Le logiciel libre dans un studio d'animation

Damien Picard

Université Paris 8 – Vincennes Saint-Denis
UFR Arts, philosophie, esthétique
École Doctorale Esthétique, Sciences et Technologies des Arts

Thèse

Pour obtenir le grade de
Docteur de l'Université Paris 8
Discipline : Esthétique, sciences et technologies des arts
Spécialité : Images numériques

Le logiciel libre dans un studio d'animation

Présentation et soutenance le 2 décembre 2020 par Damien Picard

Thèse dirigée par Alain Lioret
En codirection avec Anne-Laure George-Molland

Membres du jury

Alain Lioret (PR)	Université Paris 8
Sébastien Broca (MCF)	Université Paris 8
Réjane Hamus-Vallée (PR)	Université d'Évry
Manthos Santorineos (PR)	École des beaux-arts d'Athènes
Anne-Laure George-Molland (MCF)	Université Paul Valéry Montpellier 3

Résumé

L'animation, la pratique consistant à donner vie à des images en mouvement créées de toutes pièces, utilise aujourd'hui largement l'informatique pour la création des images. Cela signifie que les graphistes, les techniciens et développeurs utilisent des programmes spécialisés pour mener à bien toutes les étapes de création.

Parmi ces logiciels, il en est une catégorie particulière : le logiciel libre, auquel les studios d'animation s'intéressent de plus en plus depuis le début de la décennie. Ce type de programmes présente des particularités juridiques qui permettent à leurs utilisateurs de les utiliser comme bon leur semble, à l'inverse des logiciels dits propriétaires, dont l'utilisation est sévèrement restreinte par les conditions imposées par leurs licences. Les licences de logiciels libres permettent en particulier d'utiliser, d'étudier, de modifier et de distribuer le code source des programmes, ce qui permet un développement collaboratif par les communautés de développeurs et d'utilisateurs.

Certaines applications de création, comme Blender et Krita, sont aujourd'hui considérées comme des alternatives crédibles à leurs équivalents propriétaires, et peuvent trouver leur place au sein des pipelines de fabrication des studios. Ces derniers peuvent ainsi contribuer par leur savoir-faire à l'amélioration des programmes, pour le bien de l'ensemble de la communauté d'utilisateurs.

Ils peuvent également distribuer sous licence libre leurs propres scripts et programmes, qu'ils élaborent souvent dans le cadre de leurs productions. Différents studios peuvent ainsi collaborer sur des outils, profiter de moyens de production communs et ainsi partager leurs coûts de développement, plutôt que d'espérer garder des secrets de fabrication propriétaires et recommencer toujours les mêmes développements.

Mots-clefs : animation, 3D, création, logiciel libre, open source, pipeline

Abstract

Animation, the practice of giving life to moving pictures created ex nihilo, broadly uses computers to create images. Artists, technicians and developers thus use specialised programs to achieve all steps of creation.

There is one particular category of software in which animation studios have a growing interest since the beginning of the decade: free software. This type of software has unique legal particularities, which enable its users to use them as they see fit, contrary to so-called proprietary software, whose use is severely restrained by the conditions imposed by their licenses. Free software licenses allow especially the use, study, modification and distribution of the programs, which enables a collaborative development by user and developper communities.

Some content creation applications, such as Blender and Krita, are now considered credible alternatives to their proprietary equivalents, and may find a place in the animation studios' pipelines. The studios may in turn contribute to improving those programs using their professional skills, which profits the whole user community.

The studios may as well distribute their own scripts and programs under free software licenses, when they create such in-house tools for their own projets. Multiple studios may then collaborate to create and improve tools, thus sharing their development costs, rather than hoping to keep proprietary trade secrets, and developing the same tools over and over again.

Keywords: animation, 3D, creation, free software, open source, pipeline

Remerciements

Une thèse, il paraît que c'est long et difficile, je le savais avant même le début. Alors que j'écris ces lignes, je suis encore loin de l'avoir terminée, mais si vous les lisez, c'est que c'est fait et que tout va bien. Pour mener à bien cette entreprise, je n'ai pas été tout seul et je veux remercier des tas de gens. En voici quelques-uns.

Maman et Papa. Mon frère. Sa femme, ma belle sœur.

Virginie, qui m'a proposé de faire cette thèse, allant même jusqu'à insister bien longtemps. J'ai dit oui.

Anne-Laure "ALGM" George-Molland, ma codirectrice de thèse, qui m'a accompagné, suivi, et supporté sans relâche, et sans qui je n'aurais pas fini, ou en tout cas avec autant de rigueur.

Mon collègue-associé-superviseur-ami Flavio Perez (¡Hola señor !), qui a été d'une aide précieuse pour cette thèse : il m'a appris une partie de mon métier (celle où je fais comme si j'étais un TD), montré plein de choses, relu, veillé à ma santé et généralement été un bon copain.

Merci aux gens que j'ai interrogés et que je cite dans ma thèse : Damien Coureau, Flavio Perez, François Grassard, Léa Cluzel, Loqmane Bahri.

Mes colocataires, avec qui j'ai vécu plusieurs années de ma vie, mais pas tous en même temps : Max, Flavio, Duy Kévin, Alina, Mina, Lindsay, Floriane, Julien. Pour la plupart, ils furent également mes collègues — comparer avec le paragraphe suivant — et nous avons vécu en bonne intelligence malgré ça.

