             }
787         }
788         if (shape.IsResizeable)
789         {
790             i = 0;
791             ImGui.TableNextRow();
792             ImGui.TableSetColumnIndex(i);
793             ImGui.Text("Width");
794             foreach (KeyValuePair<int, List<Shape>> ←
795                         timeline in _timeline.SortedTimeline)
796             {
797                 i++;
798                 Shape? sibling = ←
799                     timeline.Value.Find(x => x.Id == ←
800                         shape.Id);
801                 if (sibling != null)
802                 {
803                     ImGui.TableSetColumnIndex(i);
804                     ImGui.Text(sibling.Width.ToString());
805                 }
806             }
807             i = 0;
808             ImGui.TableNextRow();
809             ImGui.TableSetColumnIndex(i);
810             ImGui.Text("Height");
811             foreach (KeyValuePair<int, List<Shape>> ←
812                         timeline in _timeline.SortedTimeline)
813             {
814                 i++;
815                 Shape? sibling = ←
816                     timeline.Value.Find(x => x.Id == ←
817                         shape.Id);
818                 if (sibling != null)
819                 {
820                     ImGui.TableSetColumnIndex(i);
821                     ImGui.Text(sibling.Height.ToString());
822                 }
823             }
824             if (shape.AngleChangeable)
825             {
826                 i = 0;
827                 ImGui.TableNextRow();

```

```

823     ImGui.TableSetColumnIndex(i);
824     ImGui.Text("Angle");
825     foreach (KeyValuePair<int, List<Shape>> timeline in _timeline.SortedTimeline)
826     {
827         i++;
828         Shape? sibling = timeline.Value.Find(x => x.Id == shape.Id);
829         if (sibling != null)
830         {
831             ImGui.TableSetColumnIndex(i);
832             ImGui.Text(String.Format("{0}°", sibling.Angle));
833         }
834     }
835 }
836
837 if (shape.IsCornerRadiusChangeable)
838 {
839     i = 0;
840     ImGui.TableNextRow();
841     ImGui.TableSetColumnIndex(i);
842     ImGui.Text("CornerRadius");
843     foreach (KeyValuePair<int, List<Shape>> timeline in _timeline.SortedTimeline)
844     {
845         i++;
846         Shape? sibling = timeline.Value.Find(x => x.Id == shape.Id);
847         if (sibling != null)
848         {
849             ImGui.TableSetColumnIndex(i);
850             ImGui.Text(sibling.CornerRadius.ToString());
851         }
852     }
853 }
854
855 i = 0;
856 ImGui.TableNextRow();
857 ImGui.TableSetColumnIndex(i);
858 ImGui.Text("Color");
859 foreach (KeyValuePair<int, List<Shape>> timeline in _timeline.SortedTimeline)
860 {
861     i++;
862     Shape? sibling = timeline.Value.Find(x => x.Id == shape.Id);
863     if (sibling != null)
864     {
865         ImGui.TableSetColumnIndex(i);
866         ImGui.Text(String.Format("RGBA({0},{1},{2},{3})", sibling.Color.R, sibling.Color.G, sibling.Color.B, sibling.Color.A));
867     }
868 }
869
870     ImGui.EndTable();
871 }
872 ImGui.EndTable();
873 ImGui.Spacing();
874 }
875 }
876 }
877 ImGui.End();
878
879 // Timeline side
880

```

```

881     ImGui.Begin("Timeline", ImGuiWindowFlags.NoCollapse | ←
882                 ImGuiWindowFlags.NoResize | ImGuiWindowFlags.NoMove | ←
883                 ImGuiWindowFlags.NoTitleBar);
884     ImGui.SetWindowSize(new ←
885                         System.Numerics.Vector2(ClientSize.X / 2 - 200, 200));
886     ImGui.SetWindowPos(new System.Numerics.Vector2(ClientSize.X ←
887                         / 2, ClientSize.Y - ImGui.GetWindowHeight()));
888
889     ImGui.Text("Timeline");
890     ImGui.Spacing();
891     if (ImGui.SliderInt("frame", ref _ioTimeline, ←
892                           Timeline.MIN_TIMELINE, Timeline.MAX_TIMELINE))
893     {
894         _timeline.TimelineIndex = _ioTimeline;
895         _timeline.RenderInterpolation(Shapes);
896         ResetSelection();
897     }
898
899     ImGui.SameLine();
900
901     ImGui.PushItemWidth(80f);
902     if (ImGui.InputInt("fps", ref _ioFps))
903     {
904         _ioFps = Math.Clamp(_ioFps, Timeline.MIN_FPS, ←
905                             Timeline.MAX_FPS);
906         _timeline.Fps = _ioFps;
907     }
908     ImGui.PopItemWidth();
909
910     if (_selectedShape != null)
911     {
912         if ←
913             (_timeline.SortedTimeline.ContainsKey(_timeline.TimelineIndex) ←
914              && _timeline.SortedTimeline[_timeline.TimelineIndex] ←
915              != null && ←
916              _timeline.SortedTimeline[_timeline.TimelineIndex].Any(x ←
917              => x.Id == _selectedShape.Id))
918         {
919             if (ImGui.Button(String.Format("Remove keyframe {0} ←
920                             for {1}", _timeline.TimelineIndex, ←
921                             _selectedShape.GetType().Name)))
922             {
923                 _timeline.SortedTimeline[_timeline.TimelineIndex].Remove(_ti
924
925                 => x.Id == _selectedShape.Id)); // Can't be ←
926                 null
927                 // If timeline empty then null it
928                 if ←
929                     (_timeline.SortedTimeline[_timeline.TimelineIndex].Count
930                      == 0)
931                     _timeline.SortedTimeline.Remove(_timeline.TimelineIndex)
932
933             }
934             else
935             {
936                 if (ImGui.Button(String.Format("Create keyframe for ←
937                             {0}", _selectedShape.GetType().Name)))
938                 {
939                     if ←
940                         (_timeline.SortedTimeline.ContainsKey(_timeline.Timeline
941                           == false || ←
942                           _timeline.SortedTimeline[_timeline.TimelineIndex] ←
943                           == null)
944                           _timeline.SortedTimeline[_timeline.TimelineIndex] ←
945                           = new List<Shape>());
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977

```

```

928         foreach (Shape shape in ←
929             _timeline.SortedTimeline[_timeline.TimelineIndex])
930         {
931             if (shape.Id == _selectedShape.Id)
932             {
933                 _timeline.SortedTimeline[_timeline.TimelineIndex].Re
934             }
935         }
936     _timeline.SortedTimeline[_timeline.TimelineIndex].Add(_selec
937 }
938 }
939
940 ImGui.End();
941
942
943
944 // Navbar side
945 ImGui.Begin("NavBar", ImGuiWindowFlags.NoCollapse | ←
946             ImGuiWindowFlags.NoResize | ImGuiWindowFlags.NoMove | ←
947             ImGuiWindowFlags.NoTitleBar | ImGuiWindowFlags.MenuBar);
948 ImGui.SetWindowSize(new ←
949             System.Numerics.Vector2(ClientSize.X - 200, 0));
950 ImGui.SetWindowPos(new System.Numerics.Vector2(0, 0));
951
952 if (ImGui.BeginMenuBar())
953 {
954     if (ImGui.BeginMenu("File"))
955     {
956         if (ImGui.BeginMenu("Open"))
957         {
958             if (ImGui.MenuItem("Open\u2022new\u2022project"))
959             {
960                 _dialogFilePicker = true;
961                 _importMode = ImportMode.Workspace;
962             }
963             if (ImGui.MenuItem("Import\u2022a\u2022SVG"))
964             {
965                 _dialogFilePicker = true;
966                 _importMode = ImportMode.Add;
967             }
968             if (ImGui.MenuItem("Open\u2022a\u2022new\u2022project\u2022from\u2022a\u2022
969                 SVG"))
970             {
971                 _dialogFilePicker = true;
972                 _importMode = ImportMode.Override;
973             }
974             ImGui.EndMenu();
975         }
976         if (ImGui.BeginMenu("Save"))
977         {
978             if (ImGui.MenuItem("Save"))
979             {
980                 _dialogFileSaver = true;
981                 _exportMode = ExportMode.Workspace;
982             }
983             if (ImGui.MenuItem("Save\u2022as\u2022SVG"))
984             {
985                 _dialogFileSaver = true;
986                 _exportMode = ExportMode.SVG;
987             }
988             if (ImGui.MenuItem("Save\u2022as\u2022MP4"))
989             {
990                 _dialogFileSaver = true;
991                 _exportMode = ExportMode.MP4;
992             }
993             if (ImGui.MenuItem("Save\u2022as\u2022GIF"))
994             {
995                 _dialogFileSaver = true;
996                 _exportMode = ExportMode.GIF;
997             }
998             if (ImGui.MenuItem("Save\u2022as\u2022PNG"))
999             {
1000                 _dialogFileSaver = true;
1001                 _exportMode = ExportMode.PNG;
1002             }
1003         }
1004     }
1005 }
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094

```

```

995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
    {
        _dialogFileSaver = true;
        _exportMode = ExportMode.PNG;
    }
    ImGui.EndMenu();
}
ImGui.EndMenu();
}
ImGui.EndMenuBar();
}
if (ImGui.Button("Line"))
{
    _userMode = UserMode.Create;
    _createMode = CreateMode.Line;
}
ImGui.SameLine();
if (ImGui.Button("PrimitiveShape"))
    ImGui.OpenPopup("primitive_popup");
if (ImGui.BeginPopup("primitive_popup"))
{
    ImGui.Text("Select a shape");
    ImGui.Separator();
    if (ImGui.Selectable("Circle"))
    {
        _userMode = UserMode.Create;
        _createMode = CreateMode.Circle;
    }
    if (ImGui.Selectable("Rectangle"))
    {
        _userMode = UserMode.Create;
        _createMode = CreateMode.Rectangle;
    }
    if (ImGui.Selectable("Triangle"))
    {
        _userMode = UserMode.Create;
        _createMode = CreateMode.Triangle;
    }
    ImGui.EndPopup();
}
// File picker dialog
if (_dialogFileSaver)
    ImGui.OpenPopup("save-file");
if (ImGui.BeginPopupModal("save-file"))
{
    var picker = FileSaver.GetFilePicker(this, ←
        Path.Combine(Environment.CurrentDirectory, ←
        "Content/Atlases"));
    if (picker.Draw())
    {
        string path = picker.CurrentFolder + '\\\\' + ←
            picker.Filename;
        switch (_exportMode)
        {
            case ExportMode.Workspace:
                SaveFreeFrameWorkspace(path);
                break;
            case ExportMode.GIF:
                SaveCurrentScreenToGIF(path);
                break;
            case ExportMode.MP4:
                SaveCurrentScreenToMP4(path);
                break;
            case ExportMode.PNG:
                SaveCurrentScreenToPNG(path);
                break;
            case ExportMode.SVG:
                SaveCurrentScreenToSVG(path);
                break;
            default:
                break;
        }
    }
}

```

```

1064         }
1065         FileSaver.Clear();
1066     }
1067     _dialogFileSaver = false;
1068     ImGui.EndPopup();
1069 }
1070
1071 // File picker dialog
1072 if (_dialogFilePicker)
1073     ImGui.OpenPopup("open-file");
1074 if (ImGui.BeginPopupModal("open-file"))
1075 {
1076     var picker = FilePicker.GetFilePicker(this, ←
1077         Path.Combine(Environment.CurrentDirectory, ←
1078         "Content/Atlases"), _importMode == ←
1079         ImportMode.Workspace ? ".freeframe" : ".svg");
1080     if (picker.Draw())
1081     {
1082         ResetSelection();
1083         try
1084         {
1085             switch (_importMode)
1086             {
1087                 case ImportMode.Workspace:
1088                     ImportFreeFrameWorkspace(picker.SelectedFile);
1089                     break;
1090                 case ImportMode.Add:
1091                     (List<Shape> newShapes, ←
1092                      SortedDictionary<int, List<Shape>> ←
1093                      newTimeline, _dialogCompatibility) = ←
1094                      Importer.ImportFromFile(picker.SelectedFile);
1095                     Shapes.AddRange(newShapes);
1096                     break;
1097                 case ImportMode.Override:
1098                     (Shapes, _timeline.SortedTimeline, ←
1099                      _dialogCompatibility) = ←
1100                      Importer.ImportFromFile(picker.SelectedFile);
1101                     _timeline.ResetTimeline();
1102                     break;
1103                 default:
1104                     break;
1105             }
1106         }
1107         catch (Exception)
1108         {
1109             _dialogError = true;
1110         }
1111         FilePicker.Clear();
1112     }
1113     _dialogFilePicker = false;
1114     ImGui.EndPopup();
1115 }
1116
1117 // Compatibility alert
1118 if (_dialogCompatibility)
1119     ImGui.OpenPopup("Compatibility Problem");
1120
1121 if (ImGui.BeginPopupModal("Compatibility Problem"))
1122 {
1123     ImGui.Text("Some SVG elements are not compatible. Go to ←
1124                 the list of compatible SVG elements");
1125     ImGui.Separator();
1126     if (ImGui.Button("OK"))
1127     {
1128         ImGui.CloseCurrentPopup();
1129         _dialogCompatibility = false;
1130     }
1131     ImGui.EndPopup();
1132 }

```

```

1127     // Error alert
1128     if (_dialogError)
1129         ImGui.OpenPopup("Error_Problem");
1130     if (ImGui.BeginPopupModal("Error_Problem"))
1131     {
1132         ImGui.Text("There was an error while importing the file, please check the syntax");
1133         ImGui.Separator();
1134         if (ImGui.Button("OK"))
1135         {
1136             ImGui.CloseCurrentPopup();
1137             _dialogError = false;
1138         }
1139         ImGui.EndPopup();
1140     }
1141     ImGui.End();
1142 }
1143
1144 /// <summary>
1145 /// Select the given shape
1146 /// </summary>
1147 /// <param name="shape">Shape to be selected</param>
1148 public void SelectShape(Shape shape)
1149 {
1150     _selectedShape = shape;
1151     _selector.Select(shape);
1152     _userMode = UserMode.Edit;
1153 }
1154
1155 /// <summary>
1156 /// Delete the given shape
1157 /// </summary>
1158 /// <param name="shape">Shape to be deleted</param>
1159 public void DeleteShape(Shape shape)
1160 {
1161     _timeline.RemoveElementInTimeline(shape);
1162
1163     Shapes.Remove(shape);
1164     shape.DeleteObjects();
1165
1166     ResetSelection();
1167 }
1168
1169 /// <summary>
1170 /// Duplicate the given shape
1171 /// </summary>
1172 /// <param name="shape">Shape to be duplicated</param>
1173 public void DuplicateShape(Shape shape)
1174 {
1175     Shape copy = shape.DeepCopy();
1176     ResetSelection();
1177     Shapes.Add(copy);
1178     copy.ImplementObject();
1179 }
1180
1181 /// <summary>
1182 /// Invert shapes position (layer)
1183 /// </summary>
1184 /// <param name="index1">First shape index</param>
1185 /// <param name="index2">Second shape index</param>
1186 public void InvertShape(int index1, int index2)
1187 {
1188     Shape tmpShape = Shapes[index2].ShallowCopy();
1189     Shapes[index2] = Shapes[index1].ShallowCopy();
1190     Shapes[index1] = tmpShape;
1191 }
1192
1193 /// <summary>
1194 /// Save the current workspace in MP4
1195 /// </summary>
1196 /// <param name="path">Location of the MP4 output</param>
1197 public void SaveCurrentScreenToMP4(string path)

```

```

1198     {
1199         using VideoWriter w = new VideoWriter(path + ".mp4", ←
1200             _timeline.Fps, new System.Drawing.Size(ClientSize.X, ←
1201             ClientSize.Y), true);
1202         for (int i = Timeline.MIN_TIMELINE; i <= ←
1203             Timeline.MAX_TIMELINE; i++)
1204         {
1205             RenderFrameBySecondIndex(i);
1206             w.Write(TakeSnap().ToMat());
1207         }
1208     }
1209 
1210     /// <summary>
1211     /// Save the current workspace in the FreeFrame format
1212     /// </summary>
1213     /// <param name="path">Location of the FreeFrame output</param>
1214     public void SaveFreeFrameWorkspace(string path)
1215     {
1216         Workspace workspace = new()
1217         {
1218             SortedTimeline = _timeline.SortedTimeline,
1219             BgColor = _ioBgColor,
1220             Shapes = Shapes
1221         };
1222 
1223         string jsonString = JsonConvert.SerializeObject(workspace, ←
1224             Formatting.Indented, new JsonSerializerSettings
1225         {
1226             TypeNameHandling = TypeNameHandling.Auto, // Because I ←
1227                 have abstract classes
1228             ReferenceLoopHandling = ReferenceLoopHandling.Ignore
1229         });
1230 
1231         File.WriteAllText(path + ".freeframe", jsonString);
1232     }
1233 
1234     /// <summary>
1235     /// Import a FreeFrame file
1236     /// </summary>
1237     /// <param name="path">Location of the FreeFrame file</param>
1238     public void ImportFreeFrameWorkspace(string path)
1239     {
1240         Workspace fromJson = ←
1241             JsonConvert.DeserializeObject<Workspace>(File.ReadAllText(path), ←
1242                 new JsonSerializerSettings
1243             {
1244                 TypeNameHandling = TypeNameHandling.Auto // Because I ←
1245                     have abstract classes
1246             });
1247 
1248         Shapes = fromJson.Shapes ?? new();
1249         _timeline.SortedTimeline = fromJson.SortedTimeline ?? new();
1250         _ioBgColor = fromJson.BgColor;
1251 
1252         // Implement new objects and render
1253         foreach (Shape shape in Shapes)
1254             shape.ImplementObject();
1255         _timeline.RenderInterpolation(Shapes);
1256         ResetSelection();
1257     }
1258 
1259     /// <summary>
1260     /// Save the current workspace in the GIF format
1261     /// </summary>
1262     /// <param name="path">Location of the GIF output</param>
1263     public void SaveCurrentScreenToGIF(string path)
1264     {
1265         using (var gif = AnimatedGif.AnimatedGif.Create(path + ←
1266             ".gif", 1000 / _timeline.Fps))
1267         {
1268             for (int i = Timeline.MIN_TIMELINE; i <= ←
1269                 Timeline.MAX_TIMELINE; i++)

```

```

1260         {
1261             RenderFrameBySecondIndex(i);
1262             gif.AddFrame(TakeSnap(), delay: -1, quality: ←
1263             GifQuality.Bit8);
1264         }
1265     }
1266
1267     /// <summary>
1268     /// Save the current workspace in the SVG format
1269     /// </summary>
1270     /// <param name="path">Location of the SVG output</param>
1271     public void SaveCurrentScreenToSVG(string path)
1272     {
1273         Importer.ExportToFile(Shapes, ClientSize, path + ".svg");
1274     }
1275
1276     /// <summary>
1277     /// Save the current workspace in the PNG format
1278     /// </summary>
1279     /// <param name="path">Location of the PNG output</param>
1280     public void SaveCurrentScreenToPNG(string path)
1281     {
1282         RenderFrameBySecondIndex(_timeline.TimelineIndex);
1283         TakeSnap().Save(path + ".png", ImageFormat.Png);
1284     }
1285
1286     /// <summary>
1287     /// Render the timeline
1288     /// </summary>
1289     /// <param name="second">Index of the timeline to be ←
1290     renderer</param>
1291     public void RenderFrameBySecondIndex(int second)
1292     {
1293         GL.Clear(ClearBufferMask.ColorBufferBit); // Clear the color
1294         ResetSelection();
1295
1296         _timeline.TimelineIndex = second;
1297         _timeline.RenderInterpolation(Shapes);
1298
1299         ResetSelection();
1300
1301         foreach (Shape shape in Shapes)
1302             shape.Draw(ClientSize);
1303     }
1304
1305     /// <summary>
1306     /// Take a screenshot of the current user view
1307     /// </summary>
1308     /// <returns>Bitmap that contains all the pixels of the ←
1309     screenshot</returns>
1310     public Bitmap TakeSnap()
1311     {
1312         Bitmap bmp = new Bitmap(ClientSize.X, ClientSize.Y);
1313
1314         // Lock the bits
1315         BitmapData bmpData = bmp.LockBits(new ←
1316             System.Drawing.Rectangle(0, 0, ClientSize.X, ←
1317             ClientSize.Y), ImageLockMode.WriteOnly, ←
1318             System.Drawing.Imaging.PixelFormat.Format24bppRgb);
1319
1320         // Fill with current window
1321         GL.ReadPixels(0, 0, ClientSize.X, ClientSize.Y, ←
1322             OpenTK.Graphics.OpenGL4.PixelFormat.Bgr, ←
1323             PixelType.UnsignedByte, bmpData.Scan0);
1324
1325         bmp.UnlockBits(bmpData);
1326
1327         bmp.RotateFlip(RotateFlipType.RotateNoneFlipY);
1328
1329         return bmp;
1330     }

```

```
1324    }
1325 }
```

Listing 1.2 – ./source/csharp/Window.cs

1.1.3 Importer.cs

```
1 ﻿using FreeFrame.Components.Shapes;
2  using OpenTK.Mathematics;
3  using System.Text;
4  using System.Xml;
5
6 namespace FreeFrame.Components
7 {
8     public static class Importer
9     {
10         /// <summary>
11         /// Import shapes list and timeline from a stream
12         /// </summary>
13         /// <param name="pStream">Given stream</param>
14         /// <returns>Shape list and timeline</returns>
15         /// <exception cref="Exception">If something went wrong in the ←
16         importation</exception>
17         static private (List<Shape>, SortedDictionary<int, ←
18             List<Shape>>, bool) ImportFromStream(Stream pStream)
19         {
20             List<Shape> shapes = new List<Shape>();
21             SortedDictionary<int, List<Shape>> timeline = new ←
22                 SortedDictionary<int, List<Shape>>();
23             bool compatibilityFlag = false;
24
25             Shape? previous = null;
26
27             using (XmlReader reader = XmlReader.Create(pStream))
28             {
29                 try
30                 {
31                     while (reader.Read())
32                     {
33                         if (reader.HasAttributes)
34                         {
35                             switch (reader.Name)
36                             {
37                                 case "xml":
38                                     case "svg":
39                                         break; // Skip knowned elements
40                                 case "polygon":
41                                     shapes.Add(new SVGPolygon(reader));
42                                     previous = shapes.Last();
43                                     break;
44                                 case "path":
45                                     shapes.Add(new SVGPath(reader));
46                                     previous = shapes.Last();
47                                     break;
48                                 case "rect":
49                                     shapes.Add(new SVGRectangle(reader));
50                                     previous = shapes.Last();
51                                     break;
52                                 case "circle":
53                                     shapes.Add(new SVGCircle(reader));
54                                     previous = shapes.Last();
55                                     break;
56                                 case "line":
57                                     shapes.Add(new SVGLine(reader));
58                                     previous = shapes.Last();
59                                     break;
60                                 default:
61                                     compatibilityFlag = true; // If an ←
62                                         element is unknow, the flag is ←
63                                         trigger
64
65             }
66         }
67     }
68 }
```

