PROJET INDUSTRIEL
PROMOTION 2010

Projet Industriel N° 64
Noms des élèves :
Pierre CACLIN
Pierre BARBRY-BLOT
Benoît LAVORATA
Vincent MONTAGNE
Aurel-Aimé MARMION
Julien LERAY
Pierre-Antoine MARC
Commanditaire :
Équipe de développement d'Inkscape
Tuteur industriel :
Cédric GEMY
Tuteur ECL :
René CHALON
Département d'accueil :
Informatique
Date du rapport :
18 mai 2009

Rapport final

Inkscape
Implémentation de l'outil Spray

Résumé

Le groupe de Projet Industriel 64 avait pour mission de concevoir et d'implémenter un outil de projection de formes graphiques ou Spray-tool dans le logiciel libre Inkscape. Le cahier des charges disponible sur le site internet d'Inkscape demandait l'implémentation de fonctions bien particulières qui manquaient à l'environnement de création, pourtant très complet, que proposait le logiciel.

Le cahier des charges étant particulièrement fourni, des choix ont été faits par rapport aux différentes fonctions à implémenter. Cela, à la fois pour parvenir à réaliser le travail dans les temps, mais aussi pour soumettre un outil stable au terme des 5 mois. Certaines fonctions étaient donc prioritaires par rapport à d'autres. De nombreux tests ont d'ailleurs été effectués pour s'assurer de la stabilité de l'outil réalisé.

Le projet à été mené à terme et sera implémenté dans une prochaine version officielle qui sera proposé en téléchargement gratuit. L'outil est utilisable, stable et satisfait le cahier des charges mais reste optimisable pour certaines fonctions. Cela pourra servir de base à d'autres projets d'amélioration de l'outil.

Le présent rapport retrace l'évolution de ce projet dans son ensemble, en mettant en exergue les points importants du développement et les fonctionnalités implémentées.

Table des matières

Introduction

Au cours des cinq derniers mois, nous avons travaillé sur le logiciel de dessin vectoriel Inkscape. Ce logiciel a la particularité d'être libre, c'est-à-dire que chacun peut contribuer à son développement. De ce fait, nous n'avions pas de commandiaire industriel à proprement parler. Par conséquent, notre travail doit avant tout être considéré comme une contribution au développement d'un logiciel open-source.

L'objectif du projet est l'implémentation d'une nouvelle fonction, attendue depuis un moment par la communauté d'utilisateurs : un outil de projection de formes, également appelé outil spray.

Ce projet poursuit la voie lancée par Steren Giannini, qui était à l'origine d'un projet similaire l'année dernière. Étant pour la plupart familier avec le monde de la création graphique, c'est avec enthousiasme que nous avons accepté le défi proposé par ce projet.

Notre rapport présente la mise en place du projet, ses objectifs et le travail effectué pour le réaliser.

Chapitre 1

Le projet

1.1 Contexte

1.1.1 Qu'est-ce que Inkscape ?

Inkscape est un logiciel de dessin vectoriel libre, c'est-à-dire que chacun peut accéder à son code et ainsi contribuer à son développement. C'est un logiciel multiplate-formes, disponible sous Windows, Linux ou encore Mac OS X. Il est entièrement conforme avec les standards XML, SVG [1] et CSS du W3C [2].

Inkscape est programmé en C++ et utilise la bibliothèque graphique GTK+. Il utilise des objets définis par des courbes et des points, et non pas des matrices de pixels. Le standard SVG est le format de référence d'Inkscape. C'est un format vectoriel très répandu, dans les applications mobiles par exemple, mais également utilisé en cartographie.

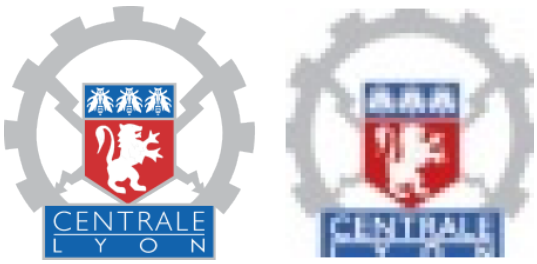

FIG. 1.1 – Différence entre une image SVG (à gauche) et une image en pixels (à droite)

Ce format permet de gérer des formes géométriques de base (comme des ellipses, des rectangles par exemple), mais il permet également de gérer des formes définies par des courbes de Bézier. On peut ainsi obtenir presque n'importe quelle forme. Ces formes peuvent se remplir par des dégradés, des couleurs de motifs ou des filtres.

[1] Scalable Vector Graphics
[2] World Wide Web Contortium

Voici un exemple de carte réalisée sous Inkscape :

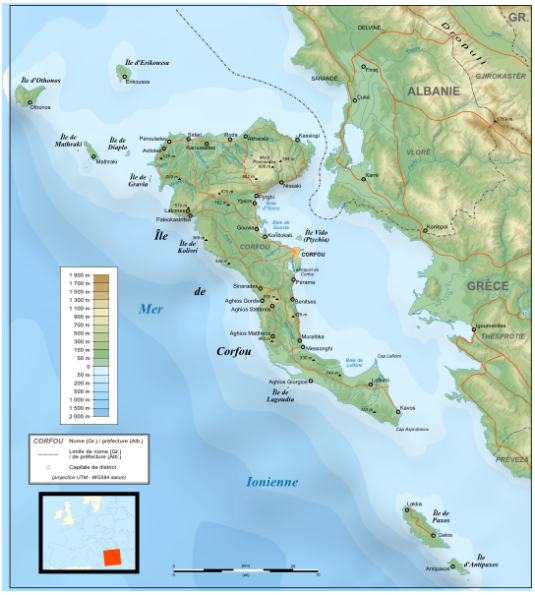

FIG. 1.2 – Carte de l'île de Corfou réalisée sous Inkscape

1.1.2 Les principales fonctions d'Inkscape

Voici un rapide tour d'horizon des fonctionnalités d'Inkscape.

- *Création d'objets et de formes*
  - Rectangles
  - Ellipses
  - Étoiles, polygones
  - Clones (objets copiés avec modification dynamique)
- *Manipulation d'objets*
  - Redimensionner, incliner, tourner...
  - Copier, coller
  - Alignement et distribution d'objets
  - Groupements d'objets
- *Remplissage et contour des formes*
  - Sélecteur de couleurs
  - Éditeur de dégradés
  - Bordures pointillées
  - Pipette à couleurs
- *Outils de dessin*
  - Lignes à main levée
  - Courbes de Bézier
  - Courbes de Spiro

Il existe encore beaucoup d'autres fonctionnalités dont nous ne ferons pas mention dans ce rapport.

1.2 La naissance du projet

Le projet de développement de cet outil nous a été proposé par Steren Giannini, qui a travaillé l'année dernière sur un projet similaire. Cet outil fait partie des tâches de développement d'Inkscape et il a décidé de le proposer en projet industriel.

Notre tuteur ECL est René Chalon, professeur au département Informatique de l'école.

Notre tuteur de projet est Cédric Gémy, responsable de la filière informatique en sciences de l'Éducation à l'Université Rennes 2, et également graphiste indépendant. Il est très familier avec le monde des logiciels libres, en particulier Inkscape.

1.3 Nos objectifs

L'objectif de ce projet est d'implémenter un outil permettant de projeter tous types de formes de façon aléatoire. Cet outil n'existe pas actuellement sur Inkscape en tant que tel, même si on peut y parvenir par des moyens détournés. On doit donc pouvoir projeter un grand nombre de formes de façon simple, par le biais de différents modes de duplication. Le cahier des charges précis de l'outil est détaillé dans la partie suivante.

Chapitre 2

Cahier des charges

En tant que logiciel libre, n'importe qui peut apporter ses modifications au code. Toutefois, on ne peut pas faire n'importe quoi, et des tâches de développement sont mises à disposition à qui veut apporter sa contribution. De la même manière, chacun peut proposer de nouvelles idées de développement. Sur un tel projet open-source, quelques personnes s'occupent de gérer les différentes contributions, et se chargent de donner des directives de développement. Notre outil spray répondait ainsi à un cahier des charges précis (appelé *Blueprint*) dont nous nous sommes inspirés pour développer.

2.1 Aperçu général des fonctionnalités envisagées

Voici une représentation de l'outil, telle qu'elle nous a été fournie par l'équipe de développement. On note qu'elle comporte déjà une liste conséquente de fonctionnalités que l'on souhaiterait implémenter dans l'outil. Bien évidemment, il a fallu cibler clairement ce qu'on était en mesure de réaliser, indépendamment de la volonté de la communauté Inkscape.

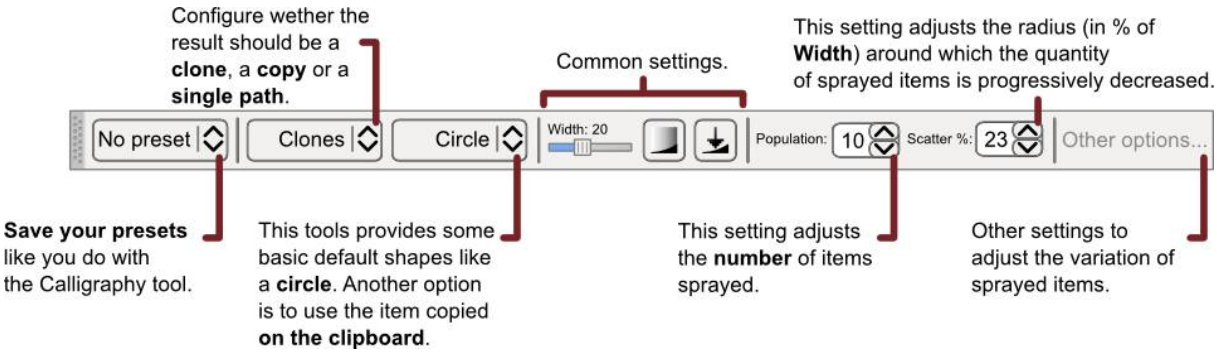

FIG. 2.1 – Interface du cahier des charges

Le Blueprint propose trois modes fondamentaux de spray : copie (copy), clone (clone) et fusion (single path) que nous expliquerons dans la suite.

Un autre bouton permet de faire un choix rapide sur la forme à projeter : un cercle, un carré ou d'autres formes élémentaires. On peut aussi créer une forme et la sauvegarder en tant que *preset* [1] afin de la réutiliser plus tard.

Ensuite viennent les réglages standards qui paramètrent la zone de projection et la répartition des formes produites.

[1] paramètres prédéfinis

2.2 Aperçu détaillé des réglages fins

A cette barre d'outils globale s'ajoute une boite de dialogue plus complexe « Other Options » permettant de régler plus présisément les paramètres du spray.

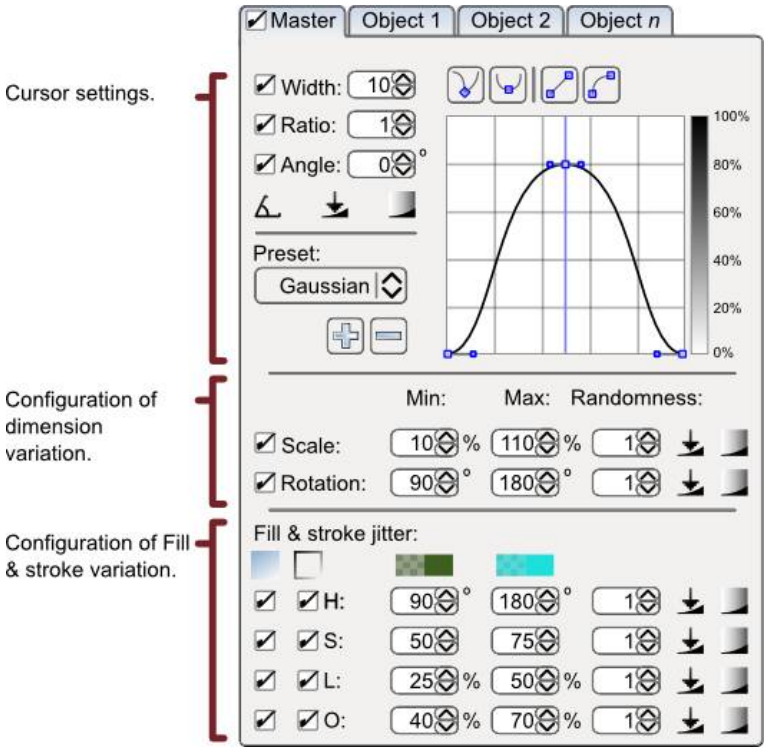

FIG. 2.2 – Interface avancée

On note la présence d'une courbe de densité. Son profil est modifiable en choisissant différentes distributions, mais aussi en ajustant précisemment à la souris sur le dessin. En dessous se trouvent les réglages de la teinte, la saturation, la luminosité et l'opacité. Cette zone de réglage n'a pas été implémentée dans son intégralité.