Mes collègues des Fées spéciales, avec qui nous avons collaboré, fait de belles choses, pas mal rigolé et partagé ensemble des moments forts, bons ou mauvais. Merci à Alina,

Amin, Aude, Carolina, Clémence, Clémentine, Cyrille, Duy Kévin, Emmeline, Eric, Ève, Gabrièla, Guilhem, Jeanne-Sylvette, Joris, Loqmane, Léa, Léo, Marie, Marion, Marthe, Mathieu, Max, Meryl, Mina, Natalène, Oriane, Philippe, Sophie, Vincent, Yann.

Mes relectrices et relecteurs, encore les mêmes : Anne-Laure, Duy, Flavio, Floriane, Natalène.

Je remercie les équipes des projets sur lesquels j'ai travaillés, en particulier :

— *Dilili* : Michel, Jean-Claude, Malek ;

— *Musée de Lodève* : Stéphane, Noisette.

Je remercie les auteurs et contributeurs des logiciels libres et projet suivants, sans qui cette thèse n'aurait pas eu grand-chose à étudier :

— Blender ;

— Godot ;

— Krita ;

— CGWire ;

— Kabaret ;

— OpenStreetMap.

Enfin, je remercie les copains, que je ne vois pas tous souvent, mais qui n'en sont pas moins là. En plus des copains déjà mentionnés parce qu'ils sont aussi des collègues, je remercie donc Adam, Alix, Clarisse, Eddy, Émilie, Erwan, Florine, François, Freddy, Gabriel, Gabriel, Gaspard, Guillaume, Juliette, Kévan, Laure, Louis-Julien, Lucile, Matthieu, Nicolas, Noélie, Ouirich, Pascal, Romain, Simon, ainsi que les membres de ma famille, qui se reconnaîtront sans ambiguïté possible.

Table des matières

Introduction générale

Le secteur du cinéma d'animation connaît des changements constants et rapides. Outre les changements esthétiques, visibles par le public, les méthodes de fabrication évoluent également. Depuis plusieurs décennies maintenant, la fabrication des films d'animation est en grande partie, ou intégralement numérique. Cela implique que soient utilisés des logiciels, dont l'usage évolue également au gré de « modes ». Parmi les logiciels, il existe une catégorie particulière : le logiciel libre. Ce mouvement existe depuis les années 1980, et il commence trente ans plus tard à produire des logiciels adoptés par les studios. La fabrication de chaque film, de chaque œuvre, a ses particularités, mais l'utilisation du logiciel libre crée des problématiques singulières car elle demande de considérer le logiciel autrement, que l'on soit développeur ou contributeur, utilisateur ou formateur.

Cette thèse de doctorat étudiera les possibilités et les limitations actuelles de l'usage du logiciel libre dans un petit studio d'animation.

Les logiciels changent constamment, et les processus aussi. Plutôt que de décrire une solution d'un bout à l'autre, cette thèse expose donc des briques logicielles existantes ou conçues au sein d'un studio, et la logique sous-jacente du développement et de la publication de ces outils.

Cette thèse a été menée dans le cadre d'une convention Cifre avec une entreprise du secteur du cinéma d'animation, les Fées spéciales (cf. section I.1.1). De ce fait, une grande partie du temps de recherche s'est effectuée au sein de l'équipe Recherche et Développement (R&D*[1]) de l'entreprise. Ainsi, mon rôle a été de concevoir, mettre en œuvre, tester, valider des programmes et processus pour la création audiovisuelle. La

1. Les termes suivis d'une astérisque sont définis dans le glossaire.

jeunesse et la taille réduite de la structure étaient propices à la recherche et à l'innovation. Une grande latitude m'a donc été laissée dans la méthode et le choix des solutions techniques. En effet, l'équipe R&D, qui ne portait même pas ce nom à mon arrivée, était composée de deux à trois programmeurs, et la responsabilité de développement s'est partagée assez naturellement, selon les compétences de chacun. De plus, la plupart du travail réalisé en entreprise a consisté à résoudre des problèmes ciblés, et n'a pas nécessité de méthode de développement complexe, mais plutôt d'une élaboration organique des scripts*, programmes et processus.

Ce poste était très technique, mais n'excluait pas des composantes artistiques, à deux titres. D'abord, j'ai participé directement à des productions en tant que graphiste, en particulier pour des programmes nécessitant des compétences techniques relativement avancées. Ensuite, j'ai collaboré avec des graphistes pour mettre en œuvre des solutions techniques pour le pipeline* de fabrication. La conception des outils a été le fruit d'un processus itératif, d'un aller-retour avec les graphistes, de l'étape du cahier des charges et des prototypes au support en cours de production, afin d'obtenir des outils adaptés aux productions de la structure, mais également de recueillir les opinions, habitudes de travail et expériences des utilisateurs principaux.

À ce poste, j'ai également pris part à des décisions concernant la stratégie de développement open source, dans l'évaluation et le choix de solutions existantes, comme dans la communication et la publication à l'extérieur de nos propres solutions, et c'est en partie dans ce cadre que je me suis documenté sur les aspects historiques, juridiques, organisationnels et communicationnels du développement de logiciel libre.