```

59                     break;
60                 }
61             }
62         }
63     }
64     catch (Exception)
65     {
66         throw new Exception("Error while importing");
67     }
68 }
69 return (shapes, new SortedDictionary<int, List<Shape>>(), ←
70       compatibilityFlag);
71 }
72
73 /// <summary>
74 /// Import shapes list and timeline from a file path
75 /// </summary>
76 /// <param name="pFilename">File path</param>
77 /// <returns>Shape list and timeline</returns>
78 /// <exception cref="ArgumentException">If file cannot be ←
79     found</exception>
80 static public (List<Shape>, SortedDictionary<int, List<Shape>>, ←
81   bool) ImportFromFile(string pFilename)
82 {
83     if (!File.Exists(pFilename))
84         throw new ArgumentException($"'{pFilename}' file cannot be ←
85           found.", nameof(pFilename));
86
87     byte[] byteArray = ←
88         Encoding.UTF8.GetBytes(File.ReadAllText(pFilename));
89
90     return ImportFromStream(new MemoryStream(byteArray));
91 }
92
93 /// <summary>
94 /// Import shapes list and timeline from a string
95 /// </summary>
96 /// <param name="pString">Given string</param>
97 /// <returns>Shape list and timeline</returns>
98 static public (List<Shape>, SortedDictionary<int, List<Shape>>, ←
99   bool) ImportFromString(string pString)
100 {
101     byte[] byteArray = Encoding.UTF8.GetBytes(pString);
102
103     return ImportFromStream(new MemoryStream(byteArray));
104 }
105
106 /// <summary>
107 /// Export a given shape list into an svg file
108 /// </summary>
109 /// <param name="shapes">Shape list</param>
110 /// <param name="clientSize">Window size</param>
111 /// <param name="path">File path</param>
112 static public void ExportToFile(List<Shape> shapes, Vector2i ←
113   clientSize, string path)
114 {
115     using (FileStream fs = new FileStream(path, FileMode.Create))
116     {
117         byte[] bytes = Encoding.ASCII.GetBytes($@"<?xml version="1.0" encoding="utf-8"?>
118 <svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="{clientSize.X}" height="{clientSize.Y}">" + ←
119 Environment.NewLine);
120
121         for (int i = 0; i < bytes.Length; i++)
122             fs.WriteByte(bytes[i]);
123
124         foreach (Shape shape in shapes)
125         {
126             bytes = Encoding.ASCII.GetBytes(shape.ToString() + ←
127               Environment.NewLine);
128         }
129     }
130 }

```

```

120             for (int i = 0; i < bytes.Length; i++)
121                 fs.WriteByte(bytes[i]);
122         }
123         bytes = Encoding.ASCII.GetBytes("</svg>");
124         for (int i = 0; i < bytes.Length; i++)
125             fs.WriteByte(bytes[i]);
126     }
127 }
128
129 /// <summary>
130 /// Convert from a string to an Color4 value
131 /// </summary>
132 /// <param name="hexadecimal">hexadecimal string</param>
133 /// <returns>Color4 value</returns>
134 static public Color4 HexadecimalToRGB(string hexadecimal)
135 {
136     float r = Convert.ToInt32(hexadecimal.Substring(1, 2), 16) ←
137         / 255f;
138     float g = Convert.ToInt32(hexadecimal.Substring(3, 2), 16) ←
139         / 255f;
140     float b = Convert.ToInt32(hexadecimal.Substring(5, 2), 16) ←
141         / 255f;
142     float a = Convert.ToInt32(hexadecimal.Substring(7, 2), 16) ←
143         / 255f;
144     return new Color4(r, g, b, a);
145 }
146 }
147 }

```

Listing 1.3 – ./source/csharp/Importer.cs

1.1.4 Timeline.cs

```

1  using FreeFrame.Components.Shapes;
2  using ImGuiNET;
3  using OpenTK.Mathematics;
4  using OpenTK.Windowing.Common;
5  using System;
6  using System.Collections.Generic;
7  using System.Linq;
8  using System.Text;
9  using System.Threading.Tasks;
10
11 namespace FreeFrame
12 {
13     public class Timeline
14     {
15         public const int DEFAULT_FPS = 24;
16         public const int MIN_FPS = 1;
17         public const int MAX_FPS = 120;
18         public const int MIN_TIMELINE = 1;
19         public const int MAX_TIMELINE = 100;
20
21         private int _timelineIndex;
22         private int _fps;
23         private bool _isPlaying;
24
25         private double _secondsElapsed;
26
27         private SortedDictionary<int, List<Shape>> _sortedTimeline;
28
29         public SortedDictionary<int, List<Shape>> SortedTimeline { get ←
30             => _sortedTimeline; set => _sortedTimeline = value; }
31         public int TimelineIndex
32         {
33             get => _timelineIndex;
34             set
35             {
36                 if (value > MAX_TIMELINE) // Avoid blinking index

```

```

36             value -= MAX_TIMELINE;
37
38         _timelineIndex = Math.Clamp(value, MIN_TIMELINE, ←
39                                     MAX_TIMELINE);
40     }
41     public int Fps { get => _fps; set => _fps = value; }
42     public bool IsPlaying { get => _isPlaying; set => _isPlaying = ←
43                           value; }
44
45     public Timeline()
46     {
47         TimelineIndex = MIN_TIMELINE;
48         Fps = DEFAULT_FPS;
49         IsPlaying = false;
50         _secondsElapsed = 0;
51         SortedTimeline = new SortedDictionary<int, List<Shape>>();
52     }
53
54     /// <summary>
55     /// If IsPlaying is True, render the timeline in loop
56     /// </summary>
57     /// <param name="e"></param>
58     /// <param name="shapes"></param>
59     public void OnRenderFrame(FrameEventArgs e, Window window)
60     {
61         if (IsPlaying)
62         {
63             double frameDuration = 1.0 / Fps;
64             _secondsElapsed += e.Time;
65             if (_secondsElapsed >= frameDuration)
66             {
67                 while (_secondsElapsed >= frameDuration)
68                 {
69                     _secondsElapsed -= frameDuration;
70                     TimelineIndex++;
71                 }
72             }
73             RenderInterpolation(window.Shapes);
74             window.ResetSelection();
75         }
76     }
77     /// <summary>
78     /// Render timeline
79     /// </summary>
80     /// <param name="shapes">Shapes available</param>
81     public void RenderInterpolation(List<Shape> shapes)
82     {
83         foreach (Shape shape in shapes)
84         {
85             if (SortedTimeline.ContainsKey(TimelineIndex) == true &←
86                 && SortedTimeline[TimelineIndex] != null && ←
87                 SortedTimeline[TimelineIndex].Any(x => x.Id == ←
88                 shape.Id))
89             {
90                 // Draw the current one in this list
91                 Shape sibling = ←
92                     SortedTimeline[TimelineIndex].Find(x => x.Id == ←
93                     shape.Id)!;
94
95                 shape.X = sibling.X;
96                 shape.Y = sibling.Y;
97                 shape.Width = sibling.Width;
98                 shape.Height = sibling.Height;
99                 shape.Angle = sibling.Angle;

```

```

100     if (SortedTimeline.Any(i => i.Value.Any(j => j.Id == shape.Id))) // If exist somewhere else but not here
101     {
102         // Retrieve all the keyframes
103         int[] keys = SortedTimeline.Where(pair =>
104             pair.Value.Any(x => x.Id == shape.Id)).Select(pair =>
105                 pair.Key).ToArray(); // Key key everywhere it exist
106
107         (int first, int second) nearest = (int.MaxValue, int.MaxValue);
108
109         nearest.first = 0;
110         nearest.second = 0;
111
112         int timelineIndex = TimelineIndex;
113
113         if (timelineIndex >= keys[keys.Length - 1]) // Handle max value
114         {
115             nearest.first = keys[keys.Length - 1];
116             nearest.second = keys[keys.Length - 1];
117         }
118         else if (timelineIndex <= keys[0]) // Handle min value
119         {
120             nearest.first = keys[0];
121             nearest.second = keys[0];
122         }
123         else
124         {
125             // Retrieve the nearest keyframe to the current index in the timeline
126             for (int i = 0; i < keys.Length; i++)
127             {
128                 if (keys[i] >= timelineIndex)
129                 {
130                     nearest.second = keys[i];
131                     if (i - 1 >= 0) // If second possible
132                         nearest.first = keys[i - 1];
133                     break;
134                 }
135             }
136             Shape first = SortedTimeline[nearest.first].Find(x => x.Id == shape.Id)!;
137             Shape second = SortedTimeline[nearest.second].Find(x => x.Id == shape.Id)!;
138
139             // Interpolate each properties
140             shape.X = LinearInterpolate(timelineIndex, nearest.first, nearest.second, first.X, second.X);
141             shape.Y = LinearInterpolate(timelineIndex, nearest.first, nearest.second, first.Y, second.Y);
142             shape.Width = LinearInterpolate(timelineIndex, nearest.first, nearest.second, first.Width, second.Width);
143             shape.Height = LinearInterpolate(timelineIndex, nearest.first, nearest.second, first.Height, second.Height);
144             shape.Angle = LinearInterpolate(timelineIndex, nearest.first, nearest.second, first.Angle, second.Angle);
145             shape.CornerRadius = LinearInterpolate(timelineIndex, nearest.first, nearest.second, first.CornerRadius, second.CornerRadius);

```

```

146                     first.CornerRadius, second.CornerRadius);
147                     shape.Color = LinearInterpolate(timelineIndex, ←
148                         nearest.first, nearest.second, first.Color, ←
149                         second.Color);
150             }
151         }
152     }
153 }
154 }
155 }
156 }
157 }
158 }
159 /// <summary>
160 /// Update the given shape found in the timeline
161 /// </summary>
162 /// <param name="shape">Shape that need to be updated in the ←
163 /// timeline</param>
164 public void UpdateShapeInTimeline(Shape shape)
165 {
166     if (SortedTimeline.ContainsKey(TimelineIndex) == true && ←
167         SortedTimeline[TimelineIndex] != null)
168     {
169         Shape? existingShape = ←
170             SortedTimeline[TimelineIndex].Find(x => x.Id == ←
171                 shape.Id);
172         if (existingShape != null) // If the shape really exist ←
173             in the timeline, update it
174         {
175             existingShape.X = shape.X;
176             existingShape.Y = shape.Y;
177             existingShape.Width = shape.Width;
178             existingShape.Height = shape.Height;
179             existingShape.Angle = shape.Angle;
180             existingShape.CornerRadius = shape.CornerRadius;
181             existingShape.Color = shape.Color;
182         }
183     }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 public int LinearInterpolate(int x, int x1, int x2, int y1, int ←
192     y2)
193 {
194     float y = y1 + (x - x1) * ((y2 - y1) / ((x2 - x1) == 0 ? 1 ←
195         : (float)(x2 - x1)));
196     if (y1 <= y2)
197         return (int)Math.Clamp(y, y1, y2);
198     else
199         return (int)Math.Clamp(y, y2, y1);
200 }
201 }
202 }
203 }
204 }
205 }
206 }

```

```

207     /// <param name="y2"></param>
208     /// <returns></returns>
209     public Color4 LinearInterpolate(int x, int x1, int x2, Color4 ←
210         y1, Color4 y2)
211     {
212         int r = LinearInterpolate(x, x1, x2, (int)(y1.R * 255), ←
213             (int)(y2.R * 255));
214         int g = LinearInterpolate(x, x1, x2, (int)(y1.G * 255), ←
215             (int)(y2.G * 255));
216         int b = LinearInterpolate(x, x1, x2, (int)(y1.B * 255), ←
217             (int)(y2.B * 255));
218         int a = LinearInterpolate(x, x1, x2, (int)(y1.A * 255), ←
219             (int)(y2.A * 255));
220         return new Color4(r / 255f, g / 255f, b / 255f, a / 255f);
221     }
222
223     /// <summary>
224     /// Remove given shape in the timeline
225     /// </summary>
226     /// <param name="shapeToRemove">The shape to remove</param>
227     public void RemoveElementInTimeline(Shape shapeToRemove)
228     {
229         List<int> emptyKeyframes = new();
230         foreach (KeyValuePair<int, List<Shape>> keyframe in ←
231             SortedTimeline) // Remove element in the timeline
232         {
233             foreach (Shape shape in keyframe.Value)
234             {
235                 if (shape.Id == shapeToRemove.Id)
236                 {
237                     shape.DeleteObjects();
238                     keyframe.Value.Remove(shape);
239                     break; // Because we know that only one shape ←
240                           id is in the current list
241                 }
242             }
243             if (keyframe.Value.Count == 0) // Remove shapes in ←
244               timeline
245               emptyKeyframes.Add(keyframe.Key);
246         }
247         foreach (int id in emptyKeyframes) // Remove empty keyframes
248             SortedTimeline.Remove(id);
249     }
250
251     /// <summary>
252     /// Empty the timeline
253     /// </summary>
254     public void ResetTimeline()
255     {
256         foreach (KeyValuePair<int, List<Shape>> keyframe in ←
257             SortedTimeline) // Remove element in the timeline
258         {
259             foreach (Shape shape in keyframe.Value)
260                 shape.DeleteObjects();
261             keyframe.Value.Clear();
262         }
263         SortedTimeline.Clear();
264     }
265     }
266 }
```

Listing 1.4 – ./source/csharp/Timeline.cs

1.1.5 Helper.cs

```

1  iz&using OpenTK.Graphics.OpenGL4;
2  using System.Diagnostics;
3  using System.Runtime.InteropServices;
4
```

```

5 namespace FreeFrame
6 {
7     static class Helper
8     {
9         private static DebugProc _debugProcCallback = DebugCallback;
10        static public void CheckErrors()
11        {
12            ErrorCode errorCode = GL.GetError();
13
14            while (errorCode != ErrorCode.NoError)
15            {
16                Console.WriteLine(errorCode.ToString());
17                errorCode = GL.GetError();
18            }
19        }
20        [DebuggerStepThrough]
21        static private void DebugCallback(DebugSource source, DebugType ←
22            type, int id, DebugSeverity severity, int length, IntPtr ←
23            message, IntPtr userParam)
24        {
25            string messageString = Marshal.PtrToStringAnsi(message, ←
26                length); // Retrieve the string from the pointer
27
28            switch (severity)
29            {
30                case DebugSeverity.DontCare:
31                case DebugSeverity.DebugSeverityNotification:
32                    Console.ForegroundColor = ConsoleColor.White;
33                    break;
34                case DebugSeverity.DebugSeverityHigh:
35                    Console.ForegroundColor = ConsoleColor.Red;
36                    break;
37                case DebugSeverity.DebugSeverityMedium:
38                    Console.ForegroundColor = ConsoleColor.DarkYellow;
39                    break;
40                case DebugSeverity.DebugSeverityLow:
41                    Console.ForegroundColor = ConsoleColor.Yellow;
42                    break;
43                default:
44                    break;
45            }
46            Console.WriteLine($"{severity} {type} {messageString}");
47            Console.ResetColor();
48
49            if (type == DebugType.DebugTypeError)
50                throw new Exception("OpenGL error");
51
52        /// <summary>
53        /// Enable the debug mode
54        /// </summary>
55        static public void EnableDebugMode()
56        {
57            GL.DebugMessageCallback(_debugProcCallback, IntPtr.Zero);
58            GL.Enable(EnableCap.DebugOutput);
59            GL.Enable(EnableCap.DebugOutputSynchronous);
60        }
61    }
62}

```

Listing 1.5 – ./source/csharp/Helper.cs

1.1.6 IDrawable.cs

```

1  #using OpenTK.Mathematics;
2
3 namespace FreeFrame
4 {
5     public interface IDrawable
6     {

```

```

7     /// <summary>
8     /// Trigger draw element through OpenGL context
9     /// </summary>
10    /// <param name="clientSize">Window size</param>
11    public void Draw(Vector2i clientSize);
12
13    /// <summary>
14    /// Deletes the objects saved in OpenGL context
15    /// </summary>
16    public void DeleteObjects();
17}
18}

```

Listing 1.6 – ./source/csharp/IDrawable.cs

1.1.7 Shape.cs

```

1  using OpenTK.Mathematics;
2
3  namespace FreeFrame.Components.Shapes
4  {
5      public abstract class Shape : IDrawable
6      {
7          #region Default values
8          public const string DefaultColor = "#000000FF";
9          #endregion
10
11         #region Common Geometry Properties
12         private int _x, _y, _width, _height, _angle, _cornerRadius;
13         private Color4 _color;
14         Guid _id;
15         bool _isMoveable = true;
16         bool _isResizeable = true;
17         bool _isAngleChangeable = true;
18         bool _isCornerRadiusChangeable = true;
19         #endregion
20
21         private List<Renderer> renderers;
22         public int X { get => _x; set => _x = value; }
23         public int Y { get => _y; set => _y = value; }
24         public int Width { get => _width; set => _width = Math.Max(0, ←
25             value); }
26         public int Height { get => _height; set => _height = ←
27             Math.Max(0, value); }
28         public Color4 Color { get => _color; set => _color = value; }
29         public int Angle { get => _angle; set => _angle = value; }
30         public bool IsMoveable { get => _isMoveable; set => _isMoveable ←
31             = value; }
32         public bool IsResizeable { get => _isResizeable; set => ←
33             _isResizeable = value; }
34         public bool IsAngleChangeable { get => _isAngleChangeable; set ←
35             => _isAngleChangeable = value; }
36         public bool IsCornerRadiusChangeable { get => ←
37             _isCornerRadiusChangeable; set => _isCornerRadiusChangeable ←
38             = value; }
39         public int CornerRadius { get => _cornerRadius; set => ←
40             _cornerRadius = value; }
41         public Guid Id { get => _id; set => _id = value; }
42         public List<Renderer> Renderers { get => renderers; set => ←
43             renderers = value; }

44         /// <summary>
45         /// Retrieve the Id but shorter (only used for display)
46         /// </summary>
47         public string ShortId
48         {
49             get => Id.ToString().Substring(0, 6);
50         }
51         public Shape()
52     }
```

```

44    {
45        Renderers = new List<Renderer>();
46        Color = Color4.Black;
47        Id = Guid.NewGuid();
48    }
49
50    /// <summary>
51    /// Trigger draw element through OpenGL context
52    /// </summary>
53    /// <param name="clientSize">Window size</param>
54    public virtual void Draw(Vector2i clientSize)
55    {
56        foreach (Renderer render in Renderers)
57            render.Draw(clientSize, Color, this);
58    }
59
60    /// <summary>
61    /// Deletes the objects saved in OpenGL context
62    /// </summary>
63    public void DeleteObjects()
64    {
65        foreach (Renderer render in Renderers)
66            render.DeleteObjects();
67        Renderers.Clear();
68    }
69
70    public Shape ShallowCopy() => (Shape)MemberwiseClone();
71
72    public Shape DeepCopy()
73    {
74        Shape shape = (Shape)MemberwiseClone();
75        shape.Id = Guid.NewGuid();
76        shape.DeleteObjects();
77        shape.Renderers = new List<Renderer>();
78        shape.ImplementObject();
79        return shape;
80    }
81
82    /// <summary>
83    /// Return the vertices position in NDC format
84    /// </summary>
85    /// <returns>array of vertices position. x, y, x, y, ... ←
86    /// (clockwise)</returns>
87    public abstract float[] GetVertices();
88
89    /// <summary>
90    /// Return the indexes position of the triangles
91    /// </summary>
92    /// <returns>array of indexes</returns>
93    public abstract uint[] GetVerticesIndexes();
94
95    /// <summary>
96    /// Retrieve the points that made the shape detectable
97    /// </summary>
98    /// <returns>Position of all the points</returns>
99    public abstract List<Vector2i> GetSelectablePoints();
100
101   /// <summary>
102   /// Reset the renderers and create new ones (use when update ←
103   /// any properties of the shape)
104   /// </summary>
105   public abstract void ImplementObject();
106
107   /// <summary>
108   /// Move the current shape to the given position
109   /// </summary>
110   /// <param name="position">New position</param>
111   public abstract void Move(Vector2i position);
112
113   /// <summary>
114   /// Resize the current shape to the given size
115   /// </summary>

```

```

114     /// <param name="size">New size</param>
115     public abstract void Resize(Vector2i size);
116
117     /// <summary>
118     /// Retrieve the Shape in the SVG format
119     /// </summary>
120     /// <returns>string of the SVG format</returns>
121     public abstract override string ToString();
122
123     /// <summary>
124     /// Convert the given Color4 into a hexadecimal string
125     /// </summary>
126     /// <param name="color">Color to convert</param>
127     /// <returns>format: #XXXXXXXX</returns>
128     public static string ColorToHexadecimal(Color4 color)
129     {
130         int r, g, b, a;
131         r = (int)(color.R * 255);
132         g = (int)(color.G * 255);
133         b = (int)(color.B * 255);
134         a = (int)(color.A * 255);
135         return '#' + r.ToString("X2") + g.ToString("X2") + ←
136             b.ToString("X2") + a.ToString("X2");
137     }
138 }
```

Listing 1.7 – ./source/csharp/Shape.cs

1.1.8 Selector.cs

```

1  iż<using FreeFrame.Components.Shapes;
2  using OpenTK.Graphics.OpenGL4;
3  using OpenTK.Mathematics;
4
5  namespace FreeFrame
6  {
7      public class Selector : IDrawable
8      {
9          public struct Area
10         {
11             public int X;
12             public int Y;
13             public int Width;
14             public int Height;
15             public Area(int x, int y, int width, int height)
16             {
17                 X = x;
18                 Y = y;
19                 Width = width;
20                 Height = height;
21             }
22             public float[] ToFloatArray() => new float[] { X, Y, X + ←
23                 Width, Y, X + Width, Y + Height, X, Y + Height }; // ←
24                 Clockwise
25         }
26         public enum SelectorType
27         {
28             Edge,
29             Move,
30             Resize,
31             None
32         }
33
34         private List<(Renderer vao, Area hitbox, SelectorType type)> ←
35             _selectors;
36
37         public Selector()
```