2.3 Différents modes de projection

Il y a différentes manières de projeter des formes. Le mode par copie permet de répéter la forme originelle et ensuite agir sur chaque forme indépendamment des autres. En revanche, en mode clone, une modification de la forme de départ (qu'on appelle le *père*) entraine la modification de toutes les autres. D'un point de vue mémoire, ce mode de projection est beaucoup plus léger. Enfin, le mode fusion unit les formes au fur et à mesure de leur projection sur l'écran. Ces modes ont tous été implémentés dans l'outil.

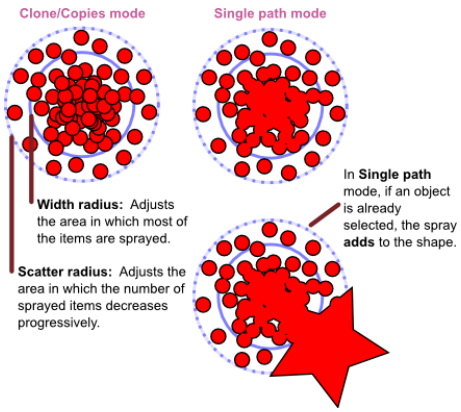

FIG. 2.3 – Les différents modes de projection envisagés

2.4 Réglages de la zone de projection

Afin d'offrir plus de libertés que le simple cercle de projection, il est possible de paramétrer la forme du curseur. Cette fonction a été implémentée.

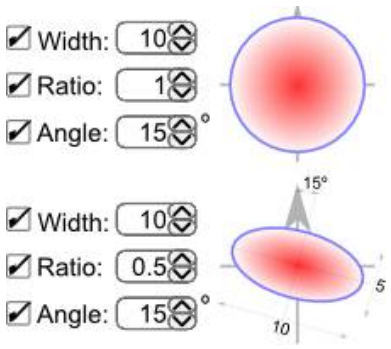

FIG. 2.4 – Allure de la zone à projeter

2.5 Répartition des objets projetés

On doit pouvoir choisir la répartition des objets indépendamment les uns des autres. Par exemple, si on a un carré et une étoile, on voudrait que la répartition des carrés soit différente de celle des étoiles. Cette fonction n'a pas été implémentée pour l'instant.

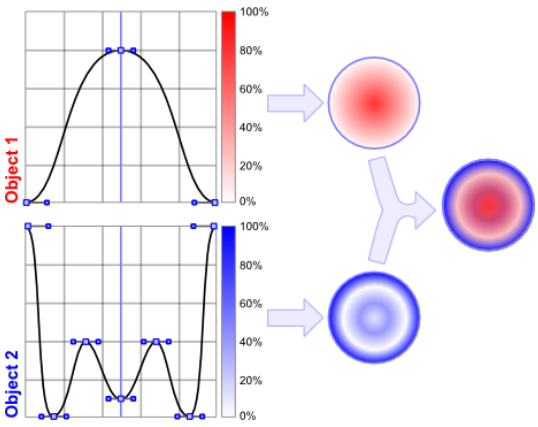

FIG. 2.5 – Allure de la zone à projeter

Chapitre 3

L'outil Spray

3.1 Fonctionnalités

Conformément au cadre de ce projet, nous avons essayé, du moins dans un premier temps, de suivre le *blueprint*. Cela nous a servi de base pour déterminer les fonctionnalités de notre outil. En effet, les fonctionnalités envisagées lors de la phase de conception (cahier des charges) ont évolué du fait de deux facteurs liés : le temps imparti et nos compétences en programmation. Il faut savoir qu'on trouve très peu de documentation sur le code source, et que les premières semaines ont été entièrement consacrées à le déchiffrer.

3.1.1 Les différents modes de duplication

Une des principales fonctionnalités qui nous a été demandée est de pouvoir choisir les modes de duplication. Comme nous l'avons vu précédemment, il existe trois modes de duplication : la copie, le clone et la fusion.

La *copie* est le premier mode qui vient à l'esprit. L'objet sélectionné est simplement copié dans son intégralité. Les objets sprayés sont alors indépendants les uns des autres et les modifications (par exemple sur la couleur, la taille ou la rotation) portées sur un objet n'ont aucune influence sur les autres.

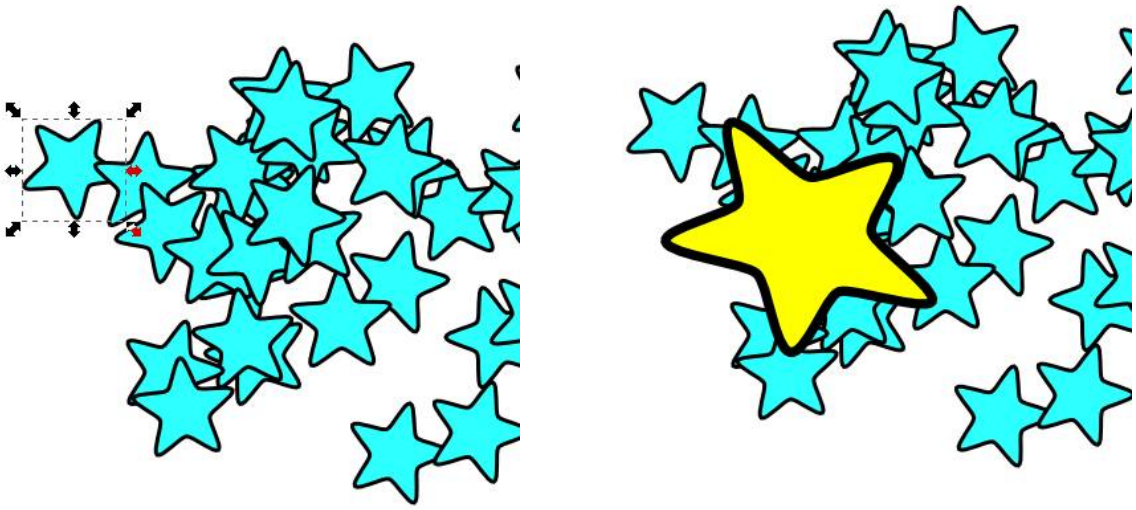

FIG. 3.1 – Objet sprayé en mode copie puis modifié (les autres demeurent inchangés)

Le mode *clone* permet de lier l'objet père sprayé à l'ensemble de ses fils. Ainsi, la modification du père entraine la même modification des objets fils.

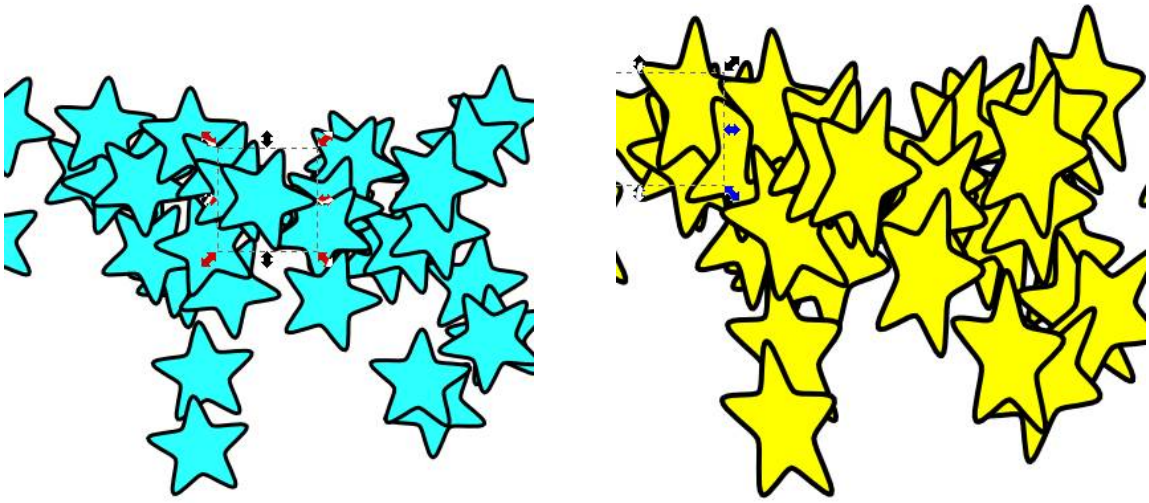

FIG. 3.2 – Objet sprayé en mode clone puis modifié (ici : forme et couleur)

Un dernier mode, le mode *fusion*, unifie les contours des objets sprayés. L'objet initial, de même que tous ses fils n'existent pas en tant que tels mais sont unis en un seul et même objet. Une modification d'échelle ou de couleur est ainsi répercutée de manière globale.

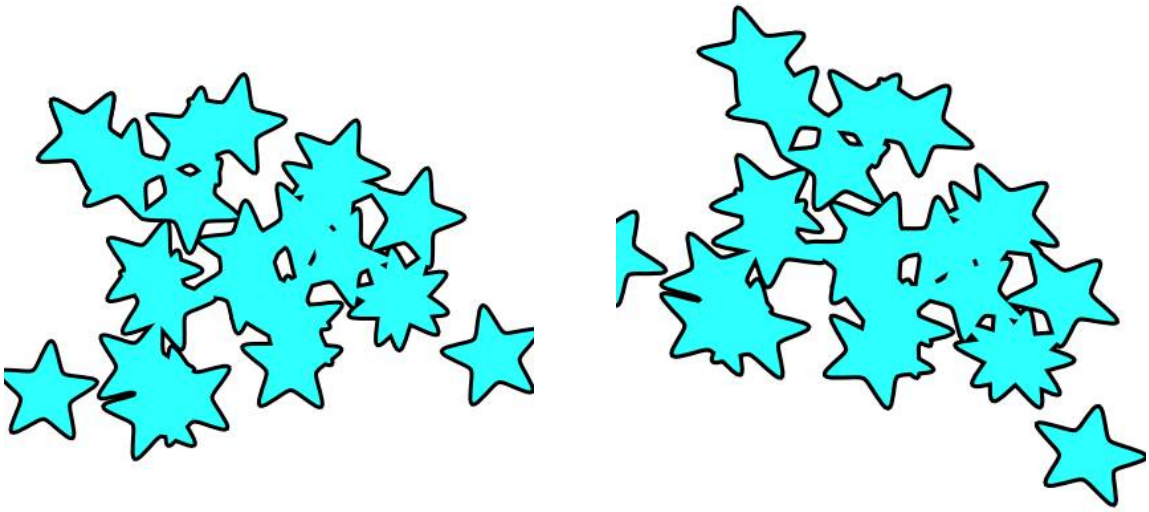

FIG. 3.3 – Objet sprayé en mode fusion puis modifié

3.1.2 Les modes de répartition aléatoire

L'outil spray permet de contrôler le profil de répartition des formes à l'intérieur du curseur. Nous avons adopté une répartition gaussienne qui nous apparaissait comme la plus intuitive. La formule théorique de la répartition sur un profil est donnée par :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