En parallèle de mon travail de R&D en entreprise, j'ai profité de périodes de recherche à l'extérieur pour faire des expériences artistiques plus personnelles et moins directement soumises aux contraintes de la production. Ces expériences ne sont pas toujours parties d'hypothèses précises, mais plutôt d'intuitions. Cet état d'esprit a permis d'explorer plus librement mon champ de recherche, et ont fourni un matériau d'analyse riche.

Ma démarche personnelle s'inscrit donc dans un processus de recherche-création en ce que les œuvres qui en résultent sont utiles à l'élaboration de ma recherche, et ne sont

pas destinées à exister uniquement en dehors de cette recherche. Elles entrent dans la catégorie que Chapman et Sawchuk [10] appellent *la création comme recherche*, c'est-à-dire « l'élaboration de projets dans lesquels la création est requise pour que puisse émerger la recherche ». Il est à noter que le terme de *recherche-création* est généralement considéré dans un contexte de sciences humaines. Dans mon cas, à la suite des études que j'ai suivies dans le département Arts et Technologies de l'Image à l'Université Paris VIII, une composante essentielle est pratique et technique, et s'attache à déterminer l'influence de la technique sur les possibilités créatrices.

Par ailleurs, la méthode que j'adopte exploite une sorte de sérendipité dans l'expérimentation, en tâchant d'éviter de présupposer des résultats que pourraient fournir une expérience, mais en orientant cette expérience pour que ses résultats, quels qu'ils soient, portent des points de données intéressants. C'est une méthode que j'avais déjà employée lors de ma recherche en master en 2014 sur le sujet de *La boucle d'animation*, que j'avais appelée « pour voir comment ça fait », ce qui ne semble pas au premier abord une méthode bien rigoureuse, mais est à la base de ma méthode de recherche artistique expérimentale. Aujourd'hui, je la reformulerais en « on verra bien ce que ça donne », en remarquant le fait que les deux formulations touchent d'abord au visuel et à la potentialité, car on ne prévoit pas à l'avance les sentiments provoqués par une création ; et aux résultats, qu'il faut comprendre comme des résultats esthétiques, mais aussi comme des résultats expérimentaux, propres à dégager des conclusions théoriques.

Cette thèse comporte trois parties. Je commencerai par exposer dans la partie I le contexte de recherche. Il est nécessaire d'avoir plusieurs notions en tête pour comprendre les spécificités du logiciel libre et open source. C'est pourquoi j'exposerai des points historiques, idéologiques, et juridiques.
La partie II analysera en détail les modalités de cette mise en œuvre par les studios à travers trois facettes concrètes et complémentaires : économique, juridique et pratique. Enfin, la partie III sera consacrée aux solutions existantes et à développer pour mettre en œuvre une chaîne de fabrication libre aussi complète que possible.

Première partie

L'animation et le libre : contexte de recherche

Introduction

Afin de bien appréhender l'élaboration d'une chaîne de fabrication libre dans un studio d'animation, il est nécessaire de bien connaître d'une part les spécificités de ce type de structure, et d'autre part de comprendre ce que constitue le logicel libre et open source.

Cette partie décrira donc les studios d'animation en général, et le studio Les Fées spéciales en particulier, dans la mesure où ce studio a une stratégie active d'utilisation et de développement de logiciel libre, et où j'y ai travaillé trois ans. Je décrirai également une chaîne de fabrication typique de film animé en 3D, de la préproduction à la post-production, et le rôle de chaque département dans la chaîne globale.

J'aborderai ensuite le logiciel libre, en rappelant rapidement son histoire et ses spécificités juridiques et idéologiques, qui sont la fondation du mouvement et des méthodes de développement open source. Je mentionnerai également les bases de données ouvertes, qui fonctionnent selon un principe similaire à celui des logiciels libres, et peuvent être de précieuses ressources pour certains projets d'animation.

Chapitre I.1

Le secteur de l'animation

Introduction

L'animation est un ensemble de techniques, de médiums* et de supports de création ayant en commun la création de séquences *animées*, c'est-à-dire, pour simplifier, en mouvement. Les images qui constituent ces séquences peuvent être des dessins, des images de synthèse, des photographies d'objets physiques (sculptures, êtres vivants, etc.), ou souvent une combinaison de plusieurs de ces sources.

Le paysage de l'animation change continuellement depuis le début du XX^{e} siècle, et se formalise de plus en plus en des pratiques industrielles uniformes. Si de l'animation peut être produite dans des contextes très différents, la plupart des productions se font dans un environnement semblable et utilisent des techniques similaires.

Je présenterai dans ce chapitre ces deux aspects de l'animation. L'environnement, c'est le studio d'animation, une entreprise dédiée à l'animation. Quant aux techniques, elles sont formalisées au sein d'un studio sous la forme d'un « pipeline » que je décrirai.