```

36     {
37         _selectors = new List<(Renderer vao, Area hitbox, ←
38             SelectorType type)>();
39     }
40
41     /// <summary>
42     /// Select the shape to draw the selector with
43     /// </summary>
44     /// <param name="shape">Selected shape</param>
45     public void Select(Shape shape)
46     {
47         // Delete current Selector
48         DeleteObjects();
49
50         // Edge
51         Area hitbox = new Area
52         {
53             X = shape.X,
54             Y = shape.Y,
55             Width = shape.Width,
56             Height = shape.Height
57         };
58         _selectors.Add((new Renderer(hitbox.ToFloatArray(), new ←
59             uint[] { 0, 1, 2, 3 }, PrimitiveType.LineLoop), hitbox, ←
60             SelectorType.Edge)); // The edge is only a square empty
61
62         if (shape.IsMoveable)
63         {
64             // Move selector (top-left)
65             hitbox = new Area
66             {
67                 X = shape.X - 5,
68                 Y = shape.Y - 5,
69                 Width = 10,
70                 Height = 10
71             };
72             _selectors.Add((new Renderer(hitbox.ToFloatArray(), new ←
73                 uint[] { 0, 1, 2, 0, 2, 3 }, ←
74                 PrimitiveType.Triangles), hitbox, ←
75                 SelectorType.Move)); // The corner are filled so ←
76                 it's two triangles
77         }
78
79         if (shape.IsResizeable)
80         {
81             // Resize selector (bottom-right)
82             hitbox = new Area
83             {
84                 X = shape.X + shape.Width - 5,
85                 Y = shape.Y + shape.Height - 5,
86                 Width = 10,
87                 Height = 10
88             };
89             _selectors.Add((new Renderer(hitbox.ToFloatArray(), new ←
90                 uint[] { 0, 1, 2, 0, 2, 3 }, ←
91                 PrimitiveType.Triangles), hitbox, ←
92                 SelectorType.Resize)); // The corner are filled so ←
93                 it's two triangles
94         }
95     }
96
97     /// <summary>
98     /// Trigger draw element through OpenGL context
99     /// </summary>
100    /// <param name="clientSize">Window size</param>
101    public void Draw(Vector2i clientSize)
102    {
103        GL.Enable(EnableCap.LineSmooth);
104        GL.LineWidth(3.0f);
105    }

```

```

96         foreach ((Renderer vao, Area _, SelectorType type) selector ←
97             in _selectors)
98         {
99             if (selector.type == SelectorType.Edge) // The color is ←
100                not the same for the edge and the corners
101                selector.vao.Draw(clientSize, new Color4(0, 125, ←
102                    200, 255));
103            else
104                selector.vao.Draw(clientSize, new Color4(0, 125, ←
105                    255, 255));
106        }
107
108        GL.Disable(EnableCap.LineSmooth);
109    }
110
111    /// <summary>
112    /// Return true if the given mouse position is in the selector ←
113    hitbox
114    /// </summary>
115    /// <param name="mousePosition">Mouse position</param>
116    /// <returns>true if touching the selector (and give the ←
117    selector type), false otherwise (and null) </returns>
118    public (bool, SelectorType?) HitBox(Vector2i mousePosition)
119    {
120        if (_selectors.Count > 0)
121            foreach ((Renderer _, Area hitbox, SelectorType type) ←
122                selector in _selectors)
123                if (selector.hitbox.X < mousePosition.X && ←
124                    selector.hitbox.X + selector.hitbox.Width > ←
125                    mousePosition.X && selector.hitbox.Y < ←
126                    mousePosition.Y && selector.hitbox.Y + ←
127                    selector.hitbox.Height > mousePosition.Y)
128                    return (true, selector.type);
129        return (false, null);
130    }
131
132    /// <summary>
133    /// Delete all the renderers for each selector
134    /// </summary>
135    public void DeleteObjects()
136    {
137        _selectors.ForEach(i => i.vao.DeleteObjects());
138        _selectors.Clear();
139    }
140}

```

Listing 1.8 – ./source/csharp/Selector.cs

1.1.9 Renderer.cs

```

1  iz&using FreeFrame.Components.Shapes;
2  using OpenTK.Graphics.OpenGL4;
3  using OpenTK.Mathematics;
4
5  namespace FreeFrame
6  {
7      public class Renderer
8      {
9          private int _vertexBufferObjectID;
10         private int _vertexArrayObjectID;
11         private int _indexBufferObjectID;
12         private int _indexCount;
13         private PrimitiveType _primitiveType;
14         private Shader _shader;
15
16         // VBO
17         public int VertexBufferObjectID { get => _vertexBufferObjectID; ←
18             private set => _vertexBufferObjectID = value; }

```

```

19 // VAO
20 public int VertexArrayObjectID { get => _vertexArrayObjectID; ←
21     private set => _vertexArrayObjectID = value; }
22
23 // IBO
24 public int IndexBufferObjectID { get => _indexBufferObjectID; ←
25     private set => _indexBufferObjectID = value; }
26
27 public Renderer() : this(PrimitiveType.Triangles) { }
28 public Renderer(PrimitiveType primitiveType)
29 {
30     _primitiveType = primitiveType;
31     VertexArrayObjectID = GL.GenVertexArray();
32     VertexBufferObjectID = GL.GenBuffer();
33     IndexBufferObjectID = GL.GenBuffer();
34     _shader = new Shader("Shaders/shader.vert", ←
35         "Shaders/shader.frag");
36 }
37
38 public Renderer(float[] vertices, uint[] indexes, PrimitiveType ←
39     primitiveType) : this(primitiveType)
40 {
41     // Create all the objects and index for OpenGL and send data
42     ImplementObjects(vertices, indexes);
43 }
44
45 public Renderer(float[] vertices, uint[] indexes, PrimitiveType ←
46     primitiveType, Shape shape) : this(primitiveType)
47 {
48     Type type = shape.GetType();
49     if (type == typeof(SVGCircle)) // Shader depend on the shape
50         _shader = new Shader("Shaders/shader.vert", ←
51             "Shaders/circle.frag");
52     else if (type == typeof(SVGRectangle))
53         _shader = new Shader("Shaders/shader.vert", ←
54             "Shaders/rectangle.frag");
55
56     // Create all the objects and index for OpenGL and send data
57     ImplementObjects(vertices, indexes);
58 }
59
60 /// <summary>
61 /// Call the Draw method of OpenGL
62 /// </summary>
63 /// <param name="clientSize">Window size</param>
64 /// <param name="color">Element color</param>
65 public void Draw(Vector2i clientSize, Color4 color)
66 {
67     _shader.Use();
68
69     // Applied projection matrix
70     int uModelToNDC = ←
71         _shader.GetUniformLocation("u_Model_To_NDC");
72     Matrix4 matrix = Matrix4.CreateOrthographicOffCenter(0, ←
73         clientSize.X, clientSize.Y, 0, -1.0f, 1.0f);
74     _shader.SetUniformMat4(uModelToNDC, matrix);
75
76     // Applied common geometry color
77     int uColor = _shader.GetUniformLocation("u_Color");
78     _shader.SetUniformVec4(uColor, (Vector4)color);
79
80     // Applied window size
81     int uResolution = _shader.GetUniformLocation("u_Resolution");
82     _shader.SetUniformVec2(uResolution, (Vector2)clientSize);
83
84     // Applied transformation
85     int uTransformation = ←
86         _shader.GetUniformLocation("u_Transformation");
87     _shader.SetUniformMat4(uTransformation, Matrix4.Identity);
88
89     // Bind VAO
90     GL.BindVertexArray(VertexArrayObjectID);

```

```

81
82     // Draw
83     GL.DrawElements(_primitiveType, _indexCount, ←
84         DrawElementsType.UnsignedInt, 0);
85
86     /// <summary>
87     /// Call the Draw method of OpenGL
88     /// </summary>
89     /// <param name="clientSize">Window size</param>
90     /// <param name="color">Element color</param>
91     /// <param name="shape">Shape type</param>
92     public void Draw(Vector2i clientSize, Color4 color, Shape shape)
93     {
94         _shader.Use();
95
96         // Applied projection matrix
97         int uModelToNDC = ←
98             _shader.GetUniformLocation("u_Model_To_NDC");
99         Matrix4 matrix = Matrix4.CreateOrthographicOffCenter(0, ←
100             clientSize.X, clientSize.Y, 0, -1.0f, 1.0f);
101         _shader.SetUniformMat4(uModelToNDC, matrix);
102
103         // Applied common geometry color
104         int uColor = _shader.GetUniformLocation("u_Color");
105         _shader.SetUniformVec4(uColor, (Vector4)color);
106
107         // Applied window size
108         int uResolution = _shader.GetUniformLocation("u_Resolution");
109         _shader.SetUniformVec2(uResolution, (Vector2)clientSize);
110
111         // Applied transformation
112         int uTransformation = ←
113             _shader.GetUniformLocation("u_Transformation");
114         Matrix4 rotation = ←
115             Matrix4.CreateRotationZ(MathHelper.DegreesToRadians(shape.Angle));
116         _shader.SetUniformMat4(uTransformation, rotation);
117
118         // Applied corner radius
119         int uRadius = _shader.GetUniformLocation("u_Radius");
120         _shader.SetUniformFloat(uRadius, shape.CornerRadius);
121
122         // Applied width
123         int uSize = _shader.GetUniformLocation("u_Size");
124         _shader.SetUniformVec2(uSize, new Vector2(shape.Width, ←
125             shape.Height));
126
127         // Bind VAO
128         GL.BindVertexArray(VertexArrayObjectID);
129
130         // Draw
131         GL.Enable(EnableCap.Blend);
132         GL.DrawElements(_primitiveType, _indexCount, ←
133             DrawElementsType.UnsignedInt, 0);
134         GL.Disable(EnableCap.Blend);
135     }
136     public void ImplementObjects(float[] vertices, uint[] indexes)
137     {
138         // VAO
139         GL.BindVertexArray(VertexArrayObjectID);
140
141         string label = $"VAO_{GetType().Name}:";
142         GL.ObjectLabel(ObjectLabelIdentifier.VertexArray, ←
143             VertexArrayObjectID, label.Length, label);
144
145         // VBO
146         GL.BindBuffer(BufferTarget.ArrayBuffer, VertexBufferObjectID);

```

```

144         GL.BufferData(BufferTarget.ArrayBuffer, vertices.Length * ←
145             sizeof(float), vertices, BufferUsageHint.StaticDraw);
146
147         label = $"VBO_{GetType().Name}:";
148         GL.ObjectLabel(ObjectLabelIdentifier.Buffer, ←
149             VertexBufferObjectID, label.Length, label);
150
151         // IBO
152         GL.BindBuffer(BufferTarget.ElementArrayBuffer, ←
153             IndexBufferObjectID);
154         GL.BufferData(BufferTarget.ElementArrayBuffer, ←
155             indexes.Length * sizeof(uint), indexes, ←
156             BufferUsageHint.StaticDraw);
157
157         label = $"IBO_{GetType().Name}:";
158         GL.ObjectLabel(ObjectLabelIdentifier.Buffer, ←
159             IndexBufferObjectID, label.Length, label);
160
161         // Link Attributes
162         GL.VertexAttribPointer(0, 2, VertexAttribPointerType.Float, ←
163             false, 2 * sizeof(float), 0); // x, y;
164         GL.EnableVertexAttribArray(0);
165
166         _indexCount = indexes.Length;
167     }
168
169     /// <summary>
170     /// Delete all the buffer objects
171     /// </summary>
172     public void DeleteObjects()
173     {
174         GL.DeleteBuffer(VertexBufferObjectID);
175         GL.DeleteBuffer(IndexBufferObjectID);
176         GL.DeleteVertexArray(VertexArrayObjectID);
177     }
178 }
```

Listing 1.9 – ./source/csharp/Renderer.cs

1.1.10 SVGLine.cs

```

1  iż<using OpenTK.Graphics.OpenGL4;
2  using OpenTK.Mathematics;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8  using System.Xml;
9
10 namespace FreeFrame.Components.Shapes
11 {
12     public class SVGLine : Shape
13     {
14         #region Default values
15         const int DefaultX1 = 0;
16         const int DefaultY1 = 0;
17         const int DefaultX2 = 0;
18         const int DefaultY2 = 0;
19         #endregion
20
21         public SVGLine(XmlReader reader) : this(
22             Convert.ToInt32(reader["x1"]),
23             Convert.ToInt32(reader["y1"]),
24             Convert.ToInt32(reader["x2"]),
25             Convert.ToInt32(reader["y2"]),
26             Convert.ToString(reader["fill"]) == null ? DefaultColor : ←
27             reader["fill"]))
28     }
```

```

28 }
29 public SVGLine() : this(DefaultX1, DefaultY1, DefaultX2, ←
30     DefaultY2, DefaultColor) { }
31 public SVGLine(int x1, int y1, int x2, int y2, string color)
32 {
33     IsCornerRadiusChangeable = false;
34
35     X = x1;
36     Y = y1;
37     Width = x2 - X;
38     Height = y2 - Y;
39     Color = Importer.HexadecimalToRGB(color);
40
41     ImplementObject();
42 }
43
44 /// <summary>
45 /// Return the vertices position in NDC format
46 /// </summary>
47 /// <returns>array of vertices position. x, y, x, y, ... ←
48 /// (clockwise)</returns>
49 public override float[] GetVertices() => new float[] { X, Y, X ←
50     + Width, Y + Height }; // x, y, x, y, x, y, ... (clockwise)
51
52 /// <summary>
53 /// Return the indexes position of the lines
54 /// </summary>
55 /// <returns>array of indexes</returns>
56 public override uint[] GetVerticesIndexes() => new uint[] { 0, ←
57     1 };
58
59 /// <summary>
60 /// Reset the renderers and create new ones (use when update ←
61 /// any properties of the shape)
62 /// </summary>
63 public override void ImplementObject()
64 {
65     foreach (Renderer vao in Renderers)
66         vao.DeleteObjects();
67     Renderers.Clear();
68
69     Renderers.Add(new Renderer(GetVertices(), ←
70                 GetVerticesIndexes(), PrimitiveType.Lines, this ));
71 }
72
73 /// <summary>
74 /// Retrieve the points that made the shape detectable
75 /// </summary>
76 /// <returns>Position of all the points</returns>
77 public override List<Vector2i> GetSelectablePoints()
78 {
79     List<Vector2i> points = new();
80     points.Add(new Vector2i(X, Y));
81     points.Add(new Vector2i(Width + X, Height + Y));
82     return points;
83 }
84
85 /// <summary>
86 /// Move the current shape to the given position
87 /// </summary>
88 /// <param name="position">New position</param>
89 public override void Move(Vector2i position)
90 {
91     X = position.X;
92     Y = position.Y;
93
94     ImplementObject();
95 }
96
97 /// <summary>
98 /// Resize the current shape to the given size
99 /// </summary>

```

```

94     /// <param name="size">New size</param>
95     public override void Resize(Vector2i size)
96     {
97         Width = size.X;
98         Height = size.Y;
99
100        ImplementObject();
101    }
102
103    /// <summary>
104    /// Retrieve the Shape in the SVG format
105    /// </summary>
106    /// <returns>string of the SVG format</returns>
107    public override string ToString() => $"<line x1=\"{X}\" ↵
108        y1=\"{Y}\" x2=\"{Width+X}\" y2=\"{Height+Y}\" ↵
109        fill=\"{ColorToHexadecimal(Color)}\"/>";
}

```

Listing 1.10 – ./source/csharp/SVGLine.cs

1.1.11 SVGCircle.cs

```

1  #using OpenTK.Graphics.OpenGL4;
2  using OpenTK.Mathematics;
3  using System;
4  using System.Collections.Generic;
5  using System.Drawing;
6  using System.Text;
7  using System.Xml;
8
9  namespace FreeFrame.Components.Shapes
10 {
11     public class SVGCircle : Shape
12     {
13         #region Default values
14         const int DefaultR = 0;
15         const int DefaultCY = 0;
16         const int DefaultCX = 0;
17         #endregion
18         public SVGCircle(XmlReader reader) : this(
19             Convert.ToInt32(reader["r"]),
20             Convert.ToInt32(reader["cx"]),
21             Convert.ToInt32(reader["cy"]),
22             Convert.ToString(reader["fill"]) == null ? DefaultColor : ←
23             reader["fill"])
24     { }
25     public SVGCircle() : this(DefaultR, DefaultCX, DefaultCY, ←
26         "#000000FF") { }
27     public SVGCircle(int r, int cx, int cy, string color)
28     {
29         IsCornerRadiusChangeable = false;
30         X = cx - r;
31         Y = cy - r;
32         Height = r * 2;
33         Width = Height;
34         Color = Importer.HexadecimalToRGB(color);
35
36         ImplementObject();
37     }
38
39     /// <summary>
40     /// Return the vertices position in NDC format
41     /// </summary>
42     /// <returns>array of vertices position. x, y, x, y, ... ←
43     /// (clockwise)</returns>
44     public override float[] GetVertices() => new float[] { X, Y, X ←
45         + Width, Y, X + Width, Y + Height, X, Y + Height }; // x, y, ←
46         x, y, x, y, ... (clockwise)

```

```

42
43     /// <summary>
44     /// Return the indexes position of the triangles
45     /// </summary>
46     /// <returns>array of indexes</returns>
47     public override uint[] GetVerticesIndexes() => new uint[] { 0, ←
48         1, 2, 0, 2, 3 };
49
50     /// <summary>
51     /// Reset the renderers and create new ones (use when update ←
52     /// any properties of the shape)
53     /// </summary>
54     public override void ImplementObject()
55     {
56         foreach (Renderer render in Renderers)
57             render.DeleteObjects();
58         Renderers.Clear();
59
60         Renderers.Add(new Renderer(GetVertices(), ←
61             GetVerticesIndexes(), PrimitiveType.Triangles, this));
62     }
63
64     /// <summary>
65     /// Retrieve the points that made the shape detectable
66     /// </summary>
67     /// <returns>Position of all the points</returns>
68     public override List<Vector2i> GetSelectablePoints()
69     {
70         List<Vector2i> points = new();
71         points.Add(new Vector2i(X, Y));
72         points.Add(new Vector2i(X + Width, Y));
73         points.Add(new Vector2i(X + Width, Y + Height));
74         points.Add(new Vector2i(X, Y + Height));
75         return points;
76     }
77
78     /// <summary>
79     /// Move the current shape to the given position
80     /// </summary>
81     /// <param name="position">New position</param>
82     public override void Move(Vector2i position)
83     {
84         X = position.X;
85         Y = position.Y;
86         ImplementObject();
87     }
88
89     /// <summary>
90     /// Resize the current shape to the given size
91     /// </summary>
92     /// <param name="size">New size</param>
93     public override void Resize(Vector2i size)
94     {
95         Width = size.X;
96         Height = size.Y;
97         ImplementObject();
98     }
99
100    /// <summary>
101    /// Retrieve the Shape in the SVG format
102    /// </summary>
103    /// <returns>string of the SVG format</returns>
104    public override string ToString() => $"<circle cx=\"{X+Width/2}\" cy=\"{Y+Height/2}\" r=\"{Width/2}\" fill=\"{ColorToHexadecimal(Color)}\"/>";
105 }
```

Listing 1.11 – ./source/csharp/SVGCircle.cs

1.1.12 SVGRectangle.cs

```
1  işfusing OpenTK.Graphics.OpenGL4;
2  using OpenTK.Mathematics;
3  using System.Xml;
4
5  namespace FreeFrame.Components.Shapes
6  {
7      public class SVGRectangle : Shape
8      {
9          #region Default values
10         const int DefaultX = 0;
11         const int DefaultY = 0;
12         const int DefaultWidth = 0;
13         const int DefaultHeight = 0;
14         const int DefaultRX = 0;
15         const int DefaultRY = 0;
16     #endregion
17
18     public SVGRectangle(XmlReader reader) : this(
19         Convert.ToInt32(reader["width"]),
20         Convert.ToInt32(reader["height"]),
21         Convert.ToInt32(reader["x"]),
22         Convert.ToInt32(reader["y"]),
23         Convert.ToInt32(reader["rx"]),
24         Convert.ToInt32(reader["ry"]),
25         Convert.ToString(reader["fill"]) == null ? DefaultColor : ←
26             reader["fill"]))
27     {
28
29     public SVGRectangle() : this(DefaultWidth, DefaultHeight, ←
30         DefaultX, DefaultY) { }
31     public SVGRectangle(int width, int height) : this(width, ←
32         height, DefaultX, DefaultY) { }
33     public SVGRectangle(int width, int height, int x, int y) : ←
34         this(width, height, x, y, DefaultRX, DefaultRY, ←
35         DefaultColor) { }
36     public SVGRectangle(int width, int height, int x, int y, int ←
37         rx, int ry, string color)
38     {
39         X = x;
40         Y = y;
41         Width = width;
42         Height = height;
43         CornerRadius = Math.Max(rx, ry);
44         Color = Importer.HexadecimalToRGB(color);
45
46         ImplementObject();
47     }
48
49     /// <summary>
50     /// Should return the vertices position in NDC format
51     /// </summary>
52     /// <returns>array of vertices position. x, y, x, y, ... ←
53     /// (clockwise)</returns>
54     public override float[] GetVertices() => new float[] { X, Y, X ←
55         + Width, Y, X + Width, Y + Height, X, Y + Height }; // x, y, ←
56         x, y, x, y, ... (clockwise)
57
58     /// <summary>
59     /// Should return the indexes position of the triangles
60     /// </summary>
61     /// <returns>array of indexes</returns>
62     public override uint[] GetVerticesIndexes() => new uint[] { 0, ←
63         1, 2, 0, 2, 3 };
64
65     /// <summary>
66     /// Reset the renderers and create new ones (use when update ←
67     /// any properties of the shape)
68     /// </summary>
69     public override void ImplementObject()
```

```

59     {
60         foreach (Renderer render in Renderers)
61             render.DeleteObjects();
62         Renderers.Clear();
63
64         Renderers.Add(new Renderer(GetVertices(), ←
65                         GetVerticesIndexes(), PrimitiveType.Triangles, this));
66     }
67
68     /// <summary>
69     /// Retrieve the points that made the shape detectable
70     /// </summary>
71     /// <returns>Position of all the points</returns>
72     public override List<Vector2i> GetSelectablePoints()
73     {
74         List<Vector2i> points = new();
75         points.Add(new Vector2i(X, Y));
76         points.Add(new Vector2i(X + Width, Y));
77         points.Add(new Vector2i(X + Width, Y + Height));
78         points.Add(new Vector2i(X, Y + Height));
79         return points;
80     }
81
82     /// <summary>
83     /// Move the current shape to the given position
84     /// </summary>
85     /// <param name="position">New position</param>
86     public override void Move(Vector2i position)
87     {
88         X = position.X;
89         Y = position.Y;
90         ImplementObject();
91     }
92
93     /// <summary>
94     /// Resize the current shape to the given size
95     /// </summary>
96     /// <param name="size">New size</param>
97     public override void Resize(Vector2i size)
98     {
99         Width = size.X;
100        Height = size.Y;
101        ImplementObject();
102    }
103
104    /// <summary>
105    /// Retrieve the Shape in the SVG format
106    /// </summary>
107    /// <returns>string of the SVG format</returns>
108    public override string ToString()
109    {
110        return $"<rect x=\"{X}\" y=\"{Y}\" width=\"{Width}\" ←
111                         height=\"{Height}\" rx=\"{CornerRadius}\" ←
112                         ry=\"{CornerRadius}\" ←
113                         fill=\"{ColorToHexadecimal(Color)}\"/>";
114    }
115}
116}

```

Listing 1.12 – ./source/csharp/SVGRectangle.cs

1.1.13 SVGPolygon.cs