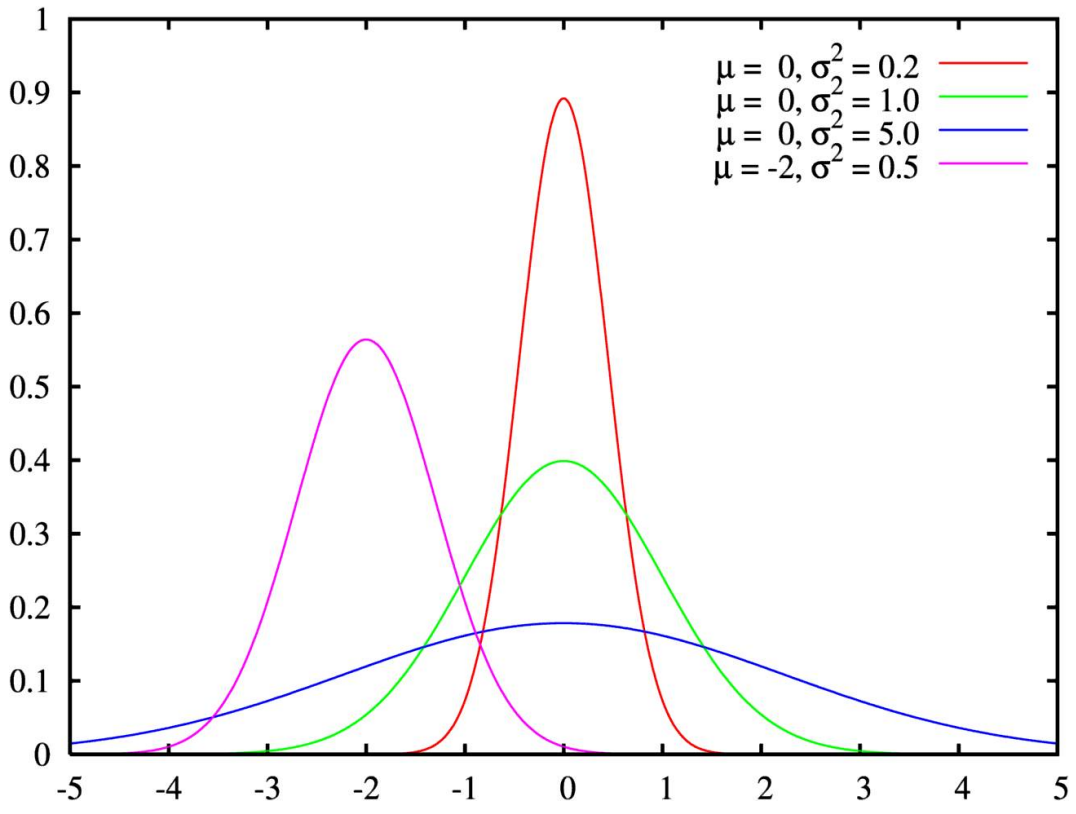

FIG. 3.4 – Gaussienne

Les paramètres importants sur lesquels on peut influer sont : la moyenne (μ) et l'écart-type (σ). Ces paramètres sont appelés dans l'interface par le biais de curseurs appelés mean (moyenne) et SD (*standard deviation* pour « écart-type »).

FIG. 3.5 – Influence de la *standard deviation* sur la répartition (à taille de curseur constante)

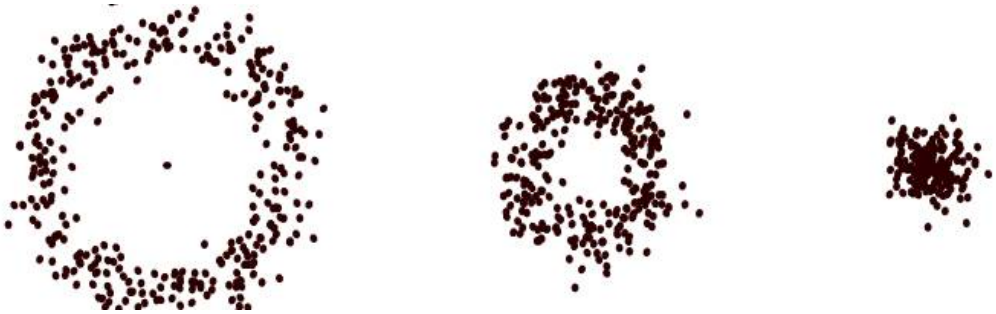

FIG. 3.6 – Influence de la moyenne sur la répartition (à taille de curseur constante)

3.1.3 Modification aléatoire des caractéristiques

Lors de l'utilisation de l'outil spray, on pourra choisir de modifier plusieurs paramètres de la forme servant de modèle, de manière aléatoire. Les options de l'outil permettant de configurer ces modifications se trouvent dans le panneau latéral que nous avons implémenté.

Rotation aléatoire des formes

Il est ainsi possible d'appliquer une rotation aléatoire aux formes sprayées. On définit pour cela les bornes supérieure et inférieure de l'angle de rotation, en degrés, dans le panneau latéral de commande. Chaque forme sprayée est alors tournée d'un angle aléatoire compris entre ces deux valeurs.

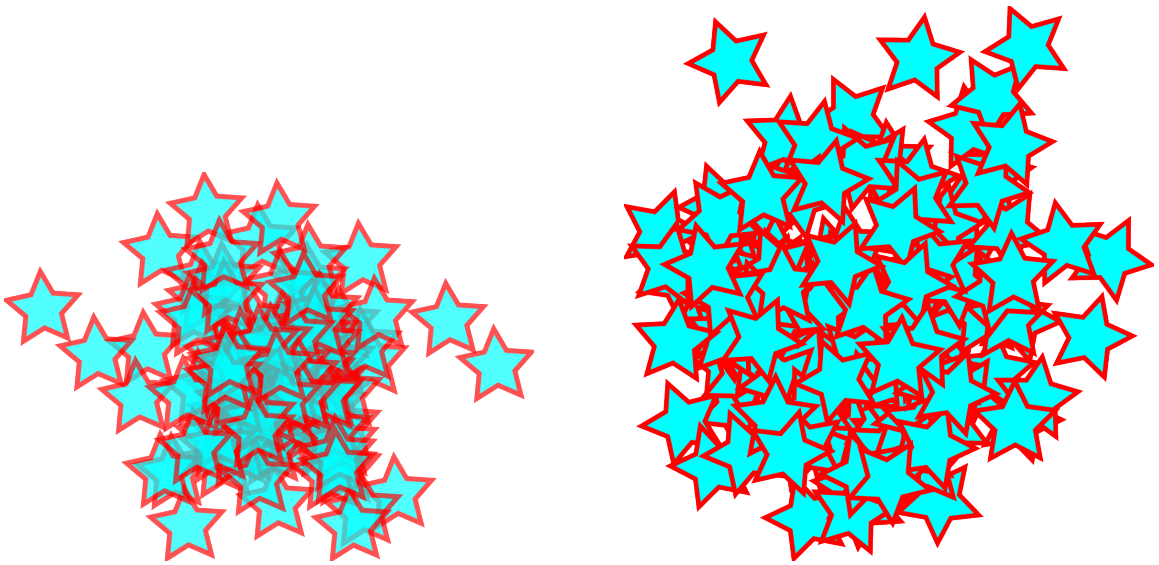

FIG. 3.7 – Rotation nulle puis rotation aléatoire des formes

Modification aléatoire de la taille

De la même manière, nous avons implémenté un système de modification aléatoire de la taille des objets sprayés. Toujours, dans le panneau latéral, on peut ainsi fixer les valeurs maximale et minimale de gain en taille, R_{min} et R_{max} : concrètement, la taille de chaque forme sprayée sera comprise entre R_{min} et R_{max} fois la taille du « modèle » .

FIG. 3.8 – Modification aléatoire de la taille

Modification progressive de la taille

Nous avons aussi ajouté la possibilité de modifier la taille des formes de manière maitrisée grace aux raccourcis claviers *Spray+Flèche-Haut* (augmentation de la taille) et *Spray+Flèche-Bas* (diminution de la taille), ce qui permet d'obtenir des effets de dégradé.

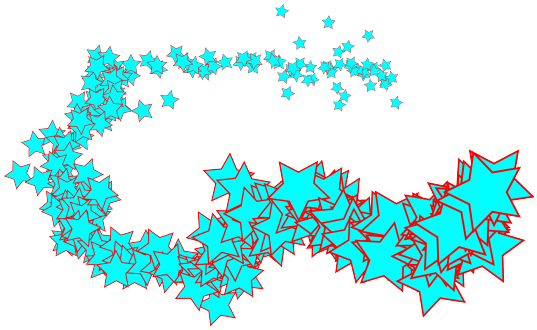

FIG. 3.9 – Modification progressive de la taille

Flux d'objets projetés

On peut enfin gérer le flux d'objets projetés par l'intermédiaire d'une variable *population* que l'on peut incrémenter ou décrémenter. Cette fonction doit pouvoir encore être optimisée à l'avenir.

3.1.4 La forme paramétrable du curseur

Afin d'offrir encore plus de possibilités dans la manière de sprayer, en accord avec le cahier des charges, la forme du curseur est paramétrable. Le curseur que nous proposons est une ellipse, caractérisée par sa taille (*width*), son excentricité (*ratio*) et son orientation (*angle*).

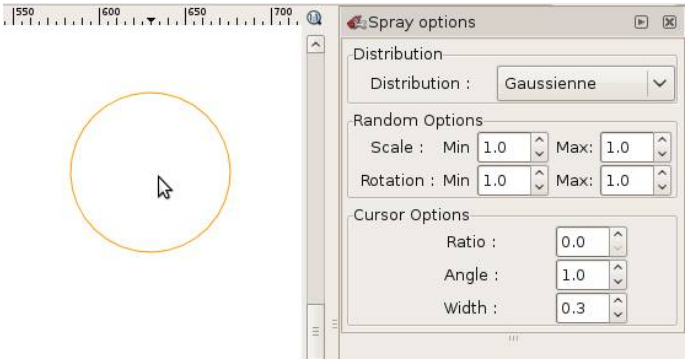

FIG. 3.10 – Modification de la forme du curseur

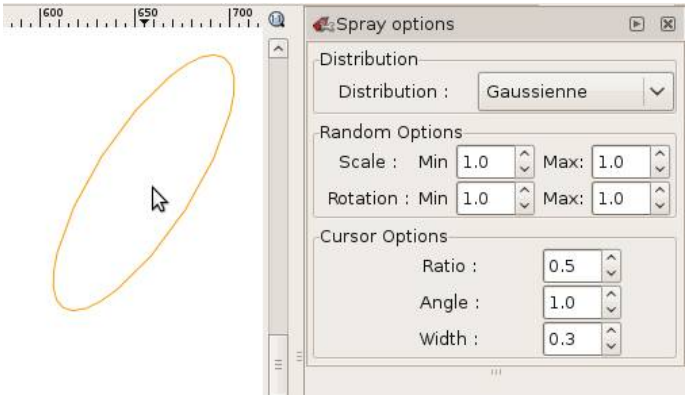

FIG. 3.11 – Modification de la forme du curseur

3.1.5 Exemples d'utilisation

Un bon outil graphique est un outil qui permet à la fois une prise en main rapide, des fonctionnalités performantes et la possibilité de prendre plus de libertés avec l'expérience. C'est dans cette optique que nous avons cherché à rendre l'outil le plus intuitif et maniable, adapté à plusieurs profils d'utilisateurs : novice ou experimenté, voire professionnel. Les images suivantes présentent brièvement les possibilités offertes par notre outil.