I.1.1 Les studios d'animation

Certains animateurs travaillent seuls, avec pour seul support du papier et pour seuls outils des crayons, mais la grande majorité de la production de films d'animation se fait

au sein d'entreprises communément appelées des studios d'animation, par des équipes plus ou moins nombreuses constituées de graphistes, de techniciens, et de personnel non productif (administration, production). Parmi les graphistes et techniciens, de nombreuses compétences sont nécessaires à la fabrication d'un film, et de nombreuses spécialités interviennent durant la fabrication (cf. section I.1.2). Cette diversité existe dans les médiums dits traditionnels comme le dessin animé, mais elle est encore plus prégnante sur les œuvres en images de synthèse.

Rôle de l'informatique dans les studios

Dans la production actuelle de cinéma d'animation, les studios utilisent l'informatique dans différentes mesures. Certains se servent de programmes spécifiques pour simplifier ou accélérer certaines étapes du processus de fabrication, par exemple le montage ou le compositing, y compris sur des productions utilisant un médium physique comme le dessin animé ou la stop motion. Pour d'autres, l'œuvre est fabriquée d'un bout à l'autre par ordinateur, que ce soient des films en image de synthèse ou des dessins animés dessinés par ordinateur. Dans tous les cas, les studios utilisent des logiciels spécialisés, qui constituent leurs outils de production.

Convergence des médiums dans l'animation

De plus, on observe actuellement une convergence dans la fabrication des médiums artistiques utilisant l'informatique, parmi lesquels figurent l'animation, mais aussi le jeu vidéo, la réalité virtuelle, et les « nouveaux médias » [1]. Les techniques ne sont pas les mêmes dans ces différents médias, mais elles ont assez de points communs pour que les acteurs de ces industries créent des liens entre elles, autant du côté des studios, que des développeurs de logiciels, et des financeurs.

De fait, le studio au sein duquel j'ai effectué ma thèse s'intéresse à des formes,

1. À titre d'exemple, la conférence Rencontres Animation Développement Innovation, destinée aux professionnels du secteur de l'animation et se tenant chaque année à Angoulême, a vu des interventions sur le sujet chaque année depuis 2016, concernant la réalité virtuelle et les moteurs dits « temps réel » provenant du jeu vidéo [22].

techniques et médias divers : il a été amené à travailler sur du long-métrage et de la série d'animation, des effets spéciaux, de la modélisation d'objets pour un musée, des applications interactives (jeu vidéo, réalité virtuelle et augmentée). Ces multiples projets sont rapprochés par les similarités des compétences nécessaires à accomplir chaque type d'œuvre. Sans affirmer que ces compétences sont les mêmes, elles sont assez proches dans le domaine de la 3D pour pouvoir les compter dans la définition plus générale de l'animation.

Je me suis intéressé principalement pour cette thèse au film d'animation et à d'autres médias liés (série télévisée et web, muséographie), mais la proximité croissante des médias me fera également faire des incursions dans des médias connexes.

Le studio d'animation Les Fées spéciales

La société Les Fées spéciales est une entreprise basée à Montpellier, dont l'objet est principalement la fabrication de films d'animation, mais également la production et la réalisation. Elle crée aussi d'autres formes d'œuvres audiovisuelles ou interactives, propose des formations aux logiciels pour l'animation, et développe des outils de création graphique. Elle a été fondée en 2015 par quatre professionnels de l'animation sous la forme juridique de SCOP (société coopérative et participative), un des statuts de sociétés coopératives françaises. Ce statut, qui s'insère dans le mouvement de l'économie sociale et solidaire (ESS), a pour particularités une gouvernance démocratique et le fait d'être obligatoirement détenue à plus de la moitié par les salariés de l'entreprise.

Les quatre fondateurs de la structure ont tous une grande expérience de la production de film d'animation, pour avoir travaillé dans divers studios, notamment à Paris. Trois d'entre eux ont quitté cette ville pour s'installer à Montpellier et essayer de faire du film d'animation d'une manière nouvelle, qui se manifeste entre autres choses par le choix du statut coopératif, qui est très rare en France [2] dans ce secteur. Elle se présentait ainsi en 2017, dans un dossier de candidature à un appel d'offre :

2. D'après la base de données Sirene® recensant les entreprises françaises, en 2019 seules trente-neuf sociétés coopératives ont pour objet la production de films et de programmes pour la télévision ou pour le cinéma, ou de films institutionnels et publicitaires, sur un total de 35 714.

> « [Les Fées spéciales] est une SCOP qui fait le vœu d'ouvrir un nouveau chapitre de la fabrication de films d'animation numérique (FAN) dans le sud de la France, à contre-courant de la délocalisation des emplois et de la taylorisation des tâches et contribue ainsi au rayonnement médiatique de la Région Occitanie. LFS adopte une perspective citoyenne, pédagogique, et s'attache à la valorisation du patrimoine culturel. Elle fait le choix d'une production artisanale, éthique et écologique, qui repose sur l'utilisation et l'amélioration, de logiciels libres. En s'investissant également dans la formation, les Fées Spéciales démocratisent l'accès à ces logiciels. »

Je ne détaillerai pas ici toutes les particularités du mouvement SCOP, des statuts de l'entreprise, des aspirations de gouvernance démocratique de ce type de structure. Cependant, il est utile de préciser quelques points pour comprendre l'approche de l'entreprise sur les logiciels libres.