```

1  izdeusing System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Xml;
5  using System.Text.RegularExpressions;
6  using OpenTK.Mathematics;

```

```

7  using OpenTK.Graphics.OpenGL4;
8
9  namespace FreeFrame.Components.Shapes
10 {
11     public class SVGPolygon : Shape
12     {
13         readonly Regex _pointsAttributeRegex = new(@"^*(\d+)^*,^*(\d+)^*");
14         // ↵
15         // https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/points
16
17         #region Geometry properties
18         List<Vector2i> _points = new();
19
20         public List<Vector2i> Points { get => _points; set => _points = value; }
21         #endregion
22
23         public SVGPolygon() { }
24         public SVGPolygon(XmlReader reader)
25         {
26             IsResizeable = false;
27             IsCornerRadiusChangeable = false;
28
29             string points = reader["points"] ?? throw new Exception("points not here");
30             MatchCollection matches = _pointsAttributeRegex.Matches(points); // Retrieve every points
31             foreach (Match match in matches)
32                 Points.Add((Convert.ToInt32(match.Groups[1].Value), Convert.ToInt32(match.Groups[2].Value)));
33
34             string color = reader["fill"] ?? DefaultColor;
35             Color = Importer.HexadecimalToRGB(color);
36
37             // Fill geometry values
38             X = Points.Min(i => i.X);
39             Y = Points.Min(i => i.Y);
40             Width = Points.Max(i => i.X) - Points.Min(i => i.X);
41             Height = Points.Max(i => i.Y) - Points.Min(i => i.Y);
42
43             // Made points relatives
44             for (int i = 0; i < Points.Count; i++)
45                 Points[i] = new Vector2i(Points[i].X - X, Points[i].Y - Y); // delta
46
47             if (Points.Count == 3)
48                 IsResizeable = true;
49
50             ImplementObject();
51         }
52         public SVGPolygon(int x, int y, int width, int height, string color)
53         {
54             IsCornerRadiusChangeable = false;
55
56             X = x;
57             Y = y;
58             Width = width;
59             Height = height;
60             Color = Importer.HexadecimalToRGB(color);
61
62             Points.Add(new Vector2i(X + Width / 2, Y));
63             Points.Add(new Vector2i(X, Y + Height));
64             Points.Add(new Vector2i(X + Width, Y + Height));
65
66             ImplementObject();
67         }
68
69         /// <summary>
69         /// Should return the vertices position in NDC format

```

```

70     /// </summary>
71     /// <returns>array of vertices position. x, y, x, y, ... ←
72     /// (clockwise)</returns>
73     public override float[] GetVertices()
74     {
75         if (Points.Count == 3) // Because a triangle is also base ←
76             on the size
77             return new float[] { X + Width / 2, Y, X, Y + Height, X ←
78                 + Width, Y + Height };
79         else // Any other polygon
80         {
81             List<float> vertices = new List<float>();
82             foreach (Vector2i point in Points)
83             {
84                 vertices.Add(X + point.X);
85                 vertices.Add(Y + point.Y);
86             }
87             return vertices.ToArray();
88         }
89     }
90
91     /// <summary>
92     /// Should return the indexes position of the triangles
93     /// </summary>
94     /// <returns>array of indexes</returns>
95     public override uint[] GetVerticesIndexes()
96     {
97         if (Points.Count == 3) // Triangle
98             return new uint[] { 0, 1, 2 };
99         else // Any other polygon
100            return Enumerable.Range(0, Points.Count).Select(i => ←
101                (uint)i).ToArray();
102    }
103
104    /// <summary>
105    /// Reset the renderers and create new ones (use when update ←
106    /// any properties of the shape)
107    /// </summary>
108    public override void ImplementObject()
109    {
110        foreach (Renderer render in Renderers)
111            render.DeleteObjects();
112        Renderers.Clear();
113
114        if (Points.Count == 3) // Triangle
115            Renderers.Add(new Renderer(GetVertices(), ←
116                GetVerticesIndexes(), PrimitiveType.Triangles, this));
117        else
118            Renderers.Add(new Renderer(GetVertices(), ←
119                GetVerticesIndexes(), PrimitiveType.LineLoop, this));
120    }
121
122    /// <summary>
123    /// Retrieve the points that made the shape detectable
124    /// </summary>
125    /// <returns>Position of all the points</returns>
126    public override List<Vector2i> GetSelectablePoints()
127    {
128        List<Vector2i> selectablePoints = new List<Vector2i>();
129        if (Points.Count == 3) // Because a triangle is also base ←
130            on the size
131        {
132            selectablePoints.Add(new Vector2i(X, Y));
133            selectablePoints.Add(new Vector2i(X + Width, Y));
134        }
135        foreach (Vector2i point in Points)
136            selectablePoints.Add(new Vector2i(X + point.X, Y + ←
137                point.Y));
138        return selectablePoints;
139    }
140
141    /// <summary>

```

```

133     /// Move the current shape to the given position
134     /// </summary>
135     /// <param name="position">New position</param>
136     public override void Move(Vector2i position)
137     {
138         X = position.X;
139         Y = position.Y;
140         ImplementObject();
141     }
142
143     /// <summary>
144     /// Resize the current shape to the given size
145     /// </summary>
146     /// <param name="size">New size</param>
147     public override void Resize(Vector2i size)
148     {
149         Width = size.X;
150         Height = size.Y;
151         ImplementObject();
152     }
153
154     /// <summary>
155     /// Retrieve the Shape in the SVG format
156     /// </summary>
157     /// <returns>string of the SVG format</returns>
158     public override string ToString()
159     {
160         string output = "<polygon points=\"";
161         if (Points.Count == 3) // Because a triangle is also base ↪
162             on the size
163             output += String.Format("{0},{1},{2},{3},{4},{5}", X + ↪
164             Width / 2, Y, X, Y + Height, X + Width, Y + Height);
165         else
166             foreach (Vector2i point in Points)
167                 output += string.Format("{0},{1}", X + point.X, Y ↪
168                 + point.Y);
169         return output.Trim() + $"\" fill=\\"{ColorToHexadecimal(Color)}\\"/>";
170     }
171 }

```

Listing 1.13 – ./source/csharp/SVGPolygon.cs

1.1.14 SVGPath.cs

```

1  iż&using FreeFrame.Components.Shapes.Path;
2  using OpenTK.Graphics.OpenGL4;
3  using OpenTK.Mathematics;
4  using System.Text.RegularExpressions;
5  using System.Xml;
6
7  namespace FreeFrame.Components.Shapes
8  {
9      public class SVGPath : Shape
10     {
11         // First point x,y of the path
12         int _x0, _y0 = 0;
13         readonly Dictionary<char, Regex> _dAttributeRegex = new()
14         {
15             { 'm', new Regex(@"\u00a0*(-?\d+)\u00a0*,\u00a0*(-?\d+)\u00a0*") },
16             { 'l', new Regex(@"\u00a0*(-?\d+)\u00a0*,\u00a0*(-?\d+)\u00a0*") },
17             { 'h', new Regex(@"\u00a0*(-?\d+)\u00a0*") },
18             { 'v', new Regex(@"\u00a0*(-?\d+)\u00a0*") },
19             { 'c', new Regex(@"\u00a0*(-?\d+)\u00a0*,\u00a0*(-?\d+)\u00a0+(-?\d+)\u00a0*,\u00a0*"
20                 "*(-?\d+)\u00a0+(-?\d+)\u00a0*,\u00a0*(-?\d+)\u00a0*") },
21             { 's', new Regex(@"\u00a0*(-?\d+)\u00a0*,\u00a0*(-?\d+)\u00a0+(-?\d+)\u00a0*,\u00a0*"
22                 "*(-?\d+)\u00a0*") },
23         };
24     }
25 }

```

```

21     { 'q', new Regex(@"^*(-?\d+)*,^*(-?\d+)*(-?\d+)*,^←
22         *(-?\d+)*" ) },
23     { 't', new Regex(@"^*(-?\d+)*,^*(-?\d+)*(-?\d+)*" ) },
24     { 'a', new Regex(@"^*(-?\d+)*(-?\d+)*(-?\d+)*(-?\d+)*(-?\d+)*" ),
25         +(-?\d+)*(-?\d+)*,^*(-?\d+)*(-?\d+)*" ) },
26     { 'z', new Regex("") },
27 };
28
29 List<DrawAttribute> _drawAttributes = new();
30
31 public List<DrawAttribute> DrawAttributes { get => ←
32     _drawAttributes; set => _drawAttributes = value; }
33 public SVGPath() { }
34 public SVGPath(XmlReader reader)
35 {
36     IsResizeable = false;
37     IsCornerRadiusChangeable = false;
38
39     string d = reader["d"] ?? throw new Exception("d not here");
40     Match match;
41     int startIndex = 0;
42
43     for (int i = 0; i < d.Length; i++)
44     {
45         char c = d[i]; // Get current char
46         char lowerC = char.ToLower(d[i]);
47
48         if (_dAttributeRegex.ContainsKey(lowerC))
49         {
50             match = _dAttributeRegex[lowerC].Match(d, ←
51                 startIndex); // Retrieve the associated regular ←
52                 expression
53
54             if (startIndex == 0) // Save default location point ←
55                 for z
56             {
57                 _x0 = Convert.ToInt32(match.Groups[1].Value);
58                 _y0 = Convert.ToInt32(match.Groups[2].Value);
59             }
60
61             switch (lowerC)
62             {
63                 case 'm':
64                     DrawAttributes.Add(new ←
65                         MoveTo(match.Groups[1], match.Groups[2], ←
66                             c == 'm')); // 'm' is relative and 'M' ←
67                             absolute
68                     break;
69                 case 'l':
70                     DrawAttributes.Add(new ←
71                         LineTo(match.Groups[1], match.Groups[2], ←
72                             c == 'l')); // 'l' is relative and 'L' ←
73                             absolute
74                     break;
75                 case 'h':
76                     DrawAttributes.Add(new ←
77                         HorizontalLineTo(match.Groups[1], c == ←
78                             'h')); // 'h' is relative and 'H' absolute
79                     break;
80                 case 'v':
81                     DrawAttributes.Add(new ←
82                         VerticalLineTo(match.Groups[1], c == ←
83                             'v')); // 'v' is relative and 'V' absolute
84                     break;
85                 case 'c':
86                     DrawAttributes.Add(new ←
87                         CurveTo(match.Groups[1], ←
88                             match.Groups[2], match.Groups[3], ←
89                             match.Groups[4], match.Groups[5], ←
90                             match.Groups[6], c == 'c')); // 'c' is ←
91                             relative and 'C' absolute
92                     break;
93             }
94         }
95     }
96 }

```

```

72         case 's':
73             DrawAttributes.Add(new ←
74                 SmoothCurveTo(match.Groups[1], ←
75                     match.Groups[2], match.Groups[3], ←
76                     match.Groups[4], c == 's')); // 's' is ←
77                         relative and 'S' absolute
78             break;
79         case 'q':
80             DrawAttributes.Add(new ←
81                 QuadraticBezierCurveTo(match.Groups[1], ←
82                     match.Groups[2], match.Groups[3], ←
83                     match.Groups[4], c == 'q')); // 'q' is ←
84                         relative and 'Q' absolute
85             break;
86         case 't':
87             DrawAttributes.Add(new ←
88                 SmoothQuadraticBezierCurveTo(match.Groups[1], ←
89                     match.Groups[2], c == 't')); // 't' is ←
90                         relative and 'T' absolute
91             break;
92         case 'z':
93             DrawAttributes.Add(new LineTo(_x0, _y0));
94             break;
95         default:
96             throw new Exception("Unknowed properties in ←
97                             d");
98         }
99     }
100
101     startIndex += match.Groups[0].Length + 1;
102 }
103
104 string color = reader["fill"] ?? DefaultColor;
105 Color = Importer.HexadecimalToRGB(color);
106
107
108 // Update common properties, use the given attributes
109 List<Vector2i> points = GetSelectablePoints();
110 X = points.Min(i => i.X);
111 Y = points.Min(i => i.Y);
112 Width = points.Max(i => i.X) - points.Min(i => i.X);
113 Height = points.Max(i => i.Y) - points.Min(i => i.Y);
114
115 ImplementObject();
116
117 public override float[] GetVertices() => new float[] { };
118 public override uint[] GetVerticesIndexes() => new uint[] { };
119
120 /// <summary>
121 /// Reset the renderers and create new ones (use when update ←
122 /// any properties of the shape)
123 /// </summary>
124 public override void ImplementObject()
125 {
126     Move(new Vector2i(X, Y)); // Update the position since the ←
127         attr doesnt use the X and Y variables directly
128 }
129
130 /// <summary>
131 /// Retrieve the points that made the shape detectable
132 /// </summary>
133 /// <returns>Position of all the points</returns>
134 public override List<Vector2i> GetSelectablePoints()
135 {
136     List<Vector2i> points = new();
137
138     foreach (DrawAttribute attr in DrawAttributes)
139     {
140         points.AddRange(attr.GetSelectablePoints());
141         attr.UpdateLast(); // Each attr depend on the previous. ←
142                         The previous element is the last element that called ←
143                         GetVertices()

```

```

128         }
129     return points;
130 }
131
132 /// <summary>
133 /// Move the current shape to the given position
134 /// </summary>
135 /// <param name="position">New position</param>
136 public override void Move(Vector2i position)
137 {
138     int x = 0, y = 0;
139     int? deltaX = null, deltaY = null;
140     List<Vector2i> points = GetSelectablePoints();
141
142     DeleteObjects(); // Clear the vaos
143
144     if (points.Count > 0)
145     {
146         x = points.Min(i => i.X); // Get current min X and Y of ←
147         the path
148         y = points.Min(i => i.Y);
149     }
150
151     foreach (DrawAttribute attr in DrawAttributes)
152     {
153         Type attrType = attr.GetType();
154         if (attr.IsRelative == false) // Update position of ←
155             each absolute attr
156         {
157             if (deltaX == null || deltaY == null)
158             {
159                 // Get delta X and Y only one time
160                 deltaX = position.X - x;
161                 deltaY = position.Y - y;
162             }
163             attr.X += (int)deltaX;
164             attr.Y += (int)deltaY;
165
166             attr.X1 += (int)deltaX;
167             attr.Y1 += (int)deltaY;
168
169             if (attrType == typeof(CurveTo))
170             {
171                 ((CurveTo)attr).X2 += (int)deltaX;
172                 ((CurveTo)attr).Y2 += (int)deltaY;
173             }
174             else if (attrType == typeof(SmoothCurveTo))
175             {
176                 ((SmoothCurveTo)attr).X2 += (int)deltaX;
177                 ((SmoothCurveTo)attr).Y2 += (int)deltaY;
178             }
179             if (attrType == typeof(CurveTo) ||
180                 attrType == typeof(SmoothCurveTo) ||
181                 attrType == typeof(QuadraticBezierCurveTo) ||
182                 attrType == typeof(SmoothQuadraticBezierCurveTo)) ←
183                 // Only thoses are LineStrip type
184             {
185                 Renderers.Add(new Renderer(attr.GetVertices(), ←
186                     attr.GetVerticesIndexes(), ←
187                     PrimitiveType.LineStrip, this));
188             }
189             else
190             {
191                 Renderers.Add(new Renderer(attr.GetVertices(), ←
192                     attr.GetVerticesIndexes(), PrimitiveType.Lines, ←
193                     this));
194             }
195             attr.UpdateLast(); // Update Last's inner variables
196         }
197
198         // Update common properties

```

```

193         X = position.X;
194         Y = position.Y;
195     }
196
197     /// <summary>
198     /// Resize the current shape to the given size
199     /// </summary>
200     /// <param name="size">New size</param>
201     public override void Resize(Vector2i size) => throw new ←
202         NotImplementedException("Can't resize a path");
203
204     /// <summary>
205     /// Retrieve the Shape in the SVG format
206     /// </summary>
207     /// <returns>string of the SVG format</returns>
208     public override string ToString()
209     {
210         string output = "<path d=\"";
211         DrawAttributes.ForEach(d => output += d.ToString() + " ");
212         return output.Trim() + $"\" fill={ColorToHexadecimal(Color)}\"/>";
213     }
214 }
```

Listing 1.14 – ./source/csharp/SVGPath.cs

1.1.15 DrawAttribute.cs

```

1  iż<using OpenTK.Mathematics;
2  using System.Text.RegularExpressions;
3
4  namespace FreeFrame.Components.Shapes.Path
5  {
6      /// <summary>
7      /// DrawAttribute.
8      /// Attribute: d.
9      /// More info: ←
10     /// https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d
11     /// </summary>
12     public abstract class DrawAttribute
13     {
14         public const int CURVE_ACCURACY = 100;
15
16         int _x, _y, _x1, _y1 = 0;
17
18         private bool _isRelative;
19         public bool IsRelative { get => _isRelative; set => _isRelative ←
20             = value; }
21
22         // Current X1,Y1 point
23         public int Y1 { get => _y1; set => _y1 = value; }
24         public int X1 { get => _x1; set => _x1 = value; }
25
26         // Current X,Y point
27         public int X { get => _x; set => _x = value; }
28         public int Y { get => _y; set => _y = value; }
29
30         // Last drawn values
31         public static (int X, int Y, int X1, int Y1) Last = (0, 0, 0, 0);
32
33         /// <summary>
34         /// Update the Last drawn values
35         /// </summary>
36         public abstract void UpdateLast();
37
38         /// <summary>
39         /// Return the vertices position in NDC format
40         /// </summary>
```

```

39     /// <returns>array of vertices position. x, y, x, y, ... ←
40     /// (clockwise)</returns>
41     public abstract float[] GetVertices();
42
43     /// <summary>
44     /// Return the indexes position of the triangles/lines
45     /// </summary>
46     /// <returns>array of indexes</returns>
47     public abstract uint[] GetVerticesIndexes();
48
49     /// <summary>
50     /// Retrieve the points that made the attribute detectable
51     /// </summary>
52     /// <returns>Position of all the points</returns>
53     public abstract List<Vector2i> GetSelectablePoints();
54
55     /// <summary>
56     /// Retrieve the attribute for the d attribute
57     /// </summary>
58     /// <returns>string of the attribute</returns>
59     public abstract override string ToString();
60 }
61
62     /// <summary>
63     /// MoveTo , M or m.
64     /// Moving the current point to another point.
65     /// More info: ←
66     /// https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d#moveto_path_command
67     /// </summary>
68     public class MoveTo : DrawAttribute
69     {
70         public MoveTo() { }
71
72         /// <summary>
73         /// Moving the current point to another point.
74         /// </summary>
75         /// <param name="x">position x</param>
76         /// <param name="y">positin y</param>
77         /// <param name="isRelative">if true, the points becomes ←
78         /// relatives to the last point</param>
79         public MoveTo(Group x, Group y, bool isRelative = false) : ←
80             this(Convert.ToInt32(x.Value), Convert.ToInt32(y.Value), ←
81             isRelative) { }
82
83         /// <summary>
84         /// Moving the current point to another point.
85         /// </summary>
86         /// <param name="x">position x</param>
87         /// <param name="y">positin y</param>
88         /// <param name="isRelative">if true, the points becomes ←
89         /// relatives to the last point</param>
90         public MoveTo(int x, int y, bool isRelative = false)
91         {
92             X = x;
93             Y = y;
94             IsRelative = isRelative;
95         }
96         public override float[] GetVertices() => Array.Empty<float>(); ←
97             // Move doesnt have any vertices
98
99         public override uint[] GetVerticesIndexes() => ←
100             Array.Empty<uint>();
101
102         public override string ToString() => String.Format("{0}←
103             {1},{2}", IsRelative ? 'm' : 'M', X, Y);
104
105         public override List<Vector2i> GetSelectablePoints() => new ←
106             List<Vector2i>();
107
108         public override void UpdateLast()
109         {
110             if (IsRelative)

```

```

101         {
102             Last.X += X;
103             Last.Y += Y;
104         }
105     else
106     {
107         Last.X = X;
108         Last.Y = Y;
109     }
110 }
111 }
112
113 /// <summary>
114 /// LineTo , L or l.
115 /// Use to draw a straight line from the current point to the given ←
116 /// point.
117 /// More info: ←
118 /// https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d#lineto_path_com
119 /// </summary>
120 public class LineTo : DrawAttribute
121 {
122     public LineTo() { }
123
124     /// <summary>
125     /// Use to draw a straight line from the current point to the ←
126     /// given point.
127     /// </summary>
128     /// <param name="x">end point x</param>
129     /// <param name="y">end point y</param>
130     /// <param name="isRelative">if true, the points becomes ←
131     /// relatives to the last point</param>
132     public LineTo(Group x, Group y, bool isRelative = false) : ←
133         this(Convert.ToInt32(x.Value), Convert.ToInt32(y.Value), ←
134         isRelative) { }
135
136     /// <summary>
137     /// Use to draw a straight line from the current point to the ←
138     /// given point.
139     /// </summary>
140     /// <param name="x">end point x</param>
141     /// <param name="y">end point y</param>
142     /// <param name="isRelative">if true, the points becomes ←
143     /// relatives to the last point</param>
144     public LineTo(int x, int y, bool isRelative = false)
145     {
146         X = x;
147         Y = y;
148         IsRelative = isRelative;
149     }
150     public override float[] GetVertices()
151     {
152         float[] vertices;
153
154         if (IsRelative)
155             vertices = new float[] { Last.X, Last.Y, Last.X + X, ←
156                                     Last.Y + Y };
157         else
158             vertices = new float[] { Last.X, Last.Y, X, Y };
159
160         return vertices;
161     }
162     public override uint[] GetVerticesIndexes() => new uint[] { 0, ←
163         1 };
164
165     public override string ToString() => String.Format("{0}←
166     {1},{2}", IsRelative ? 'l' : 'L', X, Y);
167
168     public override List<Vector2i> GetSelectablePoints()
169     {
170         List<Vector2i> points = new List<Vector2i>();
171
172         if (IsRelative)

```