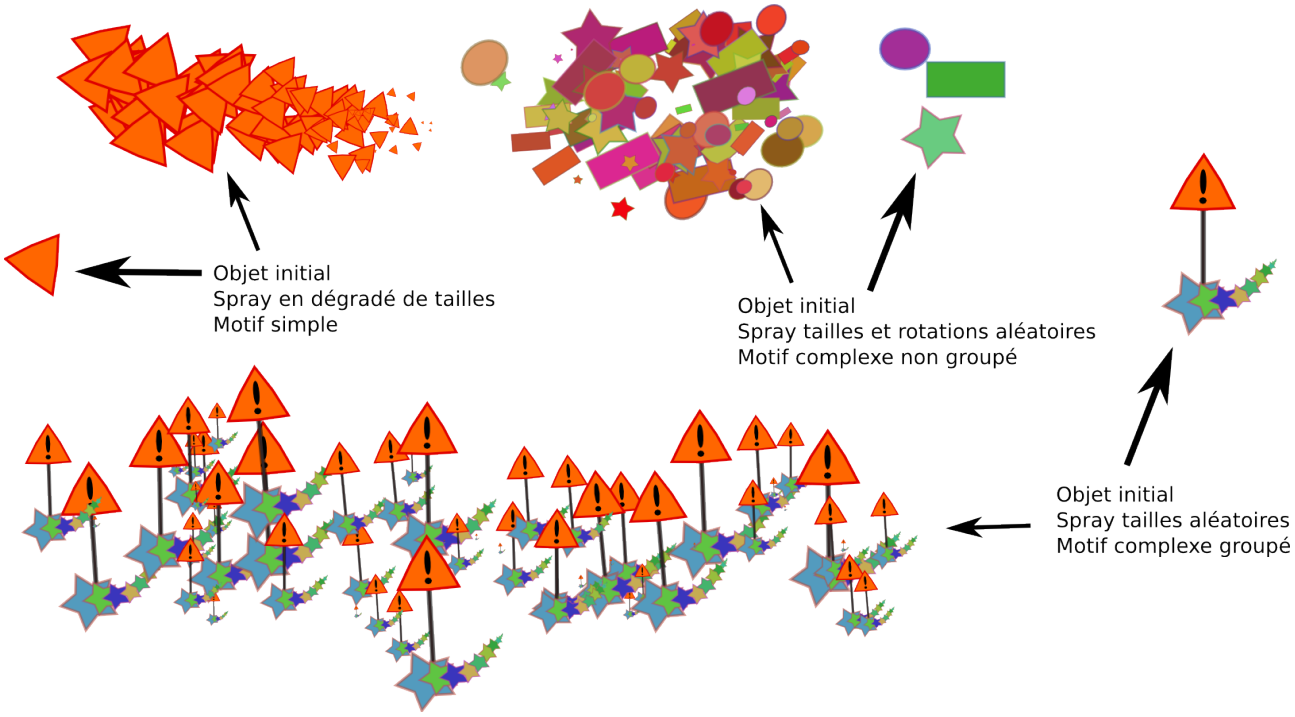

FIG. 3.12 – Projection d'objets simples et de groupes

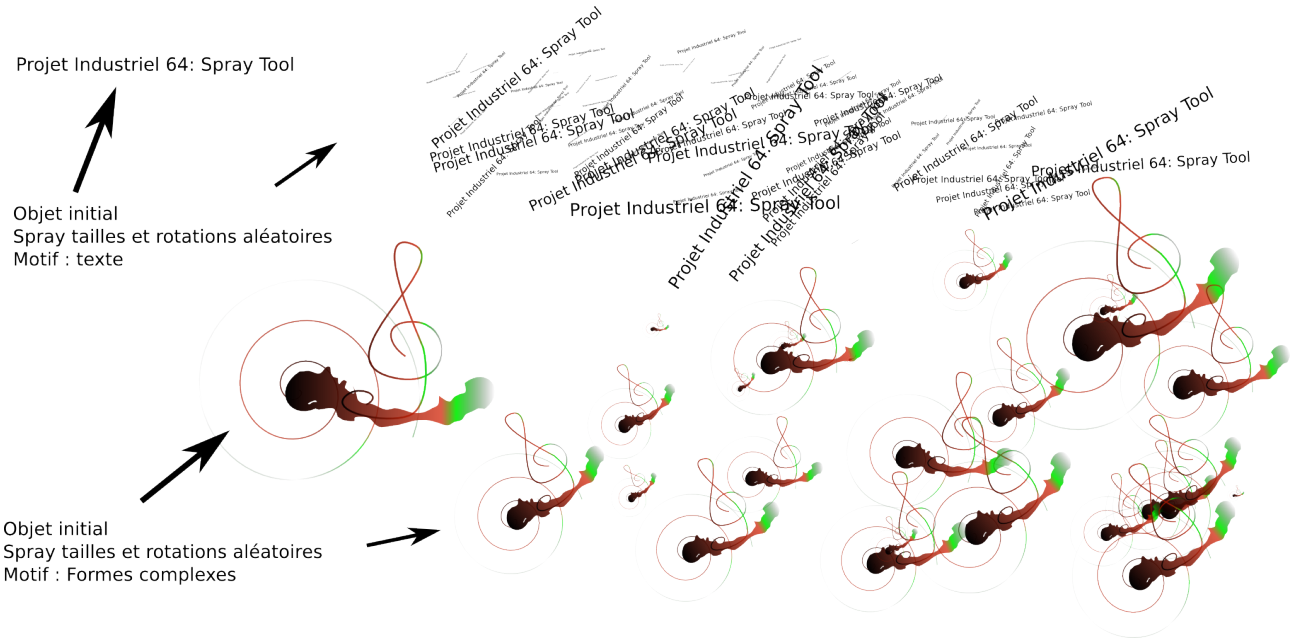

FIG. 3.13 – Projection d'objets complexes et de texte

Projection de motifs

Le Spray-tool permet la projection simultanée de plusieurs formes sélectionnées de deux manières, à savoir indépendamment les unes des autres (exemple des étoiles, rectangles et ellipses sur la première image) ou de manière identique à la selection si celle-ci est groupée (exemple du panneau de signalisation).

Projection de texte et d'objets complexes

Le Spray-tool offre aussi la possibilité de projeter des éléments textuels et de leur appliquer les mêmes transformations qu'aux formes classiques. De plus, la projection de formes complexes réalisées par d'autres outils est possible et est d'autant mieux réalisée par la fonction de projection par clones qui stocke moins de données en mémoire.

3.2 Limites et pistes de développement

Dès le début du projet, il était probable que nous n'aurions pas le temps de nous approprier pleinement le code source de Inkscape et de l'appréhender dans sa globalité. Il en résulte que quelques unes des fonctionnalités prévues par le *blueprint* fourni par l'équipe de développement n'ont pas pu être développées, notamment les *presets* (choix des utilisateurs) et le choix de quelques formes par défaut.

Par exemple, et comme cela a été mentionné plus haut, nous n'avons implémenté que deux modes de répartition, *uniforme* et *gaussien*. Le *Blueprint* semblait suggérer une possibilité de fixer « à la main » la répartition des formes, via une courbe entièrement paramétrable par l'utilisateur. Or, s'il est possible de fixer arbitrairement la répartition par une fonction mathématique, cette modification doit pour le moment être effectuée directement dans le code source, qui doit alors être recompilé pour que les changements soient effectifs. Lors de nos tests, il a été possible d'implémenter d'autres fonctions de répartition. Nous n'avons pas conservé ces fonctions, car elles n'apportaient finalement pas grand chose pour l'utilisateur. Cependant, il va être intéressant de suivre l'opinion de la communauté et de voir comment pourra être amélioré l'outil : c'est là tout l'intérêt d'un développement open-source.

Enfin, comme nous allons le voir par la suite, l'outil *Tweak* nous a inspiré certaines fonctionnalités qu'il aurait pu être intéressant d'implémenter. Ainsi, nous aurions aimé pouvoir *modifier*

la couleur des formes projetées, un peu à la manière des options de *décalage des couleurs* de l'outil *Tweak*.

Chapitre 4

Organisation et développement de l'outil

Novices dans le domaine de la programmation, nous avons rapidement cherché à récupérer un maximum de fonctions déjà présentes afin de limiter l'apparition de bugs. En effet, un des handicaps majeurs de la phase de développement fut la faible connaissance du code source et la faible documentation disponible sur les modules que nous avons ré-utilisés.

Cependant, en naviguant à travers l'interface d'Inkscape, nous avons pu remonter aux fonctions essentielles que nous voulions regrouper ou modifier pour notre outil. Ainsi notre point de départ fut un outil existant, *Tweak*, dont nous avons copié l'architecture afin de l'utiliser pour notre propre outil. Les fonctionnalités empruntées à cet outil sont l'objet d'une brève présentation car elles expliquent la logique de programmation que nous avons adoptée. Dans un deuxième temps, on verra comment le travail a été organisé au sein du groupe de travail et enfin la logique qui se cache derrière notre outil.

4.1 L'outil Tweak : la base de l'outil Spray

L'outil Tweak, symbolisé par une icone en forme de vague dans Inkscape, propose des fonctionnalités proches de celles demandées dans notre cahier des charges. En effet, il permet la duplication de formes à partir d'une selection, la répartition aléatoire d'une sélection d'objets autour du curseur et le regroupement des formes à l'endroit ou pointe le curseur. Ces trois fonctions sont essentielles pour notre démarche et nous avons commencé à les copier puis à les modifier pour en comprendre la logique, et notamment la géométrie cachée par le code.