Les sociétés coopératives de production se distinguent de leurs homologues « classiques » par plusieurs aspects juridiques. Elles sont légalement possédées à plus de la moitié par les salariés travaillant pour la structure. Ces derniers peuvent devenir associés (sociétaires) en achetant des parts sociales de l'entreprise. À ce titre, ils acquièrent un pouvoir de décision dans la gestion de la structure, à hauteur d'une voix par personne, contrairement aux autres sociétés, pour lesquelles le pouvoir décisionnaire est à hauteur du nombre d'actions possédées. Il existe une limite à la proportion des bénéfices reversés aux sociétaires, et inversement, une partie des bénéfices doit être réinvestie dans la structure, et une autre partie échoit aux salariés.

Les SCOP ont donc en commun une ambition d'utilité sociale, comme en témoigne la loi de 2014 relative à l'ESS [68]. On peut rapprocher cette ambition de la définition de *logiciel libre*, que nous donnerons plus loin (cf. chapitre I.2), qui se distingue de celle de l'open source (cf. section I.2.3). Retenons pour l'instant que le logiciel libre est à considérer, comme dans la définition de la Free Software Foundation [33], comme un mouvement à portée sociale, correspondant bien à l'ambition de la SCOP.

Dans les statuts de l'entreprise est inscrite la volonté d'utiliser « notamment les

logiciels libres », et cette volonté est généralement paraphrasée au sein de la structure en « autant que possible ». C'est d'ailleurs partant de cette volonté qu'a été initiée ma thèse, qui correspond assez justement au but de déterminer la place possible des logiciels libres dans l'animation.

La recherche et développement aux Fées spéciales

La plupart des jeunes studios d'animation utilisent des techniques, processus et outils déjà connus, des programmes éprouvés dominant le marché et enseignés dans les écoles (cf. section II.2.1). À l'opposé de cette tendance, les Fées spéciales ont choisi dès leur création de mettre en œuvre une stratégie de recherche et développement (R&D*) visant à s'en affranchir.

Cette stratégie a plusieurs conséquences. Elle signifie la création de postes techniques en plus des postes de graphisme et de production. Ces postes ne sont pas forcément directement productifs, mais ils permettent de développer des outils sur mesure, d'apporter de nouvelles possibilités graphiques, ou d'augmenter la productivité des graphistes en automatisant une partie de leur travail.

La création de ces postes a un coût substantiel, mais la société estimait que l'utilisation de logiciels libres nécessiterait de la R&D dès le premier projet (cf. section III.1.1), et que les outils qui résulteraient de ces développements pourraient être valorisés et réutilisés dans de futurs projets. De fait, l'équipe chargée de la R&D est polyvalente et intervient à tous les stades de la fabrication. Cette polyvalence a conduit à parler d'équipe technique, plutôt que R&D, certaines missions étant plus proches du métier de TD* que de celui de chercheur.

Rôle dans la structure

C'est dans cette équipe technique que j'ai été embauché par l'entreprise en convention industrielle de formation à la recherche (Cifre). Mes responsabilités à ce poste étaient donc multiples, centrées surtout sur la R&D*, mais aussi sur quelques tâches de graphisme, et demandaient donc une grande polyvalence. Le point commun de mes

attributions était donc leur côté technique, et on peut grossièrement les diviser en quatre.

— J'ai développé des outils graphiques au sein des logiciels de création, afin de faciliter la tâche des graphistes, en particulier dans les domaines du rig, de l'animation, et du layout.

— J'ai développé et utilisé des scripts* de pipeline* pour permettre des échanges de données.

— J'ai travaillé sur des tâches de graphisme pour des programmes audiovisuels, en particulier celles demandant des compétences techniques de scripting, de cartographie, d'animation procédurale, de conception assistée par ordinateur.

— J'ai enfin fait de l'administration et de la maintenance informatique.

Ces différentes expériences peuvent approximativement être résumées en un sigle : TD*[3]. Chacune d'elles demande d'utiliser des logiciels métiers spécialisés, et chacune a donc été l'occasion de tester des logiciels libres et leurs implications dans la production.

I.1.2 Le pipeline de fabrication

Cette thèse étudie l'impact de l'utilisation et du développement de logiciels libres pour l'animation. Il est important dans ce contexte de définir la notion de pipeline*, qui est un élément fondamental dans la manière de fabriquer les œuvres animées, car le choix des logiciels et processus de développement a une grande influence sur le pipeline, et par extension, sur le travail de conception et de fabrication, et potentiellement sur les œuvres elles-mêmes.

Ce mot anglais faisait à l'origine référence aux tuyaux permettant d'acheminer des fluides sur de grandes distances (aqueducs, gazoducs, oléoducs, etc.). Il est ensuite passé dans le jargon informatique, pour désigner une manière de passer des données d'un processus à l'autre, avant d'être adopté dans le contexte de la création d'image.

La notion même de pipeline est malheureusement ambiguë, et peut désigner plusieurs principes connectés mais distincts. Flavio Perez, dans son mémoire de recherche sur le

3. Dans son mémoire de master [19], Pierre Lelièvre analyse en détail les différents rôles de ce poste dans les studios de post-production français.

pipeline du film d'animation [20], analyse ces différentes définitions et leur emploi dans le contexte de l'animation 3D en France. Il ne parvient pas à en donner une seule définition, en raison de la multiplicité et de la confusion des emplois. Néanmoins, il parvient à la circonscrire à un champ d'intervention à travers deux définitions.