```

162        {
163            points.Add(new Vector2i(Last.X, Last.Y));
164            points.Add(new Vector2i(Last.X + X, Last.Y + Y));
165        }
166        else
167        {
168            points.Add(new Vector2i(Last.X, Last.Y));
169            points.Add(new Vector2i(X, Y));
170        }
171        return points;
172    }
173
174    public override void UpdateLast()
175    {
176        if (IsRelative)
177        {
178            Last.X += X; // Update last position
179            Last.Y += Y; // Update last position
180        }
181        else
182        {
183            Last.X = X; // Update last position
184            Last.Y = Y; // Update last position
185        }
186    }
187
188
189    /// <summary>
190    /// HorizontalLineTo , H or h .
191    /// Use to draw a horizontal line from the current point to the ←
192    /// given point .
193    /// More info : ←
194    /// https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d#lineto_path_com
195    /// </summary>
196    public class HorizontalLineTo : DrawAttribute
197    {
198        public HorizontalLineTo() { }
199
200        /// <summary>
201        /// Use to draw a horizontal line from the current point to the ←
202        /// given point .
203        /// </summary>
204        /// <param name="x">offset end point x</param>
205        /// <param name="isRelative">if true , the points becomes ←
206        /// relatives to the last point</param>
207        public HorizontalLineTo(Group x, bool isRelative = false) : ←
208            this(Convert.ToInt32(x.Value), isRelative) { }
209
210        /// <summary>
211        /// Use to draw a horizontal line from the current point to the ←
212        /// given point .
213        /// </summary>
214        /// <param name="x">offset end point x</param>
215        /// <param name="isRelative">if true , the points becomes ←
216        /// relatives to the last point</param>
217        public HorizontalLineTo(int x, bool isRelative = false)
218        {
219            X = x;
220            IsRelative = isRelative;
221        }
222
223        public override float[] GetVertices()
224        {
225            float[] vertices;
226
227            if (IsRelative)
228                vertices = new float[] { Last.X, Last.Y, Last.X + X, ←
229                    Last.Y };
230            else
231                vertices = new float[] { Last.X, Last.Y, X, Last.Y };
232
233            return vertices;
234        }

```

```

226     }
227
228     public override uint[] GetVerticesIndexes() => new uint[] { 0, ←
229         1 };
230
231     public override string ToString() => String.Format("{0} ↴ {1}", ←
232         IsRelative ? 'h' : 'H', X);
233
234     public override List<Vector2i> GetSelectablePoints()
235     {
236         List<Vector2i> points = new List<Vector2i>();
237
238         if (IsRelative)
239         {
240             points.Add(new Vector2i(Last.X, Last.Y));
241             points.Add(new Vector2i(Last.X + X, Last.Y));
242         }
243         else
244         {
245             points.Add(new Vector2i(Last.X, Last.Y));
246             points.Add(new Vector2i(X + X, Last.Y));
247         }
248         return points;
249     }
250
251     public override void UpdateLast()
252     {
253         if (IsRelative)
254             Last.X += X; // Update last position
255         else
256             Last.X = X; // Update last position
257     }
258
259     /// <summary>
260     /// VerticalLineTo, V or v.
261     /// Use to draw a vertical line from the current point to the given ←
262     /// point.
263     /// More info: ←
264     /// https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d#lineto_path_com
265     /// </summary>
266     public class VerticalLineTo : DrawAttribute
267     {
268         public VerticalLineTo() { }
269
270         /// <summary>
271         /// Use to draw a vertical line from the current point to the ←
272         /// given point.
273         /// </summary>
274         /// <param name="y">offset end point y</param>
275         /// <param name="isRelative">if true, the points becomes ←
276         /// relatives to the last point</param>
277         public VerticalLineTo(Group y, bool isRelative = false) : ←
278             this(Convert.ToInt32(y.Value), isRelative) { }
279
280         /// <summary>
281         /// Use to draw a vertical line from the current point to the ←
282         /// given point.
283         /// </summary>
284         /// <param name="y">offset end point y</param>
285         /// <param name="isRelative">if true, the points becomes ←
286         /// relatives to the last point</param>
287         public VerticalLineTo(int y, bool isRelative = false)
288         {
289             Y = y;
290             IsRelative = isRelative;
291         }
292
293         public override float[] GetVertices()
294         {
295             float[] vertices;
296
297             if (IsRelative)
298             {
299                 vertices = new float[2];
300                 vertices[0] = Y;
301                 vertices[1] = Last.Y;
302             }
303             else
304             {
305                 vertices = new float[2];
306                 vertices[0] = Y;
307                 vertices[1] = Y + Last.Y;
308             }
309         }
310
311         public override void SetVertices(float[] vertices)
312         {
313             if (IsRelative)
314             {
315                 Y = vertices[0];
316                 Last.Y = vertices[1];
317             }
318             else
319             {
320                 Y = vertices[0];
321                 Last.Y = vertices[1] - Y;
322             }
323         }
324
325         public override void SetVertices(float x, float y)
326         {
327             if (IsRelative)
328             {
329                 Y = y;
330                 Last.Y = y;
331             }
332             else
333             {
334                 Y = y;
335                 Last.Y = y - X;
336             }
337         }
338
339         public override void SetVertices(Vector2i vertex)
340         {
341             if (IsRelative)
342             {
343                 Y = vertex.Y;
344                 Last.Y = vertex.Y;
345             }
346             else
347             {
348                 Y = vertex.Y;
349                 Last.Y = vertex.Y - X;
350             }
351         }
352
353         public override void SetVertices(int y)
354         {
355             if (IsRelative)
356             {
357                 Y = y;
358                 Last.Y = y;
359             }
360             else
361             {
362                 Y = y;
363                 Last.Y = y - X;
364             }
365         }
366
367         public override void SetVertices(uint index)
368         {
369             if (IsRelative)
370             {
371                 Y = (int)vertices[index];
372                 Last.Y = (int)vertices[index];
373             }
374             else
375             {
376                 Y = (int)vertices[index];
377                 Last.Y = (int)vertices[index] - X;
378             }
379         }
380
381         public override void SetVertices(string vertex)
382         {
383             if (IsRelative)
384             {
385                 Y = Convert.ToInt32(vertex);
386                 Last.Y = Y;
387             }
388             else
389             {
390                 Y = Convert.ToInt32(vertex);
391                 Last.Y = Y - X;
392             }
393         }
394
395         public override void SetVertices(ulong index)
396         {
397             if (IsRelative)
398             {
399                 Y = (int)vertices[(int)index];
400                 Last.Y = (int)vertices[(int)index];
401             }
402             else
403             {
404                 Y = (int)vertices[(int)index];
405                 Last.Y = (int)vertices[(int)index] - X;
406             }
407         }
408
409         public override void SetVertices(List<Vector2i> vertices)
410         {
411             if (IsRelative)
412             {
413                 Y = vertices[0].Y;
414                 Last.Y = Y;
415             }
416             else
417             {
418                 Y = vertices[0].Y;
419                 Last.Y = Y - X;
420             }
421         }
422
423         public override void SetVertices(List<float> vertices)
424         {
425             if (IsRelative)
426             {
427                 Y = vertices[0];
428                 Last.Y = Y;
429             }
430             else
431             {
432                 Y = vertices[0];
433                 Last.Y = Y - X;
434             }
435         }
436
437         public override void SetVertices(List<float> vertices, int offset)
438         {
439             if (IsRelative)
440             {
441                 Y = vertices[offset];
442                 Last.Y = Y;
443             }
444             else
445             {
446                 Y = vertices[offset];
447                 Last.Y = Y - X;
448             }
449         }
450
451         public override void SetVertices(List<float> vertices, int offset, int count)
452         {
453             if (IsRelative)
454             {
455                 Y = vertices[offset];
456                 Last.Y = Y;
457             }
458             else
459             {
460                 Y = vertices[offset];
461                 Last.Y = Y - X;
462             }
463         }
464
465         public override void SetVertices(List<float> vertices, int index)
466         {
467             if (IsRelative)
468             {
469                 Y = vertices[(int)index];
470                 Last.Y = Y;
471             }
472             else
473             {
474                 Y = vertices[(int)index];
475                 Last.Y = Y - X;
476             }
477         }
478
479         public override void SetVertices(List<float> vertices, int index, int count)
480         {
481             if (IsRelative)
482             {
483                 Y = vertices[(int)index];
484                 Last.Y = Y;
485             }
486             else
487             {
488                 Y = vertices[(int)index];
489                 Last.Y = Y - X;
490             }
491         }
492
493         public override void SetVertices(List<float> vertices, int index, int count, int stride)
494         {
495             if (IsRelative)
496             {
497                 Y = vertices[(int)index];
498                 Last.Y = Y;
499             }
500             else
501             {
502                 Y = vertices[(int)index];
503                 Last.Y = Y - X;
504             }
505         }
506
507         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset)
508         {
509             if (IsRelative)
510             {
511                 Y = vertices[(int)index];
512                 Last.Y = Y;
513             }
514             else
515             {
516                 Y = vertices[(int)index];
517                 Last.Y = Y - X;
518             }
519         }
520
521         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize)
522         {
523             if (IsRelative)
524             {
525                 Y = vertices[(int)index];
526                 Last.Y = Y;
527             }
528             else
529             {
530                 Y = vertices[(int)index];
531                 Last.Y = Y - X;
532             }
533         }
534
535         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type)
536         {
537             if (IsRelative)
538             {
539                 Y = vertices[(int)index];
540                 Last.Y = Y;
541             }
542             else
543             {
544                 Y = vertices[(int)index];
545                 Last.Y = Y - X;
546             }
547         }
548
549         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2)
550         {
551             if (IsRelative)
552             {
553                 Y = vertices[(int)index];
554                 Last.Y = Y;
555             }
556             else
557             {
558                 Y = vertices[(int)index];
559                 Last.Y = Y - X;
560             }
561         }
562
563         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3)
564         {
565             if (IsRelative)
566             {
567                 Y = vertices[(int)index];
568                 Last.Y = Y;
569             }
570             else
571             {
572                 Y = vertices[(int)index];
573                 Last.Y = Y - X;
574             }
575         }
576
577         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4)
578         {
579             if (IsRelative)
580             {
581                 Y = vertices[(int)index];
582                 Last.Y = Y;
583             }
584             else
585             {
586                 Y = vertices[(int)index];
587                 Last.Y = Y - X;
588             }
589         }
590
591         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5)
592         {
593             if (IsRelative)
594             {
595                 Y = vertices[(int)index];
596                 Last.Y = Y;
597             }
598             else
599             {
600                 Y = vertices[(int)index];
601                 Last.Y = Y - X;
602             }
603         }
604
605         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6)
606         {
607             if (IsRelative)
608             {
609                 Y = vertices[(int)index];
610                 Last.Y = Y;
611             }
612             else
613             {
614                 Y = vertices[(int)index];
615                 Last.Y = Y - X;
616             }
617         }
618
619         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7)
620         {
621             if (IsRelative)
622             {
623                 Y = vertices[(int)index];
624                 Last.Y = Y;
625             }
626             else
627             {
628                 Y = vertices[(int)index];
629                 Last.Y = Y - X;
630             }
631         }
632
633         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8)
634         {
635             if (IsRelative)
636             {
637                 Y = vertices[(int)index];
638                 Last.Y = Y;
639             }
640             else
641             {
642                 Y = vertices[(int)index];
643                 Last.Y = Y - X;
644             }
645         }
646
647         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9)
648         {
649             if (IsRelative)
650             {
651                 Y = vertices[(int)index];
652                 Last.Y = Y;
653             }
654             else
655             {
656                 Y = vertices[(int)index];
657                 Last.Y = Y - X;
658             }
659         }
660
661         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10)
662         {
663             if (IsRelative)
664             {
665                 Y = vertices[(int)index];
666                 Last.Y = Y;
667             }
668             else
669             {
670                 Y = vertices[(int)index];
671                 Last.Y = Y - X;
672             }
673         }
674
675         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11)
676         {
677             if (IsRelative)
678             {
679                 Y = vertices[(int)index];
680                 Last.Y = Y;
681             }
682             else
683             {
684                 Y = vertices[(int)index];
685                 Last.Y = Y - X;
686             }
687         }
688
689         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12)
690         {
691             if (IsRelative)
692             {
693                 Y = vertices[(int)index];
694                 Last.Y = Y;
695             }
696             else
697             {
698                 Y = vertices[(int)index];
699                 Last.Y = Y - X;
700             }
701         }
702
703         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13)
704         {
705             if (IsRelative)
706             {
707                 Y = vertices[(int)index];
708                 Last.Y = Y;
709             }
710             else
711             {
712                 Y = vertices[(int)index];
713                 Last.Y = Y - X;
714             }
715         }
716
717         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14)
718         {
719             if (IsRelative)
720             {
721                 Y = vertices[(int)index];
722                 Last.Y = Y;
723             }
724             else
725             {
726                 Y = vertices[(int)index];
727                 Last.Y = Y - X;
728             }
729         }
730
731         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15)
732         {
733             if (IsRelative)
734             {
735                 Y = vertices[(int)index];
736                 Last.Y = Y;
737             }
738             else
739             {
740                 Y = vertices[(int)index];
741                 Last.Y = Y - X;
742             }
743         }
744
745         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16)
746         {
747             if (IsRelative)
748             {
749                 Y = vertices[(int)index];
750                 Last.Y = Y;
751             }
752             else
753             {
754                 Y = vertices[(int)index];
755                 Last.Y = Y - X;
756             }
757         }
758
759         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17)
759         {
760             if (IsRelative)
761             {
762                 Y = vertices[(int)index];
763                 Last.Y = Y;
764             }
765             else
766             {
767                 Y = vertices[(int)index];
768                 Last.Y = Y - X;
769             }
770         }
771
772         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18)
771         {
772             if (IsRelative)
773             {
774                 Y = vertices[(int)index];
775                 Last.Y = Y;
776             }
777             else
778             {
779                 Y = vertices[(int)index];
780                 Last.Y = Y - X;
781             }
782         }
783
784         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19)
784         {
785             if (IsRelative)
786             {
787                 Y = vertices[(int)index];
788                 Last.Y = Y;
789             }
790             else
791             {
792                 Y = vertices[(int)index];
793                 Last.Y = Y - X;
794             }
795         }
796
797         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20)
796         {
797             if (IsRelative)
798             {
799                 Y = vertices[(int)index];
800                 Last.Y = Y;
801             }
802             else
803             {
804                 Y = vertices[(int)index];
805                 Last.Y = Y - X;
806             }
807         }
808
809         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21)
809         {
810             if (IsRelative)
811             {
812                 Y = vertices[(int)index];
813                 Last.Y = Y;
814             }
815             else
816             {
817                 Y = vertices[(int)index];
818                 Last.Y = Y - X;
819             }
820         }
821
822         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22)
822         {
823             if (IsRelative)
824             {
825                 Y = vertices[(int)index];
826                 Last.Y = Y;
827             }
828             else
829             {
830                 Y = vertices[(int)index];
831                 Last.Y = Y - X;
832             }
833         }
834
835         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23)
824         {
825             if (IsRelative)
826             {
827                 Y = vertices[(int)index];
828                 Last.Y = Y;
829             }
830             else
831             {
832                 Y = vertices[(int)index];
833                 Last.Y = Y - X;
834             }
835         }
836
837         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24)
835         {
836             if (IsRelative)
837             {
838                 Y = vertices[(int)index];
839                 Last.Y = Y;
840             }
841             else
842             {
843                 Y = vertices[(int)index];
844                 Last.Y = Y - X;
845             }
846         }
847
848         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25)
836         {
837             if (IsRelative)
838             {
839                 Y = vertices[(int)index];
840                 Last.Y = Y;
841             }
842             else
843             {
844                 Y = vertices[(int)index];
845                 Last.Y = Y - X;
846             }
847         }
848
849         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26)
837         {
838             if (IsRelative)
839             {
840                 Y = vertices[(int)index];
841                 Last.Y = Y;
842             }
843             else
844             {
845                 Y = vertices[(int)index];
846                 Last.Y = Y - X;
847             }
848         }
849
850         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27)
838         {
839             if (IsRelative)
840             {
841                 Y = vertices[(int)index];
842                 Last.Y = Y;
843             }
844             else
845             {
846                 Y = vertices[(int)index];
847                 Last.Y = Y - X;
848             }
849         }
850
851         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28)
839         {
840             if (IsRelative)
841             {
842                 Y = vertices[(int)index];
843                 Last.Y = Y;
844             }
845             else
846             {
847                 Y = vertices[(int)index];
848                 Last.Y = Y - X;
849             }
850         }
851
852         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29)
840         {
841             if (IsRelative)
842             {
843                 Y = vertices[(int)index];
844                 Last.Y = Y;
845             }
846             else
847             {
848                 Y = vertices[(int)index];
849                 Last.Y = Y - X;
850             }
851         }
852
853         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29, int type30)
841         {
842             if (IsRelative)
843             {
844                 Y = vertices[(int)index];
845                 Last.Y = Y;
846             }
847             else
848             {
849                 Y = vertices[(int)index];
850                 Last.Y = Y - X;
851             }
852         }
853
854         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29, int type30, int type31)
842         {
843             if (IsRelative)
844             {
845                 Y = vertices[(int)index];
846                 Last.Y = Y;
847             }
848             else
849             {
850                 Y = vertices[(int)index];
851                 Last.Y = Y - X;
852             }
853         }
854
855         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29, int type30, int type31, int type32)
843         {
844             if (IsRelative)
845             {
846                 Y = vertices[(int)index];
847                 Last.Y = Y;
848             }
849             else
850             {
851                 Y = vertices[(int)index];
852                 Last.Y = Y - X;
853             }
854         }
855
856         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29, int type30, int type31, int type32, int type33)
844         {
845             if (IsRelative)
846             {
847                 Y = vertices[(int)index];
848                 Last.Y = Y;
849             }
850             else
851             {
852                 Y = vertices[(int)index];
853                 Last.Y = Y - X;
854             }
855         }
856
857         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29, int type30, int type31, int type32, int type33, int type34)
845         {
846             if (IsRelative)
847             {
848                 Y = vertices[(int)index];
849                 Last.Y = Y;
850             }
851             else
852             {
853                 Y = vertices[(int)index];
854                 Last.Y = Y - X;
855             }
856         }
857
858         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29, int type30, int type31, int type32, int type33, int type34, int type35)
846         {
847             if (IsRelative)
848             {
849                 Y = vertices[(int)index];
850                 Last.Y = Y;
851             }
852             else
853             {
854                 Y = vertices[(int)index];
855                 Last.Y = Y - X;
856             }
857         }
858
859         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29, int type30, int type31, int type32, int type33, int type34, int type35, int type36)
847         {
848             if (IsRelative)
849             {
850                 Y = vertices[(int)index];
851                 Last.Y = Y;
852             }
853             else
854             {
855                 Y = vertices[(int)index];
856                 Last.Y = Y - X;
857             }
858         }
859
860         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29, int type30, int type31, int type32, int type33, int type34, int type35, int type36, int type37)
848         {
849             if (IsRelative)
850             {
851                 Y = vertices[(int)index];
852                 Last.Y = Y;
853             }
854             else
855             {
856                 Y = vertices[(int)index];
857                 Last.Y = Y - X;
858             }
859         }
860
861         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29, int type30, int type31, int type32, int type33, int type34, int type35, int type36, int type37, int type38)
849         {
850             if (IsRelative)
851             {
852                 Y = vertices[(int)index];
853                 Last.Y = Y;
854             }
855             else
856             {
857                 Y = vertices[(int)index];
858                 Last.Y = Y - X;
859             }
860         }
861
862         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14, int type15, int type16, int type17, int type18, int type19, int type20, int type21, int type22, int type23, int type24, int type25, int type26, int type27, int type28, int type29, int type30, int type31, int type32, int type33, int type34, int type35, int type36, int type37, int type38, int type39)
850         {
851             if (IsRelative)
852             {
853                 Y = vertices[(int)index];
854                 Last.Y = Y;
855             }
856             else
857             {
858                 Y = vertices[(int)index];
859                 Last.Y = Y - X;
860            }
861         }
862
863         public override void SetVertices(List<float> vertices, int index, int count, int stride, int offset, int elementSize, int type, int type2, int type3, int type4, int type5, int type6, int type7, int type8, int type9, int type10, int type11, int type12, int type13, int type14
```

```

289         vertices = new float[] { Last.X, Last.Y, Last.X, Last.Y ←
290             + Y };
291     else
292         vertices = new float[] { Last.X, Last.Y, Last.X, Y };
293
294     return vertices;
295 }
296
297 public override uint[] GetVerticesIndexes() => new uint[] { 0, ←
298     1 };
299
300 public override List<Vector2i> GetSelectablePoints()
301 {
302     List<Vector2i> points = new();
303
304     if (IsRelative)
305     {
306         points.Add(new Vector2i(Last.X, Last.Y));
307         points.Add(new Vector2i(Last.X, Last.Y));
308     }
309     else
310     {
311         points.Add(new Vector2i(Last.X, Last.Y));
312         points.Add(new Vector2i(Last.X, Y));
313     }
314     return points;
315 }
316
317 public override string ToString() => String.Format("{0} ↵{1}", ←
318     IsRelative ? 'v' : 'V', Y);
319
320 public override void UpdateLast()
321 {
322     if (IsRelative)
323         Last.Y += Y; // Update last position
324     else
325         Last.Y = Y; // Update last position
326 }
327
328 /// <summary>
329 /// CurveTo, Cubic Bézier Curve, C or c.
330 /// Use to draw a cubic Bézier curve from the current point to the ←
331 /// given point.
332 /// More info: ↵
333 /// https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d#cubic_b%C3%A9zier
334 /// </summary>
335 public class CurveTo : DrawAttribute
336 {
337     int _x2;
338     int _y2;
339
340     public int X2 { get => _x2; set => _x2 = value; }
341     public int Y2 { get => _y2; set => _y2 = value; }
342
343     public CurveTo() { }
344
345     /// <summary>
346     /// Use to draw a cubic Bézier curve from the current point to ←
347     /// the given point.
348     /// </summary>
349     /// <param name="x1">start control point x</param>
350     /// <param name="y1">start control point y</param>
351     /// <param name="x2">end control point x</param>
352     /// <param name="y2">end control point y</param>
353     /// <param name="x">end point x</param>
354     /// <param name="y">end point y</param>
355     /// <param name="isRelative">if true, the points becomes ←
356     /// relatives to the last point</param>
357     public CurveTo(Group x1, Group y1, Group x2, Group y2, Group x, ←
358         Group y, bool isRelative = false) : ←
359         this(Convert.ToInt32(x1.Value), Convert.ToInt32(y1.Value), ←

```

```

Convert.ToInt32(x2.Value), Convert.ToInt32(y2.Value), ←
Convert.ToInt32(x.Value), Convert.ToInt32(y.Value), ←
isRelative) { }