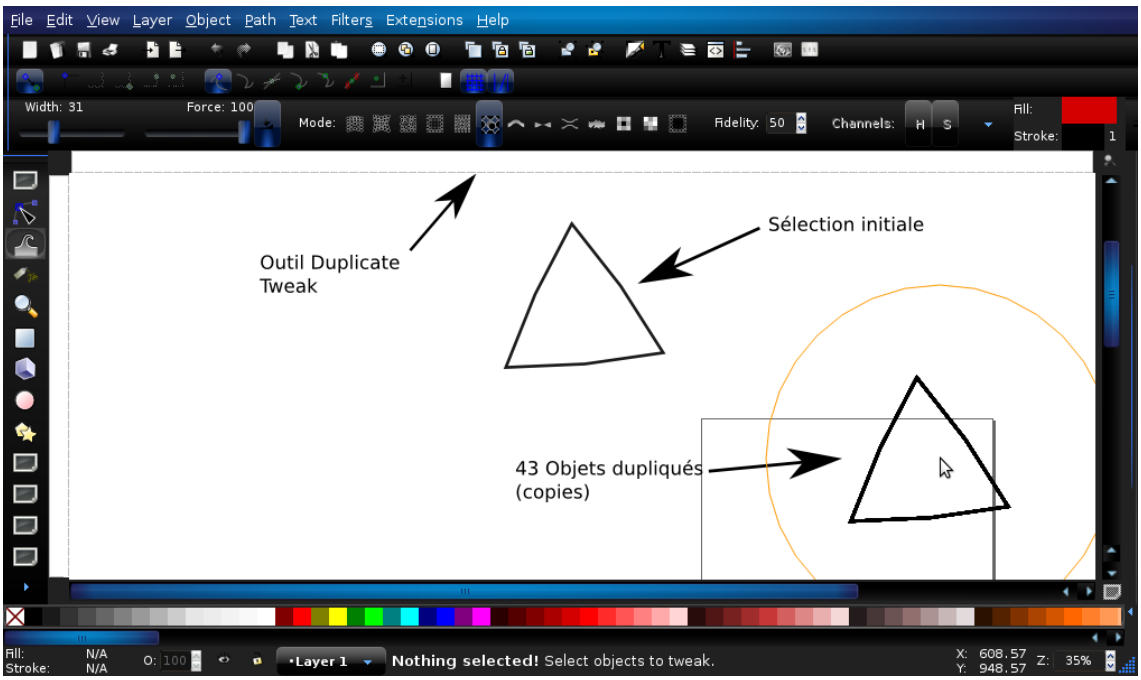

FIG. 4.1 – Tweak, outil duplicate

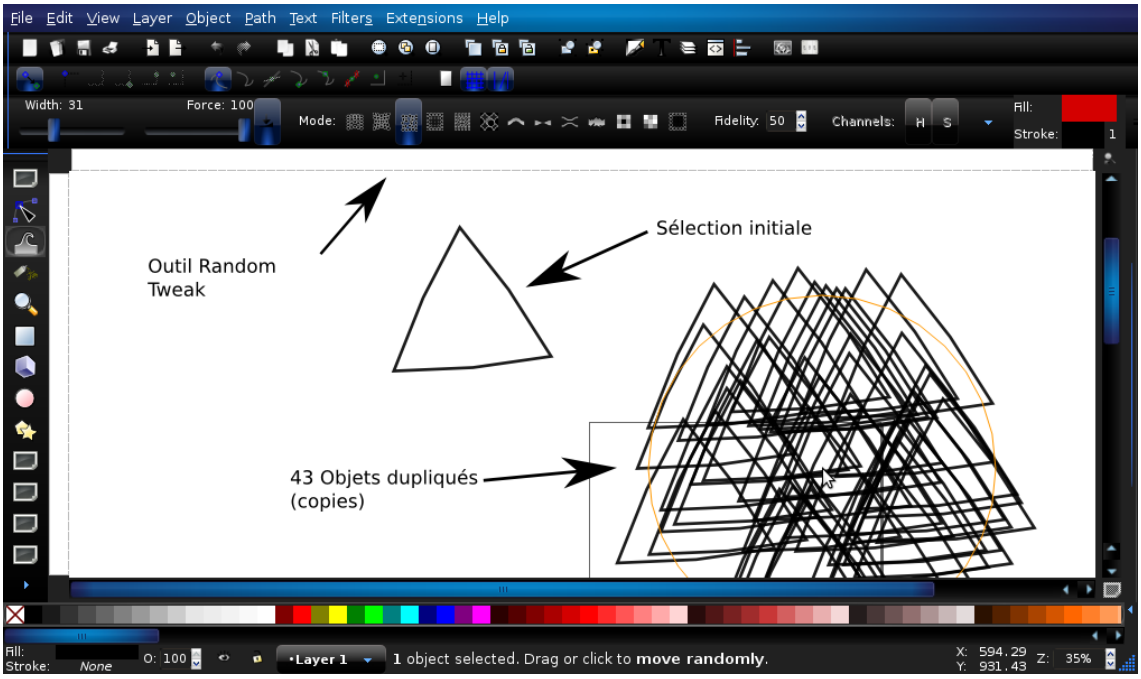

FIG. 4.2 – Tweak, outil random

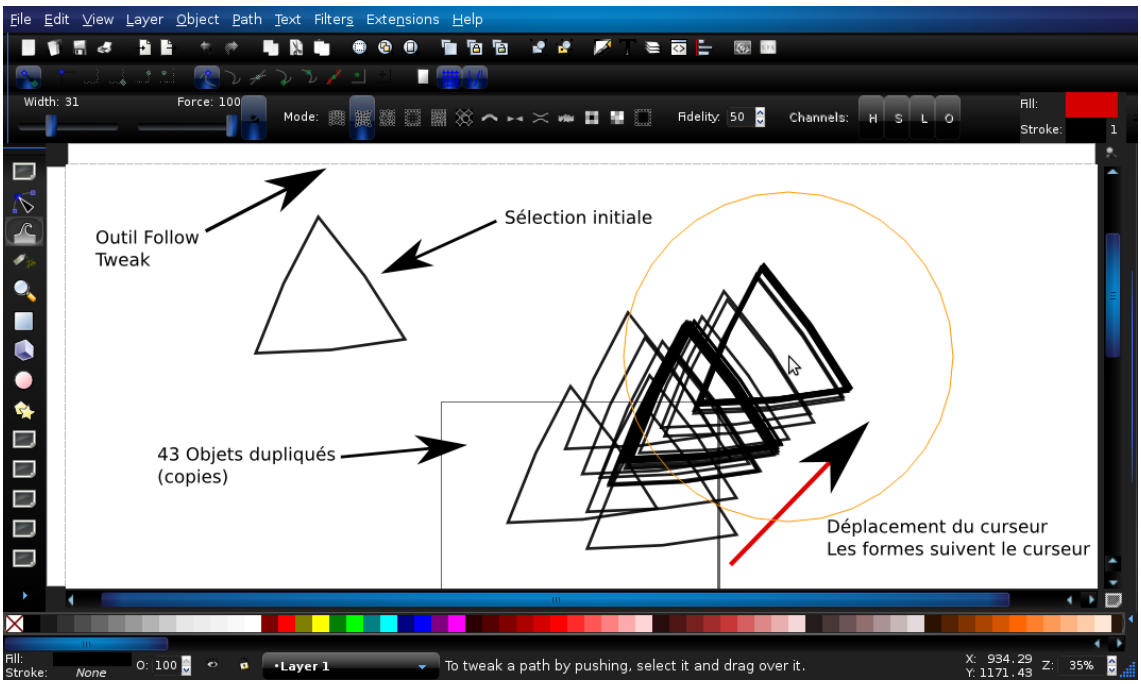

FIG. 4.3 – Tweak, outil follow

Dans un premier temps, nous avons intégralement copié l'outil Tweak dans Inkscape afin de pouvoir comprendre les dépendances de l'outil dans l'architecture globale du logiciel. Au final, nous avons obtenu un outil à l'architecture semblable mais aux fonctions améliorées (rendant certaines fonctions de Tweak obsolètes) qui profite de la stabilité d'un outil déjà debogué par les développeurs du logiciel.

4.2 Organisation et développement

4.2.1 Organisation globale du travail

Dans ce genre de travaux, la mise en place d'équipes de travail est fondamentale afin de pouvoir avancer sur plusieurs fronts sans géner le travail des autres. On a donc opté pour une répartition duale du travail, l'interface d'un coté et le noyau fonctionnel de l'autre, et au sein de chaque équipe nous avons défini des mini-objectifs à atteindre pour chaque membre. Ainsi, l'autonomie de chacun était ainsi assurée par la définition en amont des objectifs et l'avancement global du système plus rapide. Afin de s'assurer du fonctionnement de notre organisation, nous avons fixé des points de validation du travail que nous avons essayé de respecter au maximum. Enfin, nous avons mobilisé notre groupe de projet industriel pendant une semaine de vacances autour de l'outil Spray pour deboguer et finir notre outil.

4.2.2 Les équipes de travail

Interface : Implémentation de l'interface utilisateur, liaison avec le code fonctionnel et aspect visuel.

- *Vincent Montagné*
- *Aurel-Aimé Marmion*
- *Pierre Caclin*

Noyau fonctionnel : Implémentation du code fonctionnel et deboguage.

- *Pierre-Antoine Marc*
- *Benoît Lavorata*
- *Julien Leray*
- *Pierre Barbry Blot*

4.3 Interface

La programmation d'une interface graphique était quelque chose de relativement nouveau pour nous. Ainsi, il a été nécessaire de commencer par nous documenter sur les bibliothèques utilisées, notamment GTK et son pendant C++, GTKmm. Heureusement, de nombreux tutoriels sont disponibles sur internet, et il nous a été facile de prendre un premier contact avec les objets utilisés.

4.3.1 Barre d'options

Nous avons fonctionné globalement de la même manière que l'équipe du noyau fonctionnel : par mimiétisme et récupération de fonctions existantes. Une première phase a donc consisté à dupliquer intégralement l'outil *Tweak*, et notamment l'interface. Par la suite, nous avons épuré le code source de la version dupliquée en conservant uniquement les options de l'interface qui allaient nous être utiles par la suite (duplication d'objets, notamment).

Dans un second temps, on s'est intéressés au développement des autres items de l'interface, en s'inspirant au maximum du cahier des charges. Dans certains cas, on a pu récupérer directement la syntaxe provenant d'interfaces similaires déjà implémentées dans le logiciel, mais dans la plupart des cas une adaptation non-négligeable a été nécessaire. Un des points les plus critiques a été l'identification du lien avec la partie purement fonctionnelle, et la détermination des variables à faire renvoyer.

FIG. 4.4 – Barre d'options de l'outil spray

4.3.2 Panneau latéral

Afin d'obtenir une interface complète, mais néanmoins compacte, nous avons opté pour la solution proposée par le cahier des charges, et complété la barre d'outil par un panneau latéral. Si la première propose les options principales permettant de paramétrer l'outil de façon concise, le panneau, lui, contient les options supplémentaires configurant l'outil de manière plus complète : paramètres du curseur, distribution, etc...

À l'instar de la barre d'options, le panneau a été implémenté en reprenant le code correspondant à l'interface d'un outil existant, *align and distribute*. En effet, cette fonctionnalité d'Inkscape présente une interface fortement similaire à celle que nous avons mise en place.

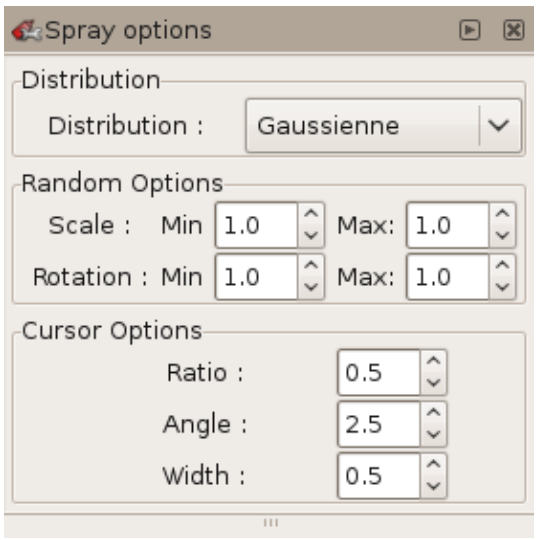

FIG. 4.5 – Panneau latéral de l'outil spray

4.4 Noyau fonctionnel

4.4.1 Architecture générale de notre outil

Le programme Inkscape regroupe près de 10 000 fichiers qui sont bien souvent liés entre eux. Les actions résultant du clic de la souris et aboutissant à la forme projetée font intervenir de nombreux programmes, mais le schéma qui suit ne retiendra que les plus importants. Son but est de faire comprendre rapidement au lecteur les mécanismes du Spray.

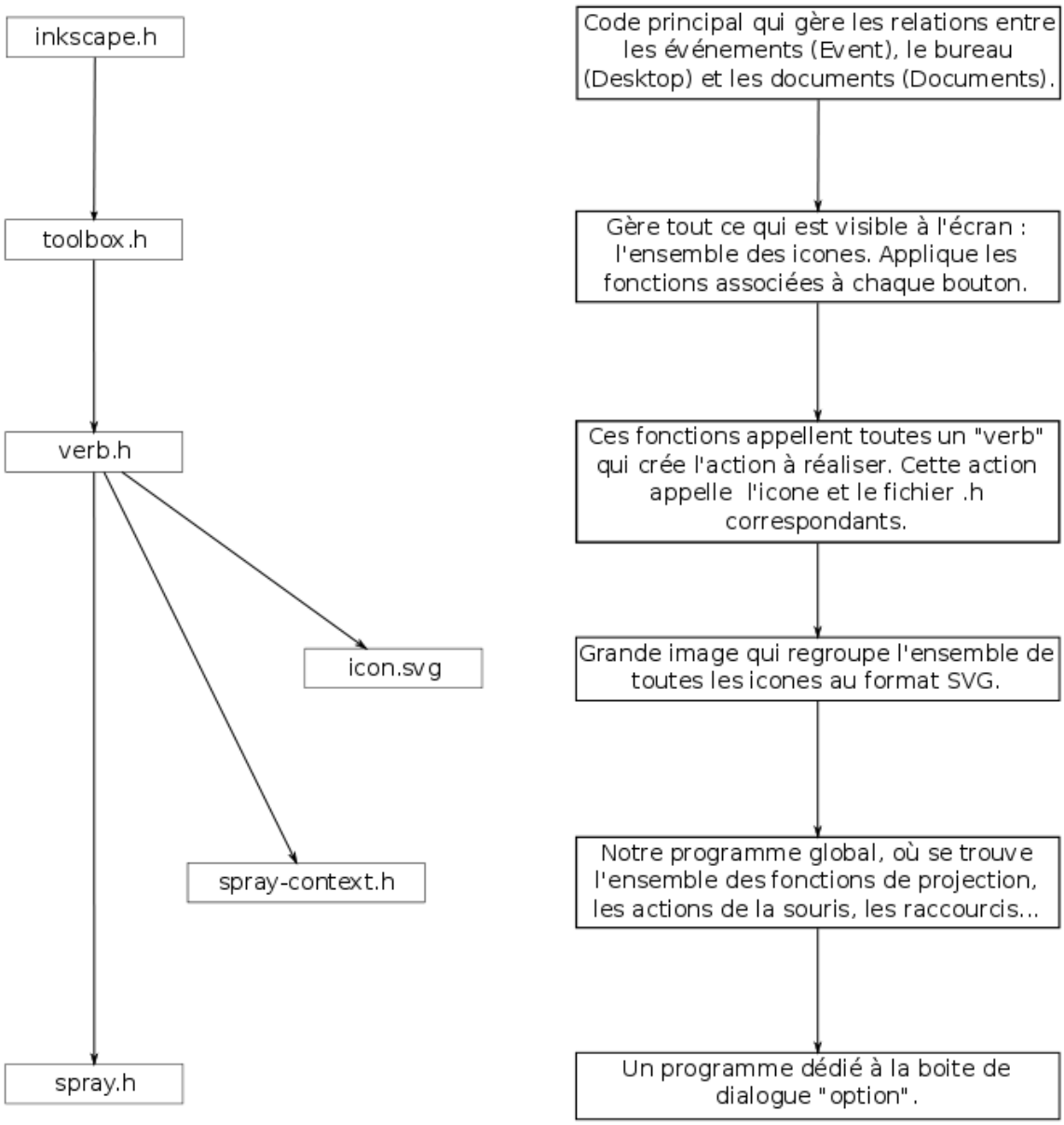

FIG. 4.6 – Architecture globale du programme

Le développement du programme s'est orienté autour de 3 modes d'utilisation distincts : le spray par copie, le spray par clone et le spray par union des formes. Le développement de ces 3 modes sera retracé par la suite en respectant la chronologie de notre projet.