Dans la première définition, plus informelle, on entend par « pipeline » la chaîne de fabrication, le processus permettant de conduire une production depuis l'idée initiale jusqu'à l'œuvre définitive. Dans cette définition, on considère chaque étape, ou chaque département, comme un maillon de la chaîne. Le pipeline est donc une vue d'ensemble du processus de fabrication et de l'organisation du travail : comment le travail réalisé par une équipe est utilisé par l'équipe suivante ; comment une étape dépend d'une ou plusieurs étapes préalables. Cette définition est quasiment synonyme d'un autre terme anglais, le *workflow*, qui désigne plus explicitement le flux de travail, les tâches et opérations des graphistes, et leur interdépendance dans une production.

Une autre définition, qui étend la précédente, se concentre en plus sur la manière dont les données circulent d'un département à l'autre, quelles solutions techniques sont employées, quelles structures de données, quels logiciels, quelle architecture. On parle alors, en complément du *workflow*, du *dataflow*, le flux de données d'un pôle à l'autre. On considère par exemple dans cette définition l'organisation et le nommage des fichiers dans les dossiers de travail ; les formats de fichiers contenant les données, et leur organisation interne ; les programmes avec lesquels on les manipule, c'est-à-dire les logiciels de création mais aussi des programmes et scripts* uniquement dédiés à l'organisation elle-même et permettant de simplifier ou d'automatiser les tâches des graphistes. Cette définition est celle des TD*, dont le rôle est de concevoir, mettre en œuvre et maintenir une chaîne de fabrication optimale, en contrôlant chaque étape et surtout les passages (les tuyaux dans la métaphore plombière) entre les étapes [4].

Dans tous les cas, la notion de pipeline est essentielle car c'est elle qui structure la collaboration au sein d'une équipe. Si un graphiste isolé peut s'organiser comme bon lui semble, ranger aussi bien ou aussi mal ses affaires qu'il l'entend, un studio

4. Une table ronde organisée en 2017 par l'Université de Lorraine aborde le pipeline du point de vue du TD, en particulier du *dataflow* [12].

emploie par essence plusieurs personnes sur un même projet. Ces personnes doivent communiquer avec un langage commun et une organisation commune. Le pipeline est la formalisation, et la mise en œuvre de cette organisation. Il est à noter que même dans le cas où un graphiste serait seul à travailler sur un projet, le fait d'être au sein d'une structure implique que le travail opéré pour le projet pourra à l'avenir être réutilisé, et éventuellement, par quelqu'un d'autre. Il est donc nécessaire, même dans ce cas de figure, de suivre des règles établies dans le studio.

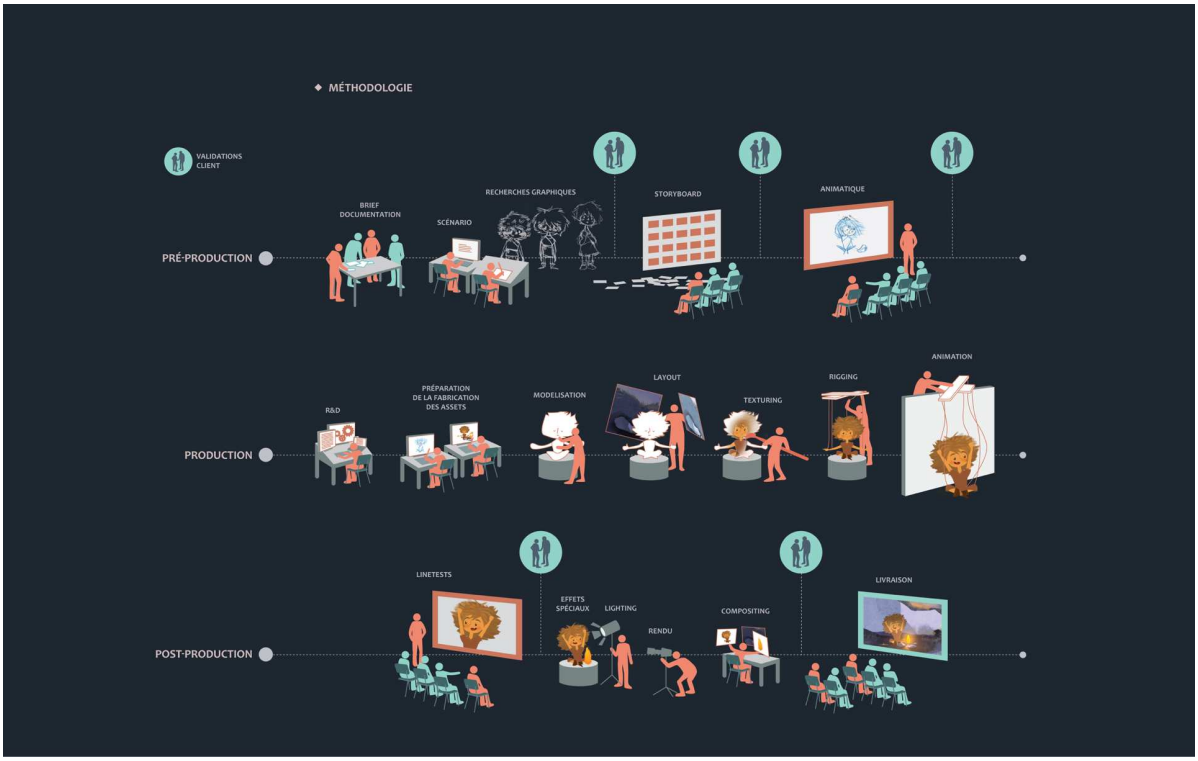


FIGURE I.1.1 – Schéma de la chaîne de fabrication d'un film d'animation. Les crédits photographiques des figures sont donnés dans la table des figures en fin d'ouvrage.