352
353    /// <summary>
354    /// Use to draw a cubic Bézier curve from the current point to ←
355    /// the given point.
356    /// </summary>
357    /// <param name="x1">start control point x</param>
358    /// <param name="y1">start control point y</param>
359    /// <param name="x2">end control point x</param>
360    /// <param name="y2">end control point y</param>
361    /// <param name="x">end point x</param>
362    /// <param name="y">end point y</param>
363    /// <param name="isRelative">if true, the points becomes ←
364    /// relatives to the last point</param>
365    public CurveTo(int x1, int y1, int x2, int y2, int x, int y, ←
366        bool isRelative = false)
367    {
368        X1 = x1;
369        Y1 = y1;
370        X2 = x2;
371        Y2 = y2;
372        X = x;
373        Y = y;
374        IsRelative = isRelative;
375    }
376    public override float[] GetVertices()
377    {
378        List<float> vertices = new List<float>();
379        double t;
380        float x, y;
381
382        // Only edges
383        if (IsRelative)
384        {
385            for (int i = 0; i < CURVE_ACCURACY; i++)
386            {
387                t = i / CURVE_ACCURACY;
388
389                x = (float)(Math.Pow((1 - t), 3) * Last.X + 3 * ←
390                    Math.Pow((1 - t), 2) * t * (Last.X + X1) + 3 * ←
391                    (1 - t) * Math.Pow(t, 2) * (Last.X + X2) + ←
392                    Math.Pow(t, 3) * (Last.X + X));
393                y = (float)(Math.Pow((1 - t), 3) * Last.Y + 3 * ←
394                    Math.Pow((1 - t), 2) * t * (Last.Y + Y1) + 3 * ←
395                    (1 - t) * Math.Pow(t, 2) * (Last.Y + Y2) + ←
396                    Math.Pow(t, 3) * (Last.Y + Y));
397
398                vertices.AddRange(new float[] { x, y });
399            }
400        }
401        else
402        {
403            for (int i = 0; i < CURVE_ACCURACY; i++)
404            {
405                t = i / CURVE_ACCURACY;
406
407                x = (float)(Math.Pow((1 - t), 3) * Last.X + 3 * ←
408                    Math.Pow((1 - t), 2) * t * X1 + 3 * (1 - t) * ←
409                    Math.Pow(t, 2) * X2 + Math.Pow(t, 3) * X);
410                y = (float)(Math.Pow((1 - t), 3) * Last.Y + 3 * ←
411                    Math.Pow((1 - t), 2) * t * Y1 + 3 * (1 - t) * ←
412                    Math.Pow(t, 2) * Y2 + Math.Pow(t, 3) * Y);
413
414                vertices.AddRange(new float[] { x, y });
415            }
416        }
417
418        return vertices.ToArray();
419    }
420}

```

```

407     public override uint[] GetVerticesIndexes() => ←
408         Enumerable.Range(0, 100).Select(i => (uint)i).ToArray(); // ←
409         Magic value please dont hard code this
410
411     public override List<Vector2i> GetSelectablePoints()
412     {
413         List<Vector2i> points = new List<Vector2i>();
414
415         if (IsRelative)
416         {
417             points.Add(new Vector2i(Last.X, Last.Y));
418             points.Add(new Vector2i(Last.X + X, Last.Y + Y));
419         }
420         else
421         {
422             points.Add(new Vector2i(Last.X, Last.Y));
423             points.Add(new Vector2i(X, Y));
424         }
425         return points;
426     }
427
428     public override string ToString() => String.Format("{0} {1}, {2} {3}, {4} {5}, {6}", IsRelative ? 'c' : 'C', X1, Y1, X2, Y2, X, Y);
429
430     public override void UpdateLast()
431     {
432         if (IsRelative)
433         {
434             if (X > X2)
435                 Last.X1 = X - X2 + X;
436             else if (X < X2)
437                 Last.X1 = X2 - X + X2;
438             else
439                 Last.X1 = X2;
440
441             if (Y > Y2)
442                 Last.Y1 = Y - Y2 + Y;
443             else if (Y < Y2)
444                 Last.Y1 = Y2 - Y + Y2;
445             else
446                 Last.Y1 += Y2;
447
448             Last.X += X;
449             Last.Y += Y;
450         }
451         else
452         {
453             if (X > X2)
454                 Last.X1 = X - X2 + X;
455             else if (X < X2)
456                 Last.X1 = X2 - X + X2;
457             else
458                 Last.X1 = X;
459
460             if (Y > Y2)
461                 Last.Y1 = Y - Y2 + Y;
462             else if (Y < Y2)
463                 Last.Y1 = Y2 - Y + Y2;
464             else
465                 Last.Y1 = Y;
466
467             Last.X = X;
468             Last.Y = Y;
469         }
470     }
471
472     /// <summary>
473     /// SmoothCurveTo, Smooth Cubic Bézier Curve, S or s.
474     /// Use to draw a smooth cubic Bézier curve from the current point to the given point.

```

```

474    /// More info: ←
475    /// https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d#cubic_b%C3%A9zier
476    /// </summary>
477    public class SmoothCurveTo : DrawAttribute
478    {
479        int _x2;
480        int _y2;
481
482        public int X2 { get => _x2; set => _x2 = value; }
483        public int Y2 { get => _y2; set => _y2 = value; }
484
485        public SmoothCurveTo() { }
486
487        /// <summary>
488        /// Use to draw a smooth cubic Bézier curve from the current ←
489        /// point to the given point.
490        /// </summary>
491        /// <param name="x2">end control point x</param>
492        /// <param name="y2">end control point y</param>
493        /// <param name="x">end point x</param>
494        /// <param name="y">end point y</param>
495        /// <param name="isRelative">if true, the points becomes ←
496        /// relatives to the last point</param>
497        public SmoothCurveTo(Group x2, Group y2, Group x, Group y, bool ←
498            isRelative = false) : this(Convert.ToInt32(x2.Value), ←
499            Convert.ToInt32(y2.Value), Convert.ToInt32(x.Value), ←
500            Convert.ToInt32(y.Value), isRelative) { }
501
502        /// <summary>
503        /// Use to draw a smooth cubic Bézier curve from the current ←
504        /// point to the given point.
505        /// </summary>
506        /// <param name="x2">end control point x</param>
507        /// <param name="y2">end control point y</param>
508        /// <param name="x">end point x</param>
509        /// <param name="y">end point y</param>
510        /// <param name="isRelative">if true, the points becomes ←
511        /// relatives to the last point</param>
512        public SmoothCurveTo(int x2, int y2, int x, int y, bool ←
513            isRelative = false)
514        {
515            X2 = x2;
516            Y2 = y2;
517            X = x;
518            Y = y;
519            IsRelative = isRelative;
520        }
521
522        public override float[] GetVertices()
523        {
524            List<float> vertices = new List<float>();
525            double t;
526            float x, y;
527
528            // Only edges
529            if (IsRelative)
530            {
531                for (int i = 0; i < CURVE_ACCURACY; i++)
532                {
533                    t = i / CURVE_ACCURACY;
534
535                    x = (float)(Math.Pow((1 - t), 3) * Last.X + 3 * ←
536                        Math.Pow((1 - t), 2) * t * (Last.X + Last.X1) + ←
537                        3 * (1 - t) * Math.Pow(t, 2) * (Last.X + X2) + ←
538                        Math.Pow(t, 3) * (Last.X + X));
539                    y = (float)(Math.Pow((1 - t), 3) * Last.Y + 3 * ←
540                        Math.Pow((1 - t), 2) * t * (Last.Y + Last.Y1) + ←
541                        3 * (1 - t) * Math.Pow(t, 2) * (Last.Y + Y2) + ←
542                        Math.Pow(t, 3) * (Last.Y + Y));
543
544                    vertices.AddRange(new float[] { x, y });
545                }
546            }
547        }
548    }
549}
550
```

```

531     }
532     else
533     {
534         for (int i = 0; i < CURVE_ACCURACY; i++)
535         {
536             t = i / CURVE_ACCURACY;
537
538             x = (float)(Math.Pow((1 - t), 3) * Last.X + 3 * ←
539                         Math.Pow((1 - t), 2) * t * Last.X1 + 3 * (1 - t) ←
540                         * Math.Pow(t, 2) * X2 + Math.Pow(t, 3) * X);
541             y = (float)(Math.Pow((1 - t), 3) * Last.Y + 3 * ←
542                         Math.Pow((1 - t), 2) * t * Last.Y1 + 3 * (1 - t) ←
543                         * Math.Pow(t, 2) * Y2 + Math.Pow(t, 3) * Y);
544
545             vertices.AddRange(new float[] { x, y });
546         }
547
548         return vertices.ToArray();
549     }
550
551     public override uint[] GetVerticesIndexes() => ←
552         Enumerable.Range(0, 100).Select(i => (uint)i).ToArray(); // ←
553         Magic value please dont hard code this
554
555     public override List<Vector2i> GetSelectablePoints()
556     {
557         List<Vector2i> points = new List<Vector2i>();
558
559         if (IsRelative)
560         {
561             points.Add(new Vector2i(Last.X, Last.Y));
562             points.Add(new Vector2i(Last.X + X, Last.Y + Y));
563         }
564         else
565         {
566             points.Add(new Vector2i(Last.X, Last.Y));
567             points.Add(new Vector2i(X, Y));
568         }
569         return points;
570     }
571
572     public override string ToString() => String.Format("{0} {1}, {2} ←
573             {3}, {4}", IsRelative ? 's' : 'S', X2, Y2, X, Y);
574
575     public override void UpdateLast()
576     {
577         if (IsRelative)
578         {
579             if (X > X2)
580             {
581                 Last.X1 = X - X2 + X;
582             }
583             else if (X < X2)
584                 Last.X1 = X2 - X + X2;
585             else
586                 Last.X1 = X;
587
588             if (Y > Y2)
589                 Last.Y1 = Y - Y2 + Y;
590             else if (Y < Y2)
591                 Last.Y1 = Y2 - Y + Y2;
592             else
593                 Last.Y1 = Y;
594
595             Last.X += X;
596             Last.Y += Y;
597         }
598         else
599         {
600             if (X > X2)
601                 Last.X1 = X - X2 + X;
602             else if (X < X2)
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695

```

```

596             Last.X1 = X2 - X + X2;
597         else
598             Last.X1 = X;
599
600         if (Y > Y2)
601             Last.Y1 = Y - Y2 + Y;
602         else if (Y < Y2)
603             Last.Y1 = Y2 - Y + Y2;
604         else
605             Last.Y1 = Y;
606
607         Last.X = X;
608         Last.Y = Y;
609     }
610 }
611
612 /// <summary>
613 /// QuadraticBezierCurveTo, Q, q.
614 /// Use to draw a quadratic Bézier curve.
615 /// More info: ↪
616 /// https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d#quadratic_b%C3%
617 /// </summary>
618 public class QuadraticBezierCurveTo : DrawAttribute
619 {
620     public QuadraticBezierCurveTo() { }
621
622     /// <summary>
623     /// Use to draw a quadratic Bézier curve.
624     /// </summary>
625     /// <param name="x1">control point x</param>
626     /// <param name="y1">control point y</param>
627     /// <param name="x">end point x</param>
628     /// <param name="y">end point y</param>
629     /// <param name="isRelative">if true, the points becomes ↪
630     /// relatives to the last point</param>
631     public QuadraticBezierCurveTo(Group x1, Group y1, Group x, ↪
632         Group y, bool isRelative = false) : ↪
633         this(Convert.ToInt32(x1.Value), Convert.ToInt32(y1.Value), ↪
634         Convert.ToInt32(x.Value), Convert.ToInt32(y.Value), ↪
635         isRelative) { }
636
637     /// <summary>
638     /// Use to draw a quadratic Bézier curve.
639     /// </summary>
640     /// <param name="x1">control point x</param>
641     /// <param name="y1">control point y</param>
642     /// <param name="x">end point x</param>
643     /// <param name="y">end point y</param>
644     /// <param name="isRelative">if true, the points becomes ↪
645     /// relatives to the last point</param>
646     public QuadraticBezierCurveTo(int x1, int y1, int x, int y, ↪
647         bool isRelative = false)
648     {
649         X1 = x1;
650         Y1 = y1;
651         X = x;
652         Y = y;
653         IsRelative = isRelative;
654     }
655     public override float[] GetVertices()
656     {
657         List<float> vertices = new List<float>();
658         double t;
659         float x, y;
660
661         // Only edges
662         if (IsRelative)
663         {
664             for (int i = 0; i < CURVE_ACCURACY; i++)
665             {
666                 t = i / CURVE_ACCURACY;
667
668                 vertices.Add(X1);
669                 vertices.Add(Y1);
670
671                 float x1 = X1 + (X - X1) * t;
672                 float y1 = Y1 + (Y - Y1) * t;
673
674                 vertices.Add(x1);
675                 vertices.Add(y1);
676             }
677         }
678     }
679 }

```

```

660             x = (float)(Math.Pow((1 - t), 2) * Last.X + 2 * (1 ←
661                         - t) * t * (Last.X + X1) + Math.Pow(t, 2) * ←
662                         (Last.X + X));
663             vertices.AddRange(new float[] { x, y });
664         }
665     }
666     else
667     {
668         for (int i = 0; i < CURVE_ACCURACY; i++)
669         {
670             t = i / CURVE_ACCURACY;
671
672             x = (float)(Math.Pow((1 - t), 2) * Last.X + 2 * (1 ←
673                         - t) * t * X1 + Math.Pow(t, 2) * X);
674             y = (float)(Math.Pow((1 - t), 2) * Last.Y + 2 * (1 ←
675                         - t) * t * Y1 + Math.Pow(t, 2) * Y);
676
677             vertices.AddRange(new float[] { x, y });
678         }
679
680         return vertices.ToArray();
681     }
682     public override uint[] GetVerticesIndexes() => ←
683         Enumerable.Range(0, 100).Select(i => (uint)i).ToArray(); // ←
684         Magic value please dont hard code this
685
686     public override List<Vector2i> GetSelectablePoints()
687     {
688         List<Vector2i> points = new List<Vector2i>();
689
690         if (IsRelative)
691         {
692             points.Add(new Vector2i(Last.X, Last.Y));
693             points.Add(new Vector2i(Last.X + X, Last.Y + Y));
694         }
695         else
696         {
697             points.Add(new Vector2i(Last.X, Last.Y));
698             points.Add(new Vector2i(X, Y));
699         }
700         return points;
701     }
702
703     public override string ToString() => String.Format("{0} {1} {2} ←
704     {3} {4}", IsRelative ? 'q' : 'Q', X1, Y1, X, Y);
705
706     public override void UpdateLast()
707     {
708         if (IsRelative)
709         {
710             if (X > X1)
711                 Last.X1 += X + X1;
712             else if (X < X1)
713                 Last.X1 += X - X1;
714             else
715                 Last.X1 += X;
716             Last.X += X;
717             Last.Y += Y;
718         }
719         else
720         {
721             if (X > X1)
722                 Last.X1 = X + X1;
723             else if (X < X1)
724                 Last.X1 = X - X1;
725             else
726                 Last.X1 = X;
727         }
728     }
729 
```

```

723             Last.X = X;
724             Last.Y = Y;
725         }
726     }
727 }
728 /// <summary>
729 /// SmoothQuadraticBezierCurveTo, T, t.
730 /// Use to draw a smooth quadratic Bézier curve.
731 /// More info: ↪
732     https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/d#quadratic\_b%C3%80
733 /// </summary>
734 public class SmoothQuadraticBezierCurveTo : DrawAttribute
735 {
736     public SmoothQuadraticBezierCurveTo() { }
737     /// <summary>
738     /// Use to draw a smooth quadratic Bézier curve.
739     /// </summary>
740     /// <param name="x">end point x</param>
741     /// <param name="y">end point y</param>
742     /// <param name="isRelative">if true, the points becomes ↪
743         relatives to the last point</param>
744     public SmoothQuadraticBezierCurveTo(Group x, Group y, bool ↪
745         isRelative = false) : this(Convert.ToInt32\(x.Value\), ↪
746             Convert.ToInt32\(y.Value\), isRelative) { }
747     /// <summary>
748     /// Use to draw a smooth quadratic Bézier curve.
749     /// </summary>
750     /// <param name="x">end point x</param>
751     /// <param name="y">end point y</param>
752     /// <param name="isRelative">if true, the points becomes ↪
753         relatives to the last point</param>
754     public SmoothQuadraticBezierCurveTo(int x, int y, bool ↪
755         isRelative = false)
756     {
757         X = x;
758         Y = y;
759         IsRelative = isRelative;
760     }
761
762     public override float[] GetVertices()
763     {
764         List<float> vertices = new List<float>();
765         double t;
766         float x, y;
767
768         // Only edges
769         if (IsRelative)
770         {
771             for (int i = 0; i < CURVE_ACCURACY; i++)
772             {
773                 t = i / CURVE_ACCURACY;
774
775                 x = (float)(Math.Pow((1 - t), 2) * Last.X + 2 * (1 ↪
776                     - t) * t * (Last.X + Last.X1) + Math.Pow(t, 2) * ↪
777                     (Last.X + X));
778                 y = (float)(Math.Pow((1 - t), 2) * Last.Y + 2 * (1 ↪
779                     - t) * t * (Last.Y + Last.Y1) + Math.Pow(t, 2) * ↪
780                     (Last.Y + Y));
781
782                 vertices.AddRange(new float[] { x, y });
783             }
784         }
785         else
786         {
787             for (int i = 0; i < CURVE_ACCURACY; i++)
788             {
789                 t = i / CURVE_ACCURACY;
790
791                 x = (float)(Math.Pow((1 - t), 2) * Last.X + 2 * (1 ↪
792                     - t) * t * Last.X1 + Math.Pow(t, 2) * X);
793                 y = (float)(Math.Pow((1 - t), 2) * Last.Y + 2 * (1 ↪
794                     - t) * t * Last.Y1 + Math.Pow(t, 2) * Y);
795             }
796         }
797     }
798 }

```

```

783             vertices.AddRange(new float[] { x, y });
784         }
785     }
786
787     return vertices.ToArray();
788 }
789 public override uint[] GetVerticesIndexes() => ←
    Enumerable.Range(0, 100).Select(i => (uint)i).ToArray(); // ←
    Magic value please dont hard code this
790
791 public override List<Vector2i> GetSelectablePoints()
792 {
793     List<Vector2i> points = new List<Vector2i>();
794
795     if (IsRelative)
796     {
797         points.Add(new Vector2i(Last.X, Last.Y));
798         points.Add(new Vector2i(Last.X + X, Last.Y + Y));
799     }
800     else
801     {
802         points.Add(new Vector2i(Last.X, Last.Y));
803         points.Add(new Vector2i(X, Y));
804     }
805     return points;
806 }
807
808
809 public override string ToString() => String.Format("{0}←
    {1}, {2}", IsRelative ? 't' : 'T', X, Y);
810
811 public override void UpdateLast()
812 {
813     if (IsRelative)
814     {
815         if (X > Last.X1)
816             Last.X1 += X + Last.X1;
817         else if (X < Last.X1)
818             Last.X1 += X - Last.X1;
819         else
820             Last.X1 += X;
821         Last.X += X;
822         Last.Y += Y;
823     }
824     else
825     {
826         if (X > Last.X1)
827             Last.X1 = X + Last.X1;
828         else if (X < Last.X1)
829             Last.X1 = X - Last.X1;
830         else
831             Last.X1 = X;
832         Last.X = X;
833         Last.Y = Y;
834     }
835 }
836 }
837 }
```

Listing 1.15 – ./source/csharp/DrawAttribute.cs

1.1.16 Shader.cs

```

1  izdeusing OpenTK.Graphics.OpenGL4;
2  using OpenTK.Mathematics;
3
4  namespace FreeFrame
5  {
6      public class Shader
```

```

7      {
8          private int _program;
9
10         public Shader(string uriVertexShader, string uriFragementShader)
11         {
12             _program = GL.CreateProgram(); // Handler for the shaders
13
14             int vertexShader = CompileShader(uriVertexShader, ←
15                 ShaderType.VertexShader);
16             int fragmentShader = CompileShader(uriFragementShader, ←
17                 ShaderType.FragmentShader);
18
19             GL.AttachShader(_program, vertexShader);
20             GL.AttachShader(_program, fragmentShader);
21
22             GL.LinkProgram(_program); // Put the shaders in their ←
23                             // respective processor
24
25             GL.DeleteShader(vertexShader);
26             GL.DeleteShader(fragmentShader);
27         }
28         int CompileShader(string uri, ShaderType type)
29         {
30             int shader = GL.CreateShader(type);
31
32             GL.ShaderSource(shader, File.ReadAllText(uri)); // Import ←
33                             // the source code of the shader
34
35             GL.CompileShader(shader);
36
37             GL.GetShader(shader, ShaderParameter.CompileStatus, out int ←
38                 compileStatus); // compileStatus is 0 if compile error
39             if (compileStatus == 0)
40             {
41                 Console.WriteLine("{0}: {1}", type.ToString(), ←
42                     GL.GetShaderInfoLog(shader));
43                 throw new Exception("Error while compiling shader");
44             }
45
46             return shader;
47         }
48         /// <summary>
49         /// Retrieve the Uniform location in a shader
50         /// </summary>
51         /// <param name="uniform">The uniform</param>
52         /// <returns>The index of the uniform in the shader</returns>
53         public int GetUniformLocation(string uniform) => ←
54             GL.GetUniformLocation(_program, uniform);
55
56         // Set the given uniform
57         public void SetUniformVec4(int uniform, Vector4 vector) => ←
58             GL.Uniform4(uniform, vector);
59         public void SetUniformVec2(int uniform, Vector2 vector) => ←
60             GL.Uniform2(uniform, vector);
61         public void SetUniformFloat(int uniform, float value) => ←
62             GL.Uniform1(uniform, value);
63         public void SetUniformMat4(int uniform, Matrix4 matrix) => ←
64             GL.UniformMatrix4(uniform, false, ref matrix);
65
66         /// <summary>
67         /// Use the current shaders
68         /// </summary>
69         public void Use() => GL.UseProgram(_program);
70     }

```

Listing 1.16 – ./source/csharp/Shader.cs

1.1.17 FilePicker.cs