4.4.2 Les différents modes de projection

Le lecteur pourra consulter dans les annexes les extraits de code relatifs aux 3 modes d'utilisation. Ceux-ci pourront permettre de mieux comprendre la logique de notre programme ainsi que l'architecture de chacun des 3 sous-programmes, dont les descriptions sont faites ci-dessous.

Spray par copie

Ce mode d'utilisation fut le plus facile à implémenter puisqu'il était, d'une part, celui présentant l'architecture la plus simple, et d'autre part présentait de grandes similitudes avec l'une des options de l'outil *Tweak*. Nous avons ainsi pu démarrer l'implémentation de notre outil en nous basant sur le code de l'outil *Tweak*, que nous avons remanié pour finalement le changer complétement. Néanmoins, au commencement de notre projet, l'option de spray par copie était une version modifiée et allégée de la fonction Duplication de Formes de l'outil *Tweak*. Étant donné notre manque de connaissances préalables du logiciel et de son architecture, cette similitude nous a permis de découvrir rapidement comment ajouter une nouvelle option à Inkscape, et quels étaient les fichiers à modifier pour affiner le développement de cette option.

Ce premier jet de l'outil de spray par copie était néanmoins très sommaire, et très peu optimisé. Le fait d'ajouter des centaines de formes générant des calculs lourds pour l'ordinateur, le manque de mémoire et de ressources se faisait rapidement sentir (au delà de 200–300 formes). Il nous a été par conséquent nécessaire de reprogrammer cette phase de calculs de sorte que la mémoire ne soit pas encombrée inutilement par autre chose que les nouvelles formes copiées. Ceci a été la principale difficulté du développement de ce mode d'utilisation puisque, dans les modes de spray par clone et, *a fortiori*, par fusion des formes, le nombre d'instances de la forme projetée n'a que peu d'importance puisque c'est la forme parente qui possède la majorité des propriétés en mémoire, celles-ci n'étant pas recopiées pour les formes filles.

Dès lors que la fonction de spray elle-même a été opérationnelle, il nous est apparu essentiel de créer une fonction permettant de contrôler la répartition des formes projetées de sorte que celle-ci puisse être appelée indépendemment du mode d'utilisation.

Le développement de cette fonction, dite « de répartition » a constitué un point important du développement de l'outil.

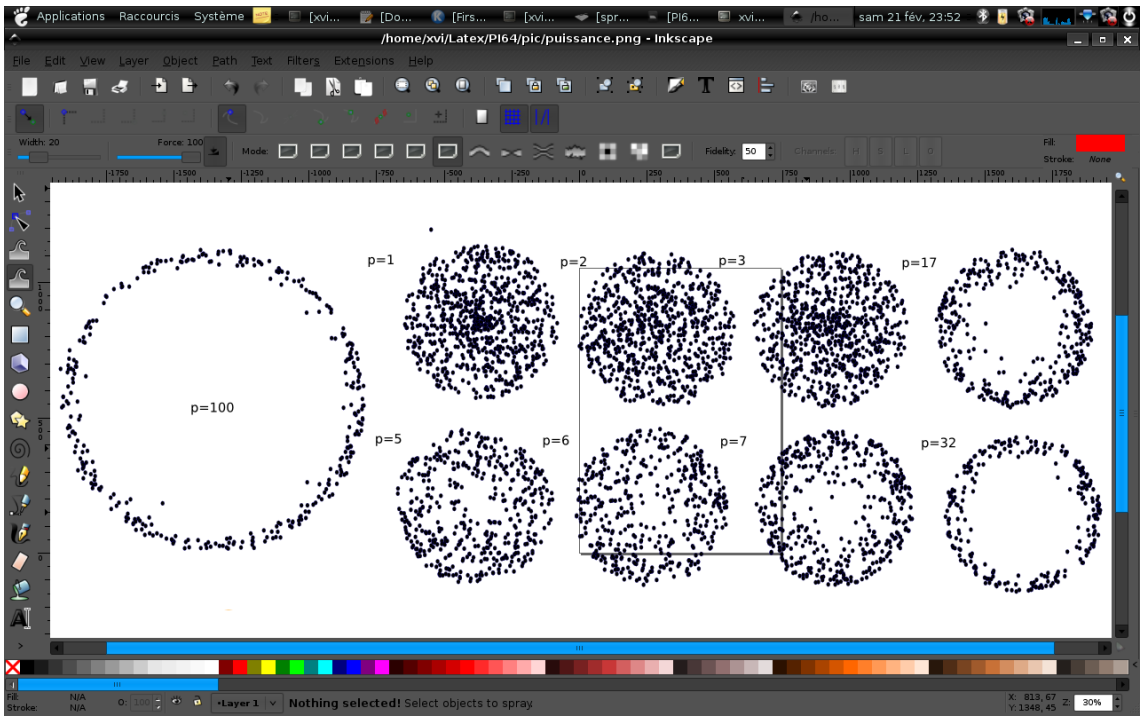

FIG. 4.7 – Premiers essais sur la répartition : polynomiale

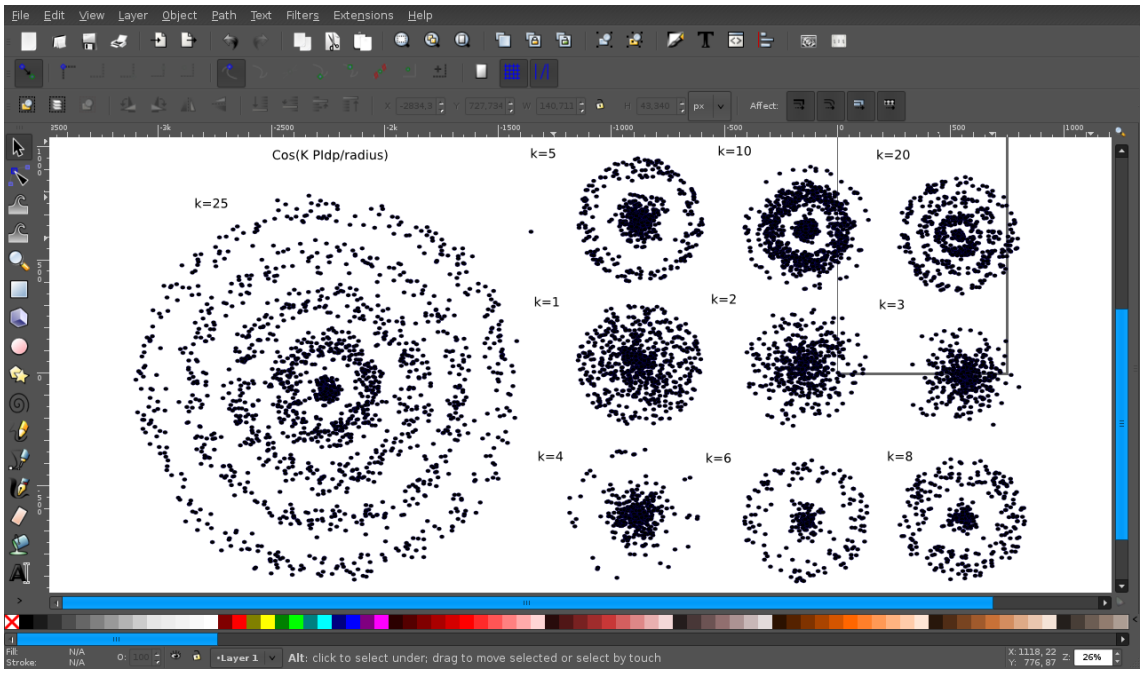

FIG. 4.8 – Premiers essais sur la répartition : concentrique

Spray par clone

Il nous a tout d'abord fallu trouver quelle était la syntaxe permettant de générer des clones à partir d'un objet « père » . Cette fonction n'étant pas du tout exploitée dans l'outil Tweak, nous avons tout d'abord analysé l'outil *Clone Tyler*, permettant de faire un « pavage de clones » à partir d'un objet initial. Néanmoins, cette outil faisant beaucoup plus qu'un simple clonage, le code s'est averé très compliqué et inexploitable dans le cadre de l'utilisation que nous désirions en faire. Nous avons donc cherché à identifier la fonction appelée par la simple commande *Edition > Cloner > Créer un clone*. Dès lors que nous avons identifié le booléen responsable de cette

commande, il nous a alors été possible de le copier dans notre outil.

Un léger temps d'adaptation a alors été nécessaire puisqu'il nous est apparu que de nombreuses propriétés d'un objet « clone » différaient d'un objet classique, ce qui jouait notamment sur le positionnement des clones, leur dimensionnement et leur orientation.

Une fois cette prise en main effectuée, nous avons pu parfaire notre outil de sorte que la gestion des clones se fasse sans erreur, et que le positionnement (à l'aide de la fonction de répartition), l'orientation et la modification aléatoire d'echelle puissent être conforme aux attentes de l'utilisateur.

Algorithme de projection par copie ou par clone Ci-dessus un schéma résumant l'algorithme de transformation puis de projection des formes :

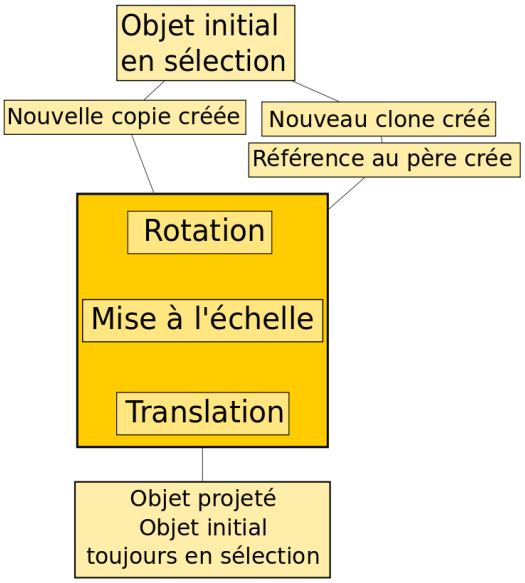

FIG. 4.9 – Principe de création par clones ou copies

Le principe est relativement simple : on crée une nouvelle forme à partir de la sélection initiale. On effectue ensuite une série de transformations sur cet objet : rotation, mise à l'échelle, translation. L'ordre de cette série d'action n'est pas dûe au hasard et permet d'éviter des problèmes de positionnement, notamment rencontrés avec les clones. La translation est régie par la fonction de répartition expliquée plus haut. Agir sur les paramètres du spray revient à agir sur ces transformations. Une fois l'objet projeté, on ne change pas la sélection : c'est toujours l'objet initial.

Spray par union de formes

Ce mode d'utilisation a sans aucun doute été le plus compliqué à mettre en place, puisqu'il repose sur une architecture qui n'était jusqu'alors pas présente dans Inkscape. En effet, il se présente sous la forme d'une boucle qui, à chaque nouvelle itération, doit créer une nouvelle forme issue de la forme « père » et fusionner celle-ci avec une forme dite « Union » qui est par conséquent mise à jour à chaque nouvelle itération.

Le point épineux de cette architecture est qu'on ne sait pas garder un objet en mémoire en passant d'une itération à l'autre, on ne sait mémoriser que l'objet selectionné dans le programme. Il a donc fallu ajouter, au fur et à mesure, les nouvelles formes à la selection active pour pouvoir y avoir accès dans les itérations suivantes. Ceci, malheureusement, a tendance à alourdir l'outil, ce qui reste légèrement handicapant pour une utilisation avec un grand nombre de formes.