On peut donc considérer le pipeline comme l'ensemble des techniques et étapes d'une production, et les liens entre elles. Je détaillerai ces différentes étapes indépendamment les unes des autres, c'est-à-dire sans donner d'exemple de *workflow* ni de *dataflow* global. D'autres ouvrages, comme la thèse de GEORGE-MOLLAND [18] ou l'ouvrage de MASSON [5] expliquent en détail ces processus, mais il est néanmoins utile de repréciser ici ce qu'on entend par chacune des étapes, et en particulier celles qui sont plus pertinentes dans le

contexte du studio où s'est déroulée ma thèse. Elles sont présentées ici séquentiellement selon un ordre logique, mais elles sont souvent en réalité traitées en parallèle, ou dans un ordre différent ou chaotique.

Les étapes ne sont même pas forcément clairement délimitées entre elles, mais ce qui les sépare en pratique est la validation par la hiérarchie, les responsables de la technique et de la direction artistique, et par le client. Une fois un élément (modélisation, cadrage, animation, plan*, effet, etc.) terminé, on évite de revenir dessus, car cela entraînerait d'incessants allers et retours, particulièrement avec des interlocuteurs perfectionnistes, et nuirait finalement à la production, et à son accomplissement dans les temps et dans le budget. Cependant, les allers et retours peuvent être au contraire prévus dans le budget, ou facturés au cas par cas. C'est souvent le cas dans des plus grosses productions, particulièrement de culture nord-américaine.

I.1.2.1 Préproduction

Scénario et concept

La création d'une œuvre animée, que ce soit un film d'animation ou un autre médium*, commence généralement par l'élaboration d'une idée et d'une histoire. Tout comme dans les films en prise de vue réelle, le scénariste rédige un scénario plus ou moins détaillé pour avoir une idée précise du déroulement du film. C'est un document de travail contenant les dialogues et les indications de mise en scène nécessaires à la compréhension de l'action, telles que les mouvements des personnages, conditions d'éclairage, décors, costumes, etc. Ce document respecte en général un format normalisé [5] et doit permettre d'estimer la qualité de l'histoire et de la narration avant la mise en scène, ainsi que la durée prévue du film.

Le scénario est l'occasion de faire un premier découpage technique du film en séquences*, donnant dès le départ une structure au film, et cette structure aura une

5. Le format normalisé, respecté avec rigueur dans le cinéma en prise de vue réelle, l'est beaucoup moins en animation, sans doute du fait que le médium est intrinsèquement plus plastique. Les auteurs d'animation joignent souvent des illustrations aux scénarios.

influence sur l'organisation de la production. Cette structure peut évoluer en cours de production, selon les contraintes du projet, aussi bien artistiques que techniques et économiques. Une première version du scénario s'avère bien souvent trop complexe et coûteuse à réaliser, et il est alors modifié en accord avec la production pour rendre la fabrication plus faisable.

Le scénariste peut être un collaborateur du studio, de la maison de production, ou un indépendant. Dans la culture francophone du film d'auteur, le scénario est souvent écrit par le réalisateur [6]. Il peut aussi souvent être coécrit par plusieurs personnes.

Selon les cas, l'étape de scénario peut également intégrer un dossier de recherche de financement, soit auprès des maisons de production, soit auprès des collectivités publiques susceptibles d'accorder des subventions.

Le scénario raconte l'histoire du film et sa narration. Dans le film d'animation, ce n'est pas le seul élément qu'il faut connaître pour qualifier le film avant sa fabrication. En effet, l'esthétique graphique est de première importance, et elle est généralement conçue au même moment. C'est l'étape du concept artistique, ou *concept art*. Elle est à la charge de l'équipe de direction artistique, et son but est de décider des codes graphiques qui serviront pour tout le film, et de concevoir des éléments concrets, comme l'apparence et le style des personnages et des décors. Les *concept arts* sont fondamentaux pour obtenir une cohérence graphique tout au long du film, mais ils fournissent également des documents servant de référence au cours de la production, tels que les model sheets* (cf. section I.1.2.2).

Storyboard

Parfois appelé « scénarimage » en français, ce document de mise en scène est un développement graphique du scénario. Se présentant un peu comme une bande dessinée, avec des strips (bandes) de dessins illustrant l'action, il donne de précieuses indications de découpage. Il comprend lui aussi les indications de mise en scène du scénario (dialogues,

6. D'après des chiffres de la base de données IMDb, sur l'ensemble des longs métrages d'animation répertoriés pour lesquels les réalisateurs et scénaristes sont connus, plus de la moitié (55,3 %) recense au moins une personne comme scénariste-réalisateur. Pour plus de détails, voir en annexe D.

actions, etc.), ce qui permet d'avoir une idée de la durée nécessaire pour dire le dialogue du plan*. Le storyboard n'est pas exclusif au film d'animation, et on le trouve aussi souvent en prise de vue réelle, en particulier pour les films figurant beaucoup d'effets spéciaux.