```
1  Original Author: prime31
2  Source: ↵
3  https://gist.github.com/prime31/91d1582624eb2635395417393018016e
4  */
5 using ImGuiNET;
6 using System;
7 using System.Collections.Generic;
8 using System.IO;
9 using Num = System.Numerics;
10
11 namespace FreeFrame.Lib.FileExplorer
12 {
13     public class FilePicker
14     {
15         static Dictionary<object, FilePicker> _filePickers = new ←
16             Dictionary<object, FilePicker>();
17
18         public string RootFolder;
19         public string CurrentFolder;
20         public string SelectedFile;
21         public List<string> AllowedExtensions;
22         public bool OnlyAllowFolders;
23
24         public static FilePicker GetFolderPicker(object o, string ←
25             startingPath)
26             => GetFilePicker(o, startingPath, null, true);
27
28         public static FilePicker GetFilePicker(object o, string ←
29             startingPath, string searchFilter = "", bool ←
30             onlyAllowFolders = false)
31         {
32             if (File.Exists(startingPath))
33             {
34                 startingPath = new FileInfo(startingPath).DirectoryName;
35             }
36             else if (string.IsNullOrEmpty(startingPath) || ←
37                 !Directory.Exists(startingPath))
38             {
39                 startingPath = Environment.CurrentDirectory;
40                 if (string.IsNullOrEmpty(startingPath))
41                     startingPath = ApplicationContext.BaseDirectory;
42             }
43
44             if (!_filePickers.TryGetValue(o, out FilePicker fp))
45             {
46                 fp = new FilePicker
47                 {
48                     RootFolder = startingPath,
49                     CurrentFolder = startingPath,
50                     OnlyAllowFolders = onlyAllowFolders
51                 };
52
53                 if (fp.AllowedExtensions != null)
54                     fp.AllowedExtensions.Clear();
55                 else
56                     fp.AllowedExtensions = new List<string>();
57
58                 fp.AllowedExtensions.AddRange(searchFilter.Split(new ←
59                     char[] { '|', }, ←
60                     StringSplitOptions.RemoveEmptyEntries));
61
62                 _filePickers.Add(o, fp);
63             }
64             return fp;
65         }
66
67         public static void Clear() => _filePickers.Clear();
68
69         public bool Draw()
70         {
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
```

```

64     if (CurrentFolder.Length >= 50)
65         ImGui.Text("CurrentFolder: " + CurrentFolder.Replace(CurrentFolder[10..(CurrentFolder.Length - 40)], "..."));
66     else
67         ImGui.Text("CurrentFolder: " + CurrentFolder);
68     bool result = false;
69
70     if (ImGui.BeginChildFrame(1, new Num.Vector2(400, 400)))
71     {
72         var di = new DirectoryInfo(CurrentFolder);
73         if (di.Exists)
74         {
75             if (di.Parent != null) // && CurrentFolder != RootFolder
76                 if (ImGui.Selectable("../", false, ImGuiSelectableFlags.DontClosePopups))
77                     CurrentFolder = di.Parent.FullName;
78
79             var fileSystemEntries = GetFileSystemEntries(di.FullName);
80             foreach (var fse in fileSystemEntries)
81             {
82                 if (Directory.Exists(fse))
83                 {
84                     var name = Path.GetFileName(fse);
85                     ImGui.PushStyleColor(ImGuiCol.Text, new Num.Vector4(0, 125, 255, 255));
86                     if (ImGui.Selectable($"[D]{name}/", false, ImGuiSelectableFlags.DontClosePopups))
87                         CurrentFolder = fse;
88                     ImGui.PopStyleColor();
89                 }
89                 else
90                 {
91                     var name = Path.GetFileName(fse);
92                     bool isSelected = SelectedFile == fse;
93                     if (ImGui.Selectable($"[F]{name}", isSelected, ImGuiSelectableFlags.DontClosePopups))
94                         SelectedFile = fse;
95
96                     if (ImGui.IsMouseDoubleClicked(0))
97                     {
98                         if (isSelected)
99                         {
100                             result = true;
101                             ImGui.CloseCurrentPopup();
102                         }
103                     }
104                 }
105             }
106         }
107     }
108     ImGui.EndChildFrame();
109
110     if (ImGui.Button("Cancel"))
111     {
112         result = false;
113         ImGui.CloseCurrentPopup();
114         Clear();
115     }
116
117     if (OnlyAllowFolders)
118     {
119         ImGui.SameLine();
120         if (ImGui.Button("Open"))
121         {
122             result = true;
123             SelectedFile = CurrentFolder;
124             ImGui.CloseCurrentPopup();
125         }
126     }

```

```

127         }
128     }
129     else if (SelectedFile != null)
130     {
131         ImGui.SameLine();
132         if (ImGui.Button("Open"))
133         {
134             result = true;
135             ImGui.CloseCurrentPopup();
136         }
137     }
138
139     return result;
140 }
141 List<string> GetFileSystemEntries(string fullName)
142 {
143     var files = new List<string>();
144     var dirs = new List<string>();
145
146     foreach (var fse in ←
147             Directory.GetFileSystemEntries(fullName, ""))
148     {
149         if (Directory.Exists(fse))
150         {
151             dirs.Add(fse);
152         }
153         else if (!OnlyAllowFolders)
154         {
155             if (AllowedExtensions != null)
156             {
157                 if ←
158                     (AllowedExtensions.Contains(Path.GetExtension(fse)))
159                     files.Add(fse);
160             }
161             else
162                 files.Add(fse);
163         }
164     }
165
166     var ret = new List<string>(dirs);
167     ret.AddRange(files);
168
169     return ret;
170 }

```

Listing 1.17 – ./source/csharp/FilePicker.cs

1.1.18 FileSaver.cs

```

1 żę /**
2 * Original Author: prime31
3 * Source: ←
4 *           https://gist.github.com/prime31/91d1582624eb2635395417393018016e
5
6 using ImGuiNET;
7 using System;
8 using System.Collections.Generic;
9 using System.IO;
10 using Num = System.Numerics;
11
12 namespace FreeFrame.Lib.FileExplorer
13 {
14     public class FileSaver
15     {
16         static readonly Dictionary<object, FileSaver> _fileSavers = new ←
17             Dictionary<object, FileSaver>();

```

```

17     public string RootFolder;
18     public string CurrentFolder;
19
20     public string Filename;
21
22     public static FileSaver GetFilePicker(object o, string ←
23         startingPath)
24     {
25         if (File.Exists(startingPath))
26         {
27             startingPath = new FileInfo(startingPath).DirectoryName;
28         }
29         else if (string.IsNullOrEmpty(startingPath) || ←
30             !Directory.Exists(startingPath))
31         {
32             startingPath = Environment.CurrentDirectory;
33             if (string.IsNullOrEmpty(startingPath))
34                 startingPath = ApplicationContext.BaseDirectory;
35         }
36
37         if (!_fileSavers.TryGetValue(o, out FileSaver fp))
38         {
39             fp = new FileSaver
40             {
41                 RootFolder = startingPath,
42                 CurrentFolder = startingPath,
43                 Filename = string.Empty,
44             };
45
46             _fileSavers.Add(o, fp);
47         }
48
49         return fp;
50     }
51     public static void Clear() => _fileSavers.Clear();
52
53     public bool Draw()
54     {
55         if (CurrentFolder.Length >= 50)
56             ImGui.Text("CurrentFolder:" + ←
57                 CurrentFolder.Replace(CurrentFolder[10..(CurrentFolder.Length ←
58                     - 40)], "..."));
59         else
60             ImGui.Text("CurrentFolder:" + CurrentFolder);
61         bool result = false;
62
63         if (ImGui.BeginChildFrame(1, new Num.Vector2(400, 400)))
64         {
65             var di = new DirectoryInfo(CurrentFolder);
66             if (di.Exists)
67             {
68                 if (di.Parent != null)
69                     if (ImGui.Selectable("../", false, ←
70                         ImGuiSelectableFlags.DontClosePopups))
71                         CurrentFolder = di.Parent.FullName;
72
73                 var fileSystemEntries = ←
74                     GetFileSystemEntries(di.FullName);
75                 foreach (var fse in fileSystemEntries)
76                 {
77                     if (Directory.Exists(fse))
78                     {
79                         var name = Path.GetFileName(fse);
80                         ImGui.PushStyleColor(ImGuiCol.Text, new ←
81                             Num.Vector4(0, 125, 255, 255));
82                         if (ImGui.Selectable($"[D]{name}/", false, ←
83                             ImGuiSelectableFlags.DontClosePopups))
84                             CurrentFolder = fse;
85                         ImGui.PopStyleColor();
86                     }
87                 }
88             }
89         }
90     }

```

```

81         }
82         ImGui.EndChildFrame();
83
84
85         if (ImGui.Button("Cancel"))
86     {
87             result = false;
88             ImGui.CloseCurrentPopup();
89             Clear();
90         }
91         ImGui.SameLine();
92         if (ImGui.Button("Savehere"))
93     {
94             result = true;
95             ImGui.CloseCurrentPopup();
96         }
97         ImGui.InputText("Filename", ref Filename, 64);
98
99         return result;
100    }
101    List<string> GetFileSystemEntries(string fullName)
102    {
103        var files = new List<string>();
104        var dirs = new List<string>();
105
106        foreach (var fse in ←
107            Directory.GetFileSystemEntries(fullName, ""))
108            if (Directory.Exists(fse))
109                dirs.Add(fse);
110
111        var ret = new List<string>(dirs);
112        ret.AddRange(files);
113
114        return ret;
115    }
116}
117

```

Listing 1.18 – ./source/csharp/FileSaver.cs

1.1.19 ImGuiUtil.cs

```

1  /*
2  * Author: Julius Häger
3  * Source: https://github.com/NogginBops/ImGui.NET\_OpenTK\_Sample
4  */
5  using System;
6  using System.Collections.Generic;
7  using System.Diagnostics;
8  using System.Diagnostics.Contracts;
9  using System.Linq;
10 using System.Runtime.CompilerServices;
11 using System.Text;
12 using System.Threading.Tasks;
13 using OpenTK.Graphics.OpenGL4;
14
15 namespace FreeFrame.Lib.ImGuiTools
16 {
17     static class ImGuiUtil
18     {
19         [Pure]
20         public static float Clamp(float value, float min, float max)
21         {
22             return value < min ? min : value > max ? max : value;
23         }
24
25         [Conditional("DEBUG")]
26         public static void CheckGLError(string title)

```

```

27     {
28         var error = GL.GetError();
29         if (error != ErrorCode.NoError)
30     {
31             Debug.Print($"{title}: {error}");
32         }
33     }
34
35 [MethodImpl(MethodImplOptions.AggressiveInlining)]
36 public static void LabelObject(ObjectLabelIdentifier ←
37     objLabelIdent, int glObject, string name)
38 {
39     GL.ObjectLabel(objLabelIdent, glObject, name.Length, name);
40
41 [MethodImpl(MethodImplOptions.AggressiveInlining)]
42 public static void CreateTexture(TextureTarget target, string ←
43     Name, out int Texture)
44 {
45     GL.CreateTextures(target, 1, out Texture);
46     LabelObject(ObjectLabelIdentifier.Texture, Texture, ←
47         $"Texture:{Name}");
48 }
49
50 [MethodImpl(MethodImplOptions.AggressiveInlining)]
51 public static void CreateProgram(string Name, out int Program)
52 {
53     Program = GL.CreateProgram();
54     LabelObject(ObjectLabelIdentifier.Program, Program, ←
55         $"Program:{Name}");
56 }
57
58 [MethodImpl(MethodImplOptions.AggressiveInlining)]
59 public static void CreateShader(ShaderType type, string Name, ←
60     out int Shader)
61 {
62     Shader = GL.CreateShader(type);
63     LabelObject(ObjectLabelIdentifier.Shader, Shader, $"Shader:{←
64         type}:{Name}");
65 }
66
67 [MethodImpl(MethodImplOptions.AggressiveInlining)]
68 public static void CreateBuffer(string Name, out int Buffer)
69 {
70     GL.CreateBuffers(1, out Buffer);
71     LabelObject(ObjectLabelIdentifier.Buffer, Buffer, $"Buffer:{←
72         Name}");
73 }
74
75 [MethodImpl(MethodImplOptions.AggressiveInlining)]
76 public static void CreateVertexBuffer(string Name, out int ←
77     Buffer) => CreateBuffer($"VBO:{Name}", out Buffer);
78
79 [MethodImpl(MethodImplOptions.AggressiveInlining)]
80 public static void CreateElementBuffer(string Name, out int ←
81     Buffer) => CreateBuffer($"EBO:{Name}", out Buffer);
82
83 [MethodImpl(MethodImplOptions.AggressiveInlining)]
84 public static void CreateVertexArray(string Name, out int VAO)
85 {
86     GL.CreateVertexArrays(1, out VAO);
87     LabelObject(ObjectLabelIdentifier.VertexArray, VAO, $"VAO:{←
88         Name}");
89 }
90 }

```

Listing 1.19 – ./source/csharp/ImGuiUtil.cs

1.1.20 ImGuiController.cs

```
1  żżż/*
2  * Author: Julius Häger
3  * Source: https://github.com/NogginBops/ImGui.NET\_OpenTK\_Sample
4  */
5  using ImGuiNET;
6  using System;
7  using System.Collections.Generic;
8  using System.Runtime.CompilerServices;
9  using OpenTK.Graphics.OpenGL4;
10 using OpenTK.Mathematics;
11 using OpenTK.Windowing.Common.Input;
12 using OpenTK.Windowing.Desktop;
13 using OpenTK.Windowing.GraphicsLibraryFramework;
14
15 namespace FreeFrame.Lib.ImGuiTools
16 {
17     public class ImGuiController : IDisposable
18     {
19         private bool _frameBegun;
20
21         private int _vertexArray;
22         private int _vertexBuffer;
23         private int _vertexBufferSize;
24         private int _indexBuffer;
25         private int _indexBufferSize;
26
27         private ImGuiTexture _fontTexture;
28         private ImGuiShader _shader;
29
30         private int _windowWidth;
31         private int _windowHeight;
32
33         private System.Numerics.Vector2 _scaleFactor = ←
34             System.Numerics.Vector2.One;
35
36         /// <summary>
37         /// Constructs a new ImGuiController.
38         /// </summary>
39         public ImGuiController(int width, int height)
40         {
41             _windowWidth = width;
42             _windowHeight = height;
43
44             IntPtr context = ImGui.CreateContext();
45             ImGui.SetCurrentContext(context);
46             var io = ImGui.GetIO();
47             io.Fonts.AddFontDefault();
48
49             io.BackendFlags |= ImGuiBackendFlags.RendererHasVtxOffset;
50
51             CreateDeviceResources();
52             SetKeyMappings();
53
54             SetPerFrameImGuiData(1f / 60f);
55
56             ImGui.NewFrame();
57             _frameBegun = true;
58         }
59
60         public void WindowResized(int width, int height)
61         {
62             _windowWidth = width;
63             _windowHeight = height;
64         }
65         public void DestroyDeviceObjects()
66         {
67             Dispose();
68         }
69     }
70 }
```

```

69
70     public void CreateDeviceResources()
71     {
72         ImGuiUtil.CreateVertexArray("ImGui", out _vertexArray);
73
74         _vertexBufferSize = 10000;
75         _indexBufferSize = 2000;
76
77         ImGuiUtil.CreateVertexBuffer("ImGui", out _vertexBuffer);
78         ImGuiUtil.CreateElementBuffer("ImGui", out _indexBuffer);
79         GL.NamedBufferData(_vertexBuffer, _vertexBufferSize, ←
80             IntPtr.Zero, BufferUsageHint.DynamicDraw);
81         GL.NamedBufferData(_indexBuffer, _indexBufferSize, ←
82             IntPtr.Zero, BufferUsageHint.DynamicDraw);
83
84         RecreateFontDeviceTexture();
85
86         string VertexSource = @"#version 330 core
87
88         uniform mat4 projection_matrix;
89
90         layout(location=0) in vec2 in_position;
91         layout(location=1) in vec2 in_texCoord;
92         layout(location=2) in vec4 in_color;
93
94         out vec4 color;
95         out vec2 texCoord;
96
97         void main()
98         {
99             gl_Position = projection_matrix * vec4(in_position, 0, 1);
100            color = in_color;
101            texCoord = in_texCoord;
102        }";
103        string FragmentSource = @"#version 330 core
104
105        uniform sampler2D in_fontTexture;
106
107        in vec4 color;
108        in vec2 texCoord;
109
110        out vec4 outputColor;
111
112        void main()
113        {
114            outputColor = color * texture(in_fontTexture, texCoord);
115        }
116        _shader = new ImGuiShader("ImGui", VertexSource, ←
117            FragmentSource);
118
119        GL.VertexArrayVertexBuffer(_vertexArray, 0, _vertexBuffer, ←
120            IntPtr.Zero, Unsafe.SizeOf<ImDrawVert>());
121        GL.VertexArrayElementBuffer(_vertexArray, _indexBuffer);
122
123        GL.EnableVertexAttribArray(_vertexArray, 0);
124        GL.VertexArrayAttribBinding(_vertexArray, 0, 0);
125        GL.VertexArrayAttribFormat(_vertexArray, 0, 2, ←
126            VertexAttribType.Float, false, 0);
127
128        GL.EnableVertexAttribArray(_vertexArray, 1);
129        GL.VertexArrayAttribBinding(_vertexArray, 1, 0);
130        GL.VertexArrayAttribFormat(_vertexArray, 1, 2, ←
131            VertexAttribType.Float, false, 8);
132
133        GL.EnableVertexAttribArray(_vertexArray, 2);
134        GL.VertexArrayAttribBinding(_vertexArray, 2, 0);
135        GL.VertexArrayAttribFormat(_vertexArray, 2, 4, ←
136            VertexAttribType.UnsignedByte, true, 16);
137
138        ImGuiUtil.CheckGLError("End of ImGui setup");
139    }

```

```

134     /// <summary>
135     /// Recreates the device texture used to render text.
136     /// </summary>
137     public void RecreateFontDeviceTexture()
138     {
139         ImGuiIOPtr io = ImGui.GetIO();
140         io.Fonts.GetTexDataAsRGBA32(out IntPtr pixels, out int width,
141                                     out int height, out int bytesPerPixel);
142
143         _fontTexture = new ImGuiTexture("ImGuiTextAtlas", width, height,
144                                         pixels);
145         _fontTexture.SetMagFilter(TextureMagFilter.Linear);
146         _fontTexture.SetMinFilter(TextureMinFilter.Linear);
147
148         io.Fonts.SetTexID((IntPtr)_fontTexture.GLTexture);
149
150     }
151
152     /// <summary>
153     /// Renders the ImGui draw list data.
154     /// </summary>
155     public void Render()
156     {
157         if (_frameBegun)
158         {
159             _frameBegun = false;
160             ImGui.Render();
161             RenderImGuiDrawData(ImGui.GetDrawData());
162         }
163
164     }
165
166     /// <summary>
167     /// Updates ImGui input and IO configuration state.
168     /// </summary>
169     public void Update(GameWindow wnd, float deltaSeconds)
170     {
171         if (_frameBegun)
172         {
173             ImGui.Render();
174
175             SetPerFrameImGuiData(deltaSeconds);
176             UpdateImGuiInput(wnd);
177
178             _frameBegun = true;
179             ImGui.NewFrame();
180         }
181
182     }
183
184     /// <summary>
185     /// Sets per-frame data based on the associated window.
186     /// This is called by Update(float).
187     /// </summary>
188     private void SetPerFrameImGuiData(float deltaSeconds)
189     {
190         ImGuiIOPtr io = ImGui.GetIO();
191         io.DisplaySize = new System.Numerics.Vector2(
192             _windowWidth / _scaleFactor.X,
193             _windowHeight / _scaleFactor.Y);
194         io.DisplayFramebufferScale = _scaleFactor;
195         io.DeltaTime = deltaSeconds; // DeltaTime is in seconds.
196     }
197
198     readonly List<char> PressedChars = new List<char>();
199
200     private void UpdateImGuiInput(GameWindow wnd)
201     {
202         ImGuiIOPtr io = ImGui.GetIO();
203
204         MouseState MouseState = wnd.MouseState;
205         KeyboardState KeyboardState = wnd.KeyboardState;

```

```

204     io.MouseDown[0] = MouseState(MouseButton.Left);
205     io.MouseDown[1] = MouseState(MouseButton.Right);
206     io.MouseDown[2] = MouseState(MouseButton.Middle);
207
208     var screenPoint = new Vector2i((int)MouseState.X, ←
209                                     (int)MouseState.Y);
210     var point = screenPoint; //wnd.PointToClient(screenPoint);
211     io.MousePos = new System.Numerics.Vector2(point.X, point.Y);
212
213     foreach (Keys key in Enum.GetValues(typeof(Keys)))
214     {
215         if (key == Keys.Unknown)
216         {
217             continue;
218         }
219         io.KeysDown[(int)key] = KeyboardState.IsKeyDown(key);
220     }
221
222     foreach (var c in PressedChars)
223     {
224         io.AddInputCharacter(c);
225     }
226     PressedChars.Clear();
227
228     io.KeyCtrl = KeyboardState.IsKeyDown(Keys.LeftControl) || ←
229                 KeyboardState.IsKeyDown(Keys.RightControl);
230     io.KeyAlt = KeyboardState.IsKeyDown(Keys.LeftAlt) || ←
231                 KeyboardState.IsKeyDown(Keys.RightAlt);
232     io.KeyShift = KeyboardState.IsKeyDown(Keys.LeftShift) || ←
233                 KeyboardState.IsKeyDown(Keys.RightShift);
234     io.KeySuper = KeyboardState.IsKeyDown(Keys.LeftSuper) || ←
235                 KeyboardState.IsKeyDown(Keys.RightSuper);
236 }
237
238     internal void PressChar(char keyChar)
239     {
240         PressedChars.Add(keyChar);
241     }
242
243     internal void MouseScroll(Vector2 offset)
244     {
245         ImGuiIOPtr io = ImGui.GetIO();
246
247         io.MouseWheel = offset.Y;
248         io.MouseWheelH = offset.X;
249     }
250
251     private static void SetKeyMappings()
252     {
253         ImGuiIOPtr io = ImGui.GetIO();
254
255         io.KeyMap[(int)ImGuiKey.Tab] = (int)Keys.Tab;
256         io.KeyMap[(int)ImGuiKey.LeftArrow] = (int)Keys.Left;
257         io.KeyMap[(int)ImGuiKey.RightArrow] = (int)Keys.Right;
258         io.KeyMap[(int)ImGuiKey.UpArrow] = (int)Keys.Up;
259         io.KeyMap[(int)ImGuiKey.DownArrow] = (int)Keys.Down;
260         io.KeyMap[(int)ImGuiKey.PageUp] = (int)Keys.PageUp;
261         io.KeyMap[(int)ImGuiKey.PageDown] = (int)Keys.PageDown;
262         io.KeyMap[(int)ImGuiKey.Home] = (int)Keys.Home;
263         io.KeyMap[(int)ImGuiKey.End] = (int)Keys.End;
264         io.KeyMap[(int)ImGuiKey.Delete] = (int)Keys.Delete;
265         io.KeyMap[(int)ImGuiKey.Backspace] = (int)Keys.Backspace;
266         io.KeyMap[(int)ImGuiKey.Enter] = (int)Keys.Enter;
267         io.KeyMap[(int)ImGuiKey.Escape] = (int)Keys.Escape;
268         io.KeyMap[(int)ImGuiKey.A] = (int)Keys.A;
269         io.KeyMap[(int)ImGuiKey.C] = (int)Keys.C;
270         io.KeyMap[(int)ImGuiKey.V] = (int)Keys.V;
271         io.KeyMap[(int)ImGuiKey.X] = (int)Keys.X;
272         io.KeyMap[(int)ImGuiKey.Y] = (int)Keys.Y;
273         io.KeyMap[(int)ImGuiKey.Z] = (int)Keys.Z;
274     }
275
276     private void RenderImGuiData(ImGuiDrawData draw_data)

```