Algorithme de projection par union de formes Comme précédemment, voici un schéma résumant l'algorithme de transformation puis de projection des formes :

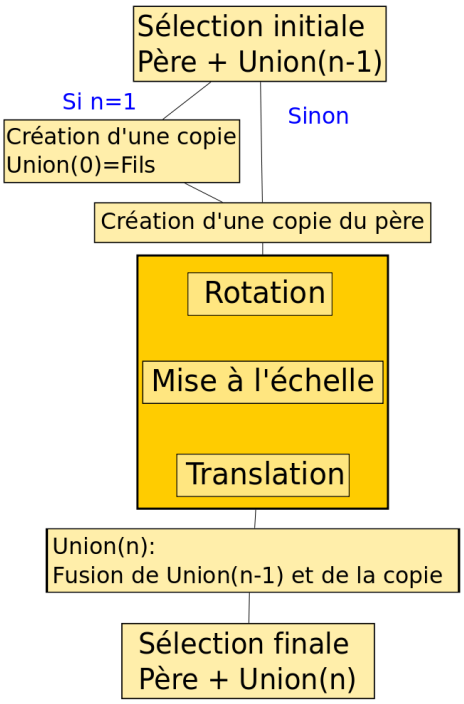

FIG. 4.10 – Principe de création par union

Fonction de répartition

Cette fonction était très intéressante à développer puisqu'elle nous permettait de contrôler la répartition, supposée aléatoire, des formes projetées. Ainsi nous avions le choix des modes de répartition prédéfinis (Gaussienne centrée, uniforme...) que nous n'avions pas à redéfinir dans chacun des outils de spray. Nous avons conçu cette fonction autour de deux modes possibles : une distribution complètement uniforme et une distribution Gaussienne paramétrable.

Cette distribution fournit les coordonnées de l'objet projeté sous forme d'un rayon dr et d'un angle dp, en prenant pour origine le centre du curseur de l'outil.

Le code de cette fonction est fourni en annexe.

Zone de projection

La fonction de répartition permet de projeter les formes dans la zone de projection, mais n'a aucune influence sur les dimensions de cette zone. Ceci est géré directement lors de la translation de l'objet créé. Cette translation définit la zone de projection des formes. Nous utilisions au départ la paramétrisation d'un cercle. La translation de l'objet était donc définie par :

$$T = \begin{pmatrix} dr\cos(dp) \\ dr\sin(dp) \end{pmatrix} + \mathbf{v}$$

dr et dp sont deux paramètres aléatoires dont les valeurs sont gérées par la fonction de répartition. Le vecteur **v** représente la distance entre le centre du curseur et le centre de la sélection initiale. Une fois la répartition selon un cercle maîtrisée, nous avons quelque peu modifié cette paramétrisation en ajoutant une excentricité e. Ceci afin de pouvoir générer des formes selon une ellipse. Nous avons également ajouté un angle D qui permet d'incliner cette ellipse. Ces deux variables sont paramétrables dans l'interface par l'utilisateur. La translation de l'objet est alors définie par :

$$T = \begin{pmatrix} \cos(D)\cos(dp)\dfrac{dr}{1-e} + \sin(D)\sin(dp)\dfrac{dr}{1+e} \\ -\sin(D)\cos(dp)\dfrac{dr}{1-e} + \cos(D)\sin(dp)\dfrac{dr}{1+e} \end{pmatrix} + \mathbf{v}$$

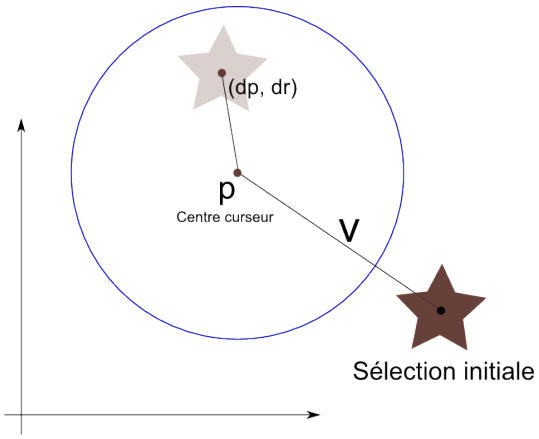

FIG. 4.11 – Schéma de la zone de projection

Il restait enfin à modifier la forme du curseur, qui est définie indépendamment de nos paramétrisations. Quelques fonctions géométriques propres à Inkscape ont suffi pour cela.

Utilisation de la molette de la souris

Nous avons rapidement été confrontés à un dilemme, puisque, dès les prémices du projet, nous souhaitions disposer d'un outil nous permettant de projetter des formes sans avoir besoin de déplacer la souris, juste en maintenant le clic. Or, ceci requiert un écouteur sur souris qui est absent du programme : en effet, il est actuellement possible de relier un outil au clic simple de la souris, de même qu'au « Drag » de la souris (clic + déplacement), mais il est pour l'instant impossible d'utiliser un clic maintenu de la souris. En l'état, cela impliquait que, pour utiliser notre outil autour d'une position fixe, il était nécessaire de conserver un léger mouvement de la souris autour de la position visée.

Pour remédier à cela, il nous est apparu la possibilité d'exploiter la molette de la souris, qui nous permet, tout en restant sur une position fixe, de projetter une forme à chaque fois que la molette est activée. Néanmoins, plusieurs difficultés ont été rencontrées lors de sa mise en place. En effet, les actions de la molette sont déjà définis par Inkscape et il a fallu accompagner la molette d'un appui sur le clic gauche pour pouvoir effectuer une projection. La programmation de la projection de formes s'effectue ensuite de la même manière que pour le clic gauche.

Conclusion

Aujourd'hui, nous pouvons dire que nous avons relevé le défi proposé. Nous n'avons pas pu implémenter toutes les fonctions demandées par le blueprint, mais nous sommes parvenus à coder un outil fonctionnel et stable. Une documentation sera toutefois nécessaire pour pouvoir utiliser pleinement l'outil. Pour ce faire, la communauté met à disposition la documentation pour que chaque développeur puisse expliquer le fonctionnement de son outil.

Nous aurions aimé implémenter cette fonction dans la prochaine version officielle qui sort dans quelques semaines, mais le temps nous a manqué. Nous avons quand même livré notre code aux développeurs et allons maintenant suivre son évolution. Et pourquoi pas y apporter encore des modifications, et l'améliorer encore. Car c'est aussi cela, le monde de l'open-source.

Table des figures

Annexe A

Impressions personnelles

A.1 Julien Leray

Au début de ce projet, je ne connaissais presque rien au langage C++ excepté les bases vues en cours à Centrale. Alors j'avais un peu d'appréhension à l'idée de travailler sur un si gros programme. Les premières semaines furent laborieuses et difficiles. Non seulement pour le déchiffrage des multiples bibliothèques que je ne connaissais pas, mais aussi pour le fonctionnement du SVN, d'Ubuntu...

Puis je me suis habitué, et j'ai surtout réussi à coder des morceaux utiles dans notre outil comme la fenêtre d'options. Maintenant, naviguer dans tous ces .h et .cpp est une formalité et un plaisir inattendu. L'utilisation du système Linux est désormais quotidienne et la rédaction de rapports en LaTeX, automatique.

A.2 Benoît Lavorata

La raison principale pour laquelle j'ai voulu faire partie de ce projet était de pouvoir appréhender un univers particulier —*l'open source*— mais surtout de mettre en place un sytème utilisé et conçu par le monde entier. Mes connaissances en informatique se limitaient à la programmation en console ou aux interfaces peu évoluées, alors que dans ce projet on se proposait de réaliser un outil de création graphique performant. Ces raisons m'ont poussé à choisir ce projet, et je suis satisfait de ce choix.

Nous nous connaissions déjà avant de former l'équipe du PI64 et aucun d'entre nous ne possédait de véritable talent de développeur. Cependant nous avons réussi à créer une ambiance agréable et productive au sein du groupe, ce qui nous a permis de finir le projet dans les temps. Cela a aussi mis en évidence l'importance d'une organisation saine et claire pour tout le monde, puisque les échéances étaient brèves.

En fin de compte, ce que je retire de ce projet, ce sont des compétences en informatique — *utilisation approfondie de linux, LaTeX, serveurs SVN et C++* — et la satisfaction de voir prochainement notre outil implémenté dans la version officielle pour le plaisir des utilisateurs ou pour servir de base à d'autres développeurs.

A.3 Pierre Caclin

N'ayant pas de base en informatique plus poussée que les cours enseignés à Centrale, j'appréhendais un peu la tâche qui nous était confiée et qui s'annonçait ardue. Le fait de connaître les autres membres du groupe et de savoir qu'ils n'en savaient pas forcément plus que moi

a globalement aidé à dissiper cette appréhension et a permis, comme l'a dit Benoît, la création d'une ambiance studieuse mais détendue. Ce fut notamment le cas durant les vacances de Pâques, où nous avons pu nous retrouver pour faire avancer le projet de manière très significative !

N'étant pas forcément féru d'informatique, c'est davantage le côté infographie du projet qui m'attirait. Au final, j'ai pris énormément de plaisir à travailler dans ce groupe, et je ne pense pas être le seul... Le fait de pouvoir choisir ses collaborateurs est un plus indéniable, et ajoute encore à la satisfaction de voir notre outil fonctionnel et bientôt, nous l'espérons, intégré à la version officielle.

A.4 Pierre-Antoine Marc

Familier avec le monde de la création graphique depuis quelque temps, ce projet fut l'occasion pour moi de découvrir en quelque sorte l'envers du décor. Et également la chance de participer à un projet open-source d'envergure, puisque Inkscape est de plus en plus reconnu par les graphistes du monde entier.

Mis à part les quelques projets en informatique réalisés à l'école, je n'avais jamais vraiment développé sur un logiciel aussi complexe. Le défi proposé par Steren — car c'en fut un ! — n'était pas gagné d'avance. En tant que chef de projet, il n'a pourtant pas été difficile de motiver les troupes, tellement le sujet nous passionnait. L'investissement réalisé dans ce projet est difficilement quantifiable : chaque membre a contribué à sa réussite.

Au final, j'ai beaucoup appris au cours des 5 derniers mois et je suis plutôt fier d'avoir travaillé sur un tel projet et espère avoir pu apporter notre pierre à cet incroyable édifice qu'est Inkscape !

A.5 Vincent Montagné

Passionné d'infographie et me dirigeant en 3A vers une filière tournée vers l'informatique, c'est tout naturellement que je me suis dirigé vers ce projet. Il a ainsi été l'occasion pour moi de faire le rapprochement entre les applications graphiques que j'utilise et l'enseignement d'informatique dispensé à l'école.

La phase de compréhension du code source s'est avérée plus longue que prévu, bien que les élèves ayant effectué l'an dernier un PI sur le même logiciel nous en avaient informé. De plus, ce logiciel étant open source, sa documentation était dans l'ensemble riche bien qu'inhomogène.

Durant ce projet, j'ai travaillé principalement au développement de l'interface. La librairie utilisée par Inkscape pour gérer l'interface utilisateur est assez documentée sur Internet, ce qui nous a permis par la suite de nous familiariser plus rapidement avec le code. Ensuite nous avons travaillé en collaboration avec l'équipe s'occupant du noyau fonctionnel pour assurer les liens entre l'interface et les fonctions appelées. Tout cela m'a montré a quel point le travail en équipe était nécessaire pour mener à bien un projet d'une telle envergure.

De plus, le fait de travailler avec Inkscape nous a familiarisé avec les logiciels libres, par exemple nous avons tous utilisé un environnement Linux, auxquel peu d'entre nous étaient habitués. Nous avons ainsi découvert une communauté soudée et active qui contribue à l'avancée de tels logiciels.

A.6 Pierre Barbry-Blot

Ce projet m'a permis, durant les 5 derniers mois, de mettre (enfin) en application les cours d'informatique dispensés à Centrale, mais surtout de compléter ceux-ci à l'aide de ce qui était sûrement l'un des meilleurs travaux pratiques que l'on pourrait concevoir. En effet, pour une fois, nous n'étions plus seulement jugés sur nos connaissances en C++, mais aussi sur notre capacité à mettre ces connaissances en commun au sein d'un groupe.

En effet, le fait de travailler sur un module s'intégrant au sein d'un logiciel existant, ainsi que le fait de travailler au sein d'une communauté pré-existante de développeurs, avec ses codes et ses règles, m'a permis de mieux prendre conscience de la complexité appelée par le travail en équipe sur un même projet.