À ce stade de la préproduction, le film est découpé en plan. Chaque plan est décrit par une ou plusieurs vignettes, selon sa complexité. Les mouvements de caméra (panoramiques, zooms, travellings, etc.) et les actions des personnages doivent être montrées avec assez de précision pour être interprétées fidèlement lors de la fabrication. La conception du storyboard nécessite des talents de mise en scène et des qualités graphiques. En général, le réalisateur le dessine lui-même ou travaille étroitement avec un ou plusieurs spécialistes, les storyboarders.

Un studio peut être seulement prestataire sur une partie d'un projet. Dans ce cas, le storyboard peut être fourni par l'auteur-réalisateur, et ne pas faire partie des documents de travail produits en interne. Par exemple, sur le film *Dilili à Paris* (cf. section III.1.1), le storyboard était dessiné sur papier par le réalisateur Michel Ocelot. À l'inverse, le studio peut s'occuper de développer le storyboard, comme ça a été le cas pour les films d'archéologie du musée de Lodève (cf. section III.1.3.1).

Animatique

Une fois le film découpé en plans dans le storyboard, chaque vignette est photographiée et insérée dans un premier montage, constituant une vidéo appelée *animatique*. Ici, de nouvelles indications plus précises peuvent être ajoutées, principalement en ce qui concerne le minutage ou *timing* des plans et des actions. Le montage pourra évoluer pendant la fabrication, mais en principe les plans du film définitif auront sensiblement la même durée que ceux de l'animatique. On peut dire que le montage, dans le film d'animation, a lieu à cette étape, bien avant que les images ne soient fabriquées, et l'animatique est d'ailleurs souvent confiée à un monteur.

Si les dialogues ont été enregistrés avant le début de l'animatique, ils peuvent être utilisés pour fixer la durée des plans. Dans le cas contraire, des dialogues témoins tem-

poraires peuvent être utilisés en attendant la bande-son définitive.

C'est une étape cruciale pour plusieurs raisons. Elle permet de bien se rendre compte de la mise en scène avant de créer les plans, de les dénombrer, de quantifier le temps nécessaire d'animation et de fabrication, et d'estimer la quantité d'éléments à produire. Pour l'œil exercé, l'animatique est suffisante pour comprendre le film en tant que tel. Il faut garder en tête que dans le film d'animation, chaque seconde est fabriquée à partir de rien et est donc coûteuse, et qu'il convient d'animer les plans au plus juste du montage final, dans l'idéal sans rien couper de ce qui a déjà été fabriqué, et donc anticiper ce montage [7]. C'est donc un outil pour la mise en scène, mais également pour la production.

Layout

Le terme anglais de *layout* est un héritage direct d'un département des maisons de production de dessin animé traditionnel [4]. Dans ce contexte, il désignait l'acte de prévoir les mouvements de caméra, les cadrages, les actions des personnages et les minutages de manière extrêmement précise, à l'image près. On consignait ces décisions dans des documents standards, les feuilles d'exposition, qui servaient à toute la suite de la production : les décors, l'animation, la prise de vue. C'est à cette étape, toujours avant le début de l'animation, que se faisait la mise en scène définitive du film.

Dans l'animation 3D, les problématiques sont différentes. Les décors ne sont plus peints à la main, ni les personnages dessinés à chaque image. Néanmoins, le fond du problème reste le même. Il faut bien connaître la mise en scène avant même le début de la production pour savoir quels éléments (accessoires, décors, etc.) il sera nécessaire et suffisant de fabriquer, et combien de séquences*, plans* et images il faudra pour raconter l'histoire. C'est pourquoi le terme s'est transmis en 3D.

En image de synthèse, le layout demande, pour chaque séquence du film, de créer la scène 3D contenant l'action ; d'y placer un décor provisoire mais représentatif du cadre final ; d'y placer une ou plusieurs caméras dont les réglages (focale, position, orientation,

7. Cette affirmation est théorique, mais dans la pratique il arrive de couper des plans au montage, y compris des plans déjà finalisés. C'est particulièrement le cas sur des productions à gros budget, où la coupe de plans, voire le remaniement total du film à quelques mois de sa sortie, s'est déjà vue.

animation) pourront être repris tels quels dans la version finale du film s'ils n'évoluent pas entre temps ; de placer des versions temporaires des personnages et accessoires dans les plans, et de les animer de manière schématique pour donner les indications de timing nécessaires. Une séquence est composée de plusieurs plans, et au stade du layout une scène 3D contient donc une caméra par plan.

Les opérateurs layout créent ces scènes 3D et génèrent des séquences vidéo à chaque étape de validation, avec la bande son si elle a été enregistrée. Ces séquences sont regardées par les équipes layout et par la réalisation et leur servent à discuter de la mise en scène et des éléments. Le rendu effectué