```

271     {
272         if (draw_data.CmdListsCount == 0)
273         {
274             return;
275         }
276
277         for (int i = 0; i < draw_data.CmdListsCount; i++)
278         {
279             ImDrawListPtr cmd_list = draw_data.CmdListsRange[i];
280
281             int vertexSize = cmd_list.VtxBuffer.Size * ←
282                 Unsafe.SizeOf<ImDrawVert>();
283             if (vertexSize > _vertexBufferSize)
284             {
285                 int newSize = (int)Math.Max(_vertexBufferSize * ←
286                     1.5f, vertexSize);
287                 GL.NamedBufferData(_vertexBuffer, newSize, ←
288                     IntPtr.Zero, BufferUsageHint.DynamicDraw);
289                 _vertexBufferSize = newSize;
290             }
291
292             int indexSize = cmd_list.IdxBuffer.Size * sizeof(ushort);
293             if (indexSize > _indexBufferSize)
294             {
295                 int newSize = (int)Math.Max(_indexBufferSize * ←
296                     1.5f, indexSize);
297                 GL.NamedBufferData(_indexBuffer, newSize, ←
298                     IntPtr.Zero, BufferUsageHint.DynamicDraw);
299                 _indexBufferSize = newSize;
300             }
301         }
302
303         // Setup orthographic projection matrix into our constant ←
304         // buffer
305         ImGuiIO* io = ImGui.GetIO();
306         Matrix4 mvp = Matrix4.CreateOrthographicOffCenter(
307             0.0f,
308             io.DisplaySize.X,
309             io.DisplaySize.Y,
310             0.0f,
311             -1.0f,
312             1.0f);
313
314         _shader.UseShader();
315         GL.UniformMatrix4(_shader.GetUniformLocation("projection_matrix"), ←
316             false, ref mvp);
317         GL.Uniform1(_shader.GetUniformLocation("in_fontTexture"), 0);
318         ImGuiUtil.CheckGLError("Projection");
319
320         GL.BindVertexArray(_vertexArray);
321         ImGuiUtil.CheckGLError("VAO");
322
323         draw_data.ScaleClipRects(io.DisplayFramebufferScale);
324
325         GL.Enable(EnableCap.Blend);
326         GL.Enable(EnableCap.ScissorTest);
327         GL.BLEND_EQUATION(BlendEquationMode.FuncAdd);
328         GL.BLEND_FUNC(BlendingFactor.SrcAlpha, ←
329             BlendingFactor.OneMinusSrcAlpha);
330         GL.Disable(EnableCap.CullFace);
331         GL.Disable(EnableCap.DepthTest);
332
333         // Render command lists
334         for (int n = 0; n < draw_data.CmdListsCount; n++)
335         {
336             ImDrawListPtr cmd_list = draw_data.CmdListsRange[n];
337
338             GL.NamedBufferSubData(_vertexBuffer, IntPtr.Zero, ←
339                 cmd_list.VtxBuffer.Size * ←
340                 Unsafe.SizeOf<ImDrawVert>(), cmd_list.VtxBuffer.Data);
341             ImGuiUtil.CheckGLError($"Data_{n} Vert_{n}");
342         }

```

```

333         GL.NamedBufferSubData(_indexBuffer, IntPtr.Zero, ←
334             cmd_list.IdxBuffer.Size * sizeof(ushort), ←
335             cmd_list.IdxBuffer.Data);
336         ImGuiUtil.CheckGLError($"Data_{n}Idx_{n}");
337     }
338     for (int cmd_i = 0; cmd_i < cmd_list.CmdBuffer.Size; ←
339         cmd_i++)
340     {
341         ImDrawCmdPtr pcmd = cmd_list.CmdBuffer[cmd_i];
342         if (pcmd.UserCallback != IntPtr.Zero)
343         {
344             throw new NotImplementedException();
345         }
346         else
347         {
348             GL.ActiveTexture(TextureUnit.Texture0);
349             GL.BindTexture(TextureTarget.Texture2D, ←
350                 (int)pcmd.TextureId);
351             ImGuiUtil.CheckGLError("Texture");
352
353             // We do _windowHeight - (int)clip.W instead of ←
354             // (int)clip.Y because gl has flipped Y when it ←
355             // comes to these coordinates
356             var clip = pcmd.ClipRect;
357             GL.Scissor((int)clip.X, _windowHeight - ←
358                 (int)clip.W, (int)(clip.Z - clip.X), ←
359                 (int)(clip.W - clip.Y));
360             ImGuiUtil.CheckGLError("Scissor");
361
362             if ((io.BackendFlags & ←
363                 ImGuiBackendFlags.RendererHasVtxOffset) != 0)
364             {
365                 GL.DrawElementsBaseVertex(PrimitiveType.Triangles, ←
366                     (int)pcmd.ElemCount, ←
367                     DrawElementsType.UnsignedShort, ←
368                     (IntPtr)(pcmd.IdxOffset * ←
369                         sizeof(ushort)), (int)pcmd.VtxOffset);
370             }
371             else
372             {
373                 GL.DrawElements(BeginMode.Triangles, ←
374                     (int)pcmd.ElemCount, ←
375                     DrawElementsType.UnsignedShort, ←
376                     (int)pcmd.IdxOffset * sizeof(ushort));
377             }
378             ImGuiUtil.CheckGLError("Draw");
379         }
380     }
381     GL.Disable(EnableCap.Blend);
382     GL.Disable(EnableCap.ScissorTest);
383 }
384
385     /// <summary>
386     /// Frees all graphics resources used by the renderer.
387     /// </summary>
388     public void Dispose()
389     {
390         GL.DeleteVertexArray(_vertexArray);
391         GL.DeleteBuffer(_vertexBuffer);
392         GL.DeleteBuffer(_indexBuffer);
393
394         _fontTexture.Dispose();
395         _shader.Dispose();
396     }
397 }
398 }
```

Listing 1.20 – ./source/csharp/ImGuiController.cs

1.1.21 ImGuiShader.cs

```
1 żżż/*
2 * Author: Julius Häger
3 * Source: https://github.com/NogginBops/ImGui.NET\_OpenTK\_Sample
4 */
5 using System;
6 using System.Collections.Generic;
7 using System.Diagnostics;
8 using System.IO;
9 using System.Linq;
10 using System.Text;
11 using System.Threading.Tasks;
12 using System.Runtime.CompilerServices;
13 using System.Threading;
14 using OpenTK.Graphics.OpenGL4;
15
16 namespace FreeFrame.Lib.ImGuiTools
17 {
18     struct UniformFieldInfo
19     {
20         public int Location;
21         public string Name;
22         public int Size;
23         public ActiveUniformType Type;
24     }
25
26     class ImGuiShader
27     {
28         public readonly string Name;
29         public int Program { get; private set; }
30         private readonly Dictionary<string, int> UniformToLocation = ←
31             new Dictionary<string, int>();
32         private bool Initialized = false;
33
34         private readonly (ShaderType Type, string Path)[] Files;
35
36         public ImGuiShader(string name, string vertexShader, string ←
37             fragmentShader)
38         {
39             Name = name;
40             Files = new []{
41                 (ShaderType.VertexShader, vertexShader),
42                 (ShaderType.FragmentShader, fragmentShader),
43             };
44             Program = CreateProgram(name, Files);
45         }
46         public void UseShader()
47         {
48             GL.UseProgram(Program);
49         }
50
51         public void Dispose()
52         {
53             if (Initialized)
54             {
55                 GL.DeleteProgram(Program);
56                 Initialized = false;
57             }
58         }
59
60         public UniformFieldInfo[] GetUniforms()
61         {
62             GL.GetProgram(Program, ←
63                 GetProgramParameterName.ActiveUniforms, out int ←
64                 UniformCount);
65
66             UniformFieldInfo[] Uniforms = new ←
67                 UniformFieldInfo[UniformCount];
68
69             for (int i = 0; i < UniformCount; i++)
```

```

65     {
66         string Name = GL.GetActiveUniform(Program, i, out int ←
67             Size, out ActiveUniformType Type);
68
69         UniformFieldInfo FieldInfo;
70         FieldInfo.Location = GetUniformLocation(Name);
71         FieldInfo.Name = Name;
72         FieldInfo.Size = Size;
73         FieldInfo.Type = Type;
74
75         Uniforms[i] = FieldInfo;
76     }
77
78     return Uniforms;
79 }
80
81 [MethodImpl(MethodImplOptions.AggressiveInlining)]
82 public int GetUniformLocation(string uniform)
83 {
84     if (UniformToLocation.TryGetValue(uniform, out int ←
85         location) == false)
86     {
87         location = GL.GetUniformLocation(Program, uniform);
88         UniformToLocation.Add(uniform, location);
89
90         if (location == -1)
91         {
92             Debug.Print($"The uniform '{uniform}' does not ←
93                 exist in the shader '{Name}'!");
94         }
95     }
96
97     return location;
98 }
99
100 private int CreateProgram(string name, params (ShaderType Type, ←
101     string source)[] shaderPaths)
102 {
103     ImGuiUtil.CreateProgram(name, out int Program);
104
105     int[] Shaders = new int[shaderPaths.Length];
106     for (int i = 0; i < shaderPaths.Length; i++)
107     {
108         Shaders[i] = CompileShader(name, shaderPaths[i].Type, ←
109             shaderPaths[i].source);
110     }
111
112     foreach (var shader in Shaders)
113         GL.AttachShader(Program, shader);
114
115     GL.LinkProgram(Program);
116
117     GL.GetProgram(Program, GetProgramParameterName.LinkStatus, ←
118         out int Success);
119     if (Success == 0)
120     {
121         string Info = GL.GetProgramInfoLog(Program);
122         Debug.WriteLine($"GL.LinkProgram had info log ←
123             [{name}]:\n{Info}");
124     }
125
126     foreach (var Shader in Shaders)
127     {
128         GL.DetachShader(Program, Shader);
129         GL.DeleteShader(Shader);
130     }
131
132     Initialized = true;
133
134     return Program;
135 }
136
137 }
```

```

130     private int CompileShader(string name, ShaderType type, string ↵
131         source)
132     {
133         ImGuiUtil.CreateShader(type, name, out int Shader);
134         GL.ShaderSource(Shader, source);
135         GL.CompileShader(Shader);
136
137         GL.GetShader(Shader, ShaderParameter.CompileStatus, out int ↵
138             success);
139         if (success == 0)
140         {
141             string Info = GL.GetShaderInfoLog(Shader);
142             Debug.WriteLine($"GL.CompileShader for shader '{Name}' ↵
143                 [{type}] had info log:\n{Info}");
144         }
145
146         return Shader;
147     }
148 }

```

Listing 1.21 – ./source/csharp/ImGuiShader.cs

1.1.22 ImGuiTexture.cs

```

1 /*
2 * Author: Julius Häger
3 * Source: https://github.com/NogginBops/ImGui.NET\_OpenTK\_Sample
4 */
5 using System;
6 using System.Collections.Generic;
7 using System.Diagnostics;
8 using System.Drawing;
9 using System.Drawing.Imaging;
10 using System.Linq;
11 using System.Text;
12 using System.Threading.Tasks;
13 using OpenTK.Graphics.OpenGL4;
14 using PixelFormat = OpenTK.Graphics.OpenGL4.PixelFormat;
15
16 namespace FreeFrame.Lib.ImGuiTools
17 {
18     public enum TextureCoordinate
19     {
20         S = TextureParameterName.TextureWrapS,
21         T = TextureParameterName.TextureWrapT,
22         R = TextureParameterName.TextureWrapR
23     }
24
25     class ImGuiTexture : IDisposable
26     {
27         public const SizedInternalFormat Srgb8Alpha8 = ←
28             (SizedInternalFormat)All.Srgb8Alpha8;
29         public const SizedInternalFormat RGB32F = ←
30             (SizedInternalFormat)All.Rgb32f;
31
32         public const GetPName MAX_TEXTURE_MAX_ANISOTROPY = ←
33             (GetPName)0x84FF;
34
35         public static readonly float MaxAniso;
36
37         static ImGuiTexture()
38         {
39             MaxAniso = GL.GetFloat(MAX_TEXTURE_MAX_ANISOTROPY);
40         }
41
42         public readonly string Name;
43         public readonly int GLTexture;
44         public readonly int Width, Height;

```

```

42     public readonly int MipmapLevels;
43     public readonly SizedInternalFormat InternalFormat;
44
45     public ImGuiTexture(string name, Bitmap image, bool ←
46         generateMipmaps, bool srgb)
47     {
48         Name = name;
49         Width = image.Width;
50         Height = image.Height;
51         InternalFormat = srgb ? Srgb8Alpha8 : ←
52             SizedInternalFormat.Rgba8;
53
54         if (generateMipmaps)
55         {
56             // Calculate how many levels to generate for this texture
57             MipmapLevels = (int)Math.Floor(Math.Log(Math.Max(Width, ←
58                 Height), 2));
59         }
60         else
61         {
62             // There is only one level
63             MipmapLevels = 1;
64         }
65
66         ImGuiUtil.CheckGLError("Clear");
67
68         ImGuiUtil.CreateTexture(TextureTarget.Texture2D, Name, out ←
69             GLTexture);
70         GL.TextureStorage2D(GLTexture, MipmapLevels, ←
71             InternalFormat, Width, Height);
72         ImGuiUtil.CheckGLError("Storage2d");
73
74         BitmapData data = image.LockBits(new Rectangle(0, 0, Width, ←
75             Height),
76             ImageLockMode.ReadOnly, ←
77             global::System.Drawing.Imaging.PixelFormat.Format32bppArgb);
78
79         GL.TextureSubImage2D(GLTexture, 0, 0, 0, Width, Height, ←
80             PixelFormat.Bgra, PixelType.UnsignedByte, data.Scan0);
81         ImGuiUtil.CheckGLError("SubImage");
82
83         image.UnlockBits(data);
84
85         if (generateMipmaps) GL.GenerateTextureMipmap(GLTexture);
86
87         GL.TextureParameter(GLTexture, ←
88             TextureParameterName.TextureWrapS, ←
89             (int)TextureWrapMode.Repeat);
90         ImGuiUtil.CheckGLError("WrapS");
91         GL.TextureParameter(GLTexture, ←
92             TextureParameterName.TextureWrapT, ←
93             (int)TextureWrapMode.Repeat);
94         ImGuiUtil.CheckGLError("WrapT");
95
96         GL.TextureParameter(GLTexture, ←
97             TextureParameterName.TextureMinFilter, ←
98             (int)(generateMipmaps ? TextureMinFilter.Linear : ←
99                 TextureMinFilter.LinearMipmapLinear));
100        GL.TextureParameter(GLTexture, ←
101            TextureParameterName.TextureMagFilter, ←
102            (int)TextureMagFilter.Linear);
103        ImGuiUtil.CheckGLError("Filtering");
104
105        GL.TextureParameter(GLTexture, ←
106            TextureParameterName.TextureMaxLevel, MipmapLevels - 1);
107
108        // This is a bit weird to do here
109        image.Dispose();
110    }
111
112    public ImGuiTexture(string name, int GLTex, int width, int ←
113        height, int mipmaplevels, SizedInternalFormat internalFormat)

```

```

95      {
96          Name = name;
97          GLTexture = GLTex;
98          Width = width;
99          Height = height;
100         MipmapLevels = mipmaplevels;
101         InternalFormat = internalFormat;
102     }
103
104    public ImGuiTexture(string name, int width, int height, IntPtr ←
105        data, bool generateMipmaps = false, bool srgb = false)
106    {
107        Name = name;
108        Width = width;
109        Height = height;
110        InternalFormat = srgb ? Srgb8Alpha8 : ←
111            SizedInternalFormat.Rgba8;
112        MipmapLevels = generateMipmaps == false ? 1 : ←
113            (int)Math.Floor(Math.Log(Math.Max(Width, Height), 2));
114
115        ImGuiUtil.CreateTexture(TextureTarget.Texture2D, Name, out ←
116            GLTexture);
117        GL.TextureStorage2D(GLTexture, MipmapLevels, ←
118            InternalFormat, Width, Height);
119
120        GL.TextureSubImage2D(GLTexture, 0, 0, 0, Width, Height, ←
121            PixelFormat.Bgra, PixelType.UnsignedByte, data);
122
123        if (generateMipmaps) GL.GenerateTextureMipmap(GLTexture);
124
125        SetWrap(TextureCoordinate.S, TextureWrapMode.Repeat);
126        SetWrap(TextureCoordinate.T, TextureWrapMode.Repeat);
127
128        GL.TextureParameter(GLTexture, ←
129            TextureParameterName.TextureMaxLevel, MipmapLevels - 1);
130
131    }
132
133    public void SetMinFilter(TextureMinFilter filter)
134    {
135        GL.TextureParameter(GLTexture, ←
136            TextureParameterName.TextureMinFilter, (int)filter);
137
138    }
139
140    public void SetMagFilter(TextureMagFilter filter)
141    {
142        GL.TextureParameter(GLTexture, ←
143            TextureParameterName.TextureMagFilter, (int)filter);
144
145    }
146
147    public void SetAnisotropy(float level)
148    {
149        const TextureParameterName TEXTURE_MAX_ANISOTROPY = ←
150            (TextureParameterName)0x84FE;
151        GL.TextureParameter(GLTexture, TEXTURE_MAX_ANISOTROPY, ←
152            ImGuiUtil.Clamp(level, 1, MaxAniso));
153
154    }
155
156    public void SetLod(int @base, int min, int max)
157    {
158        GL.TextureParameter(GLTexture, ←
159            TextureParameterName.TextureLodBias, @base);
160        GL.TextureParameter(GLTexture, ←
161            TextureParameterName.TextureMinLod, min);
162        GL.TextureParameter(GLTexture, ←
163            TextureParameterName.TextureMaxLod, max);
164
165    }
166
167    public void SetWrap(TextureCoordinate coord, TextureWrapMode mode)
168    {
169        GL.TextureParameter(GLTexture, (TextureParameterName)coord, ←
170            (int)mode);
171
172    }

```

```
152     public void Dispose()
153     {
154         GL.DeleteTexture(GLTexture);
155     }
156 }
157 }
158 }
```

Listing 1.22 – ./source/csharp/ImGuiTexture.cs

1.2 Shaders

1.2.1 shader.vert

```
1 #version 330 core
2
3 layout(location = 0) in vec2 position;
4
5 uniform mat4 u_Model_To_NDC;
6 uniform mat4 u_Transformation;
7
8 uniform vec2 u_Resolution;
9 uniform vec2 u_Position;
10 uniform vec2 u_Size;
11
12
13 void main()
14 {
15     vec4 finalPosition = vec4(position, 1.0, 1.0);
16
17     vec4 origin = vec4(u_Position.x + u_Size.x/2.0, u_Position.y + ←
18         u_Size.y/2.0, 0.0, 0.0);
19
20     finalPosition -= origin;
21
22     finalPosition = finalPosition * u_Transformation;
23
24     finalPosition += origin;
25
26     finalPosition = u_Model_To_NDC * finalPosition;
27
28     gl_Position = finalPosition;
}
```

Listing 1.23 – ./source/shaders/shader.vert

1.2.2 shader.frag

```
1 #version 330
2
3 layout(location = 0) out vec4 color;
4
5 uniform vec2 u_Resolution;
6 uniform vec4 u_Color;
7
8 void main()
9 {
10     color = u_Color;
11 }
```

Listing 1.24 – ./source/shaders/shader.frag

1.2.3 circle.frag

```
1 #version 330
2
3 layout(location = 0) out vec4 color;
4
5 uniform vec2 u_Resolution;
6 uniform vec4 u_Color;
7 uniform vec2 u_Position;
8 uniform vec2 u_Size;
9 uniform float u_Radius;
10
11 // Draw a circle at a given position with radius and color
```

```

12 vec4 circle(vec2 uv, vec2 pos, float rad, vec4 color) {
13     float d = length(pos - uv) - rad;
14     float t = clamp(d, 0.0, 1.0);
15     return vec4(color.xyz, (1.0 - t) * color.w);
16 }
17
18 void main() {
19
20     vec2 uv = gl_FragCoord.xy;
21
22     float radius = u_Size.x / 2.0;
23
24     vec2 position = vec2(u_Position.x + u_Size.x / 2.0, u_Resolution.y - ←
25         (u_Position.y + u_Size.y / 2.0)); // Invert y axis
26
27     color = circle(uv, position, radius, u_Color);
28 }
```

Listing 1.25 – ./source/shaders/circle.frag

1.2.4 rectangle.frag

```

1 iż&// Author: Amine Sebastian
2 // Source: https://www.shadertoy.com/view/WtdSDs
3 #version 330
4
5 layout(location = 0) out vec4 color;
6
7 uniform vec2 u_Resolution;
8 uniform vec4 u_Color;
9 uniform vec2 u_Position;
10 uniform vec2 u_Size;
11 uniform float u_Radius;
12
13 float roundedBox(vec2 centerPosition, vec2 size, float rad) {
14     return length(max(abs(centerPosition) - size + rad, 0.0)) - rad;
15 }
16
17 void main() {
18     // How soft the edges should be (in pixels). Higher values could be ←
19     // used to simulate a drop shadow.
20     float edgeSoftness = 0.5f;
21
22     // The radius of the corners (in pixels).
23     float radius = clamp(u_Radius, 0.0f, min(u_Size.x, u_Size.y) / 2.0f);
24
25     vec2 position = vec2(u_Position.x, u_Resolution.y - u_Position.y);
26
27     // Calculate distance to edge.
28     float distance = roundedBox(gl_FragCoord.xy - position + ←
29         vec2(-u_Size.x/2.0f, u_Size.y/2.0f), u_Size/2.0f, radius);
30
31     // Smooth the result (free antialiasing).
32     float smoothedAlpha = 1.0f - smoothstep(0.0f, 1.0f, distance);
33
34     // Return the resultant shape.
35     vec4 quadColor = vec4(u_Color.xyz, smoothedAlpha * u_Color.w);
36
37     color = quadColor;
38 }
```

Listing 1.26 – ./source/shaders/rectangle.frag