Enfin, j'ai beaucoup apprécié les fréquentes phases de collaboration nécessaire à ce que chacun des membres de l'équipe, travaillant sur son bout de code ou sa sous-fonction, puisse intégrer son travail au sein du travail de tous les autres. Cette collaboration a, pour moi, été particulièrement stimulante, et j'espère avoir pu communiquer cette motivation aux autres membres, tout comme ils m'ont communiqué la leur.

A.7 Aurel-Aimé Marmion

Utilisateur de Inkscape et aimant la programmation, ce projet m'a tout de suite séduit. D'une part, il m'a permis de voir et de comprendre (en partie) le code source qui se cache derrière un logiciel complexe et d'autre part, j'ai pu pour la première fois participer à un projet informatique.

Au début du projet, j'appréhendais un peu quant à la complexité de la tâche que nous devions accomplir n'ayant guère plus d'expérience que les cours de Centrale en la matière. A l'usage, on se rend compte de l'importance de bien documenter le code que l'on écrit surtout quand il est destiné à être partagé. Pour cela, les documentations d'Inkscape et des différentes librairies open-source utilisées furent une aide précieuse pour comprendre et avancer.

Au court du projet, j'ai compris l'importance de travailler en équipe. La séparation des tâches a permis d'avancer efficacement dans le projet et on a pu s'apercevoir que la résolution des problèmes pouvait provenir de n'importe lequel d'entre nous.

Finalement, j'ai la sensation d'avoir contribué utilement au développement du logiciel libre qu'incarne Inkscape. L'accomplissement serait que notre outil intègre rapidement la version officielle du logiciel.

Annexe B

Outils de développement

B.1 Développer sous Linux

Pour le développement d'Inkscape, nous avons fait le choix de travailler sur Linux. Cela simplifie grandement les choses à la compilation du logiciel. De plus, il suffit de coder avec un simple traitement de texte.

B.2 Travailler avec un serveur SVN

SVN, pour Subversion, est un gestionnaire de révisions du code. Il permet de synchroniser les modifications apportées au code source par chaque développeur sur un même serveur. Cet outil nous fût fort utile, et nous a permis également de rester à jour avec la branche officielle d'Inkscape qui fonctionne avec le même protocole.

Chaque révision est numérotée, et permet également d'ajouter des commentaires sur les modifications effectuées. Le serveur SVN gère automatiquement les changements dans les lignes, réalise les ajouts lui-même et repère les conflits lorsqu'une même ligne a été modifiée. C'est alors à l'utilisateur de résoudre le problème "à la main".

Nous avons aussi utilisé un serveur SVN pour réaliser le rapport, ce qui représente un gain de temps et une minimisation des erreurs de version précieux. Nous remercions d'ailleurs les membres de l'ancien groupe de PI-Inkscape qui nous ont donné l'opportunité d'utiliser ce genre de systèmes et qui l'ont installé pour notre projet sur les serveurs de l'école.

Annexe C

Code Source

C.1 Fonction de Répartition

C.1.1 Définition de la fonction NormalDistribution

Cette fonction permet, en indiquant les parametres m et s (respectivement la moyenne et l'écart-type), de génerer un nombre aléatoire suivant une distribution Gaussienne de moyenne m et d'écart-type s. Cette fonction est utilisée dans la fonction *random position*, détaillée ensuite.

```
// The following code implements NormalDistribution wich is used for the density of the spray
/*
   RAND is a macro which returns a pseudo-random numbers from a uniform
   distribution on the interval [0 1]
*/
#define RAND ((double) rand())/((double) RAND_MAX)

/*
   TWOPI = 2.0*pi
*/
#define TWOPI 2.0*3.14159265358979323846264338327950288419716939937 5
/*
   RANDN is a macro which returns a pseudo-random numbers from a normal
   distribution with mean zero and standard deviation one. This macro uses Box
   Muller's algorithm
*/
#define RANDN sqrt(-2.0*log(RAND))*cos(TWOPI*RAND)
double NormalDistribution(double mu,double sigma) {
    /*
       This function returns a pseudo-random numbers from a normal distribution with
       mean equal at mu and standard deviation equal at sigma > 0
    */
    return (mu+sigma*RANDN);
}
//Ending the crÃ©ation of NormalDistribution
```

C.1.2 Définition de la fonction random position

Cette fonction génère un angle et un rayon permettant de positionner l'élément projeté par rapport au centre de l'outil.

```
void random_position( double &r, double &p, double &a, double &s, int choix)
{
    if (choix == 0) { // Mode 1 : uniform repartition
        r = (1-pow(g_random_double_range(0, 1),2));
```

```
        p = g_random_double_range(0, M_PI*2);
    }
    if (choix == 1) { //Mode 0 : gaussian repartition
        double r_temp =-1;
        while (!((r_temp>=0)&&(r_temp<=1))) {
            r_temp = NormalDistribution(a,s/4);
        }
// generates a number following a normal distribution
        p = g_random_double_range(0, M_PI*2);
        r=r_temp;
    }
}
```

C.2 Code du Spray en mode Copie

```
if (mode == SPRAY_MODE_COPY)
{
    Geom::OptRect a = item->getBounds(sp_item_i2doc_affine(item));
    if (a) {
        double dr;
        double dp;
        random_position(dr,dp,mean,standard_deviation,_distrib);
        dr=dr*radius;
        double _fid = g_random_double_range(0,1);
        SPItem *item_copied;
        double angle = g_random_double_range(rot_min, rot_max);
        double _scale = g_random_double_range(scale_min, scale_max);
        if (_fid<=population) {
            // duplicate
            SPDocument *doc = SP_OBJECT_DOCUMENT(item);
            Inkscape::XML::Document* xml_doc = sp_document_repr_doc(doc);
            Inkscape::XML::Node *old_repr = SP_OBJECT_REPR(item);
            Inkscape::XML::Node *parent = old_repr->parent();
            Inkscape::XML::Node *copy = old_repr->duplicate(xml_doc);
            parent->appendChild(copy);

            SPObject *new_obj = doc->getObjectByRepr(copy);
            item_copied = (SPItem *) new_obj; //convertion object->item
            Geom::Point center=item->getCenter();
            sp_spray_scale_rel(center,desktop,item_copied, Geom::Scale(_scale,_scale));
            sp_spray_scale_rel(center,desktop,item_copied, Geom::Scale(scale,scale));

            sp_spray_rotate_rel(center,desktop,item_copied, Geom::Rotate(angle));
            Geom::Point move = (Geom::Point(cos(tilt)*cos(dp)*dr/(1-ratio)+sin(tilt)*sin(dp)*dr
                /(1+ratio),-sin(tilt)*cos(dp)*dr/(1-ratio)+cos(tilt)*sin(dp)*dr/(1+ratio)))+(p-a
                ->midpoint());//Move the cursor p
            sp_item_move_rel(item_copied, Geom::Translate(move[Geom::X], -move[Geom::Y]));
            did = true;
        }
    }
}
```

C.3 Code du Spray en mode Clone

```
if (mode == SPRAY_MODE_CLONE) {
    Geom::OptRect a = item->getBounds(sp_item_i2doc_affine(item));
```

```
    if (a) {
        double dr;
        double dp;
        random_position(dr,dp,mean,standard_deviation,_distrib);
        dr=dr*radius;
        double _fid = g_random_double_range(0,1);
        double angle = (g_random_double_range(rot_min, rot_max));
        double _scale = g_random_double_range(scale_min, scale_max);

        if (_fid<=population) {
            SPItem *item_copied;
            SPDocument *doc = SP_OBJECT_DOCUMENT(item);
            Inkscape::XML::Document* xml_doc = sp_document_repr_doc(doc);
            Inkscape::XML::Node *old_repr = SP_OBJECT_REPR(item);
            Inkscape::XML::Node *parent = old_repr->parent();

            //Creation of the clone
            Inkscape::XML::Node *clone = xml_doc->createElement("svg:use");
            parent->appendChild(clone); //Ajout du clone Ã  la liste d'enfants du pÃ¨re (
                selection initiale
            clone->setAttribute("xlink:href", g_strdup_printf("#%s", old_repr->attribute("id")),
                 false); //GÃ©nÃ¨re le lien entre les attributs du pÃ¨re et du fils
            SPObject *clone_object = doc->getObjectByRepr(clone);
            item_copied = (SPItem *) clone_object;//conversion object->item
            Geom::Point center=item->getCenter();
            sp_spray_scale_rel(center,desktop,item_copied, Geom::Scale(_scale,_scale));
            sp_spray_scale_rel(center,desktop,item_copied, Geom::Scale(scale,scale));
            sp_spray_rotate_rel(center,desktop,item_copied, Geom::Rotate(angle));
            Geom::Point move = (Geom::Point(cos(tilt)*cos(dp)*dr/(1-ratio)+sin(tilt)*sin(dp)*dr
                /(1+ratio),-sin(tilt)*cos(dp)*dr/(1-ratio)+cos(tilt)*sin(dp)*dr/(1+ratio)))+(p-a
                ->midpoint());
            sp_item_move_rel(item_copied, Geom::Translate(move[Geom::X], -move[Geom::Y]));
            Inkscape::GC::release(clone);
            did = true;
        }
    }
}
return did;
}
```

C.4 Code du Spray en mode Union de Formes

```
else if (mode == SPRAY_MODE_SINGLE_PATH) {
    SPItem *Pere; //Objet initial
    SPItem *item_copied;//Objet projetÃ©
    SPItem *Union;//Union prÃ©cÃ©dente
    SPItem *fils;//Copie du pÃ¨re
    int i=1;
    for (GSList *items = g_slist_copy((GSList *) selection->itemList());
            items != NULL;
            items = items->next) {

        SPItem *item1 = (SPItem *) items->data;
        if (i==1) {
            Pere=item1;
        }
        if (i==2) {
            Union=item1;
```

```
        }
        i++;
    }
    SPDocument *doc = SP_OBJECT_DOCUMENT(Pere);
    Inkscape::XML::Document* xml_doc = sp_document_repr_doc(doc);
    Inkscape::XML::Node *old_repr = SP_OBJECT_REPR(Pere);
    Inkscape::XML::Node *parent = old_repr->parent();
    Geom::OptRect a = Pere->getBounds(sp_item_i2doc_affine(Pere));
    if (a) {
        double dr;
        double dp; //initialisation des variables
        random_position(dr,dp,mean,standard_deviation,_distrib);
        dr=dr*radius;
        double _fid = g_random_double_range(0,1);
        double angle = (g_random_double_range(rot_min, rot_max));
        double _scale = g_random_double_range(scale_min, scale_max);
        if (i==2) {
            Inkscape::XML::Node *copy1 = old_repr->duplicate(xml_doc);
            parent->appendChild(copy1);
            SPObject *new_obj1 = doc->getObjectByRepr(copy1);
            fils = (SPItem *) new_obj1; //conversion object->item
            Union=fils;
            Inkscape::GC::release(copy1);
        }
        if (_fid<=population) { //Rules the population of objects sprayed
            // duplicates the father
            Inkscape::XML::Node *copy2 = old_repr->duplicate(xml_doc);
            parent->appendChild(copy2);
            SPObject *new_obj2 = doc->getObjectByRepr(copy2);
            item_copied = (SPItem *) new_obj2;
            Geom::Point move = (Geom::Point(cos(tilt)*cos(dp)*dr/(1-ratio)+sin(tilt)*sin(dp)*dr
                /(1+ratio),-sin(tilt)*cos(dp)*dr/(1-ratio)+cos(tilt)*sin(dp)*dr/(1+ratio)))+(p-a
                ->midpoint());//Move around the cursor
            Geom::Point center=Pere->getCenter();
            sp_spray_scale_rel(center,desktop,item_copied, Geom::Scale(_scale,_scale));
            sp_spray_scale_rel(center,desktop,item_copied, Geom::Scale(scale,scale));
            sp_spray_rotate_rel(center,desktop,item_copied, Geom::Rotate(angle));
            sp_item_move_rel(item_copied, Geom::Translate(move[Geom::X], -move[Geom::Y]));
//UNION
            selection->clear();
            selection->add(item_copied);
            selection->add(Union);
            sp_selected_path_union(selection->desktop());
            selection->add(Pere);
            Inkscape::GC::release(copy2);
            did = true;
        }
    }
}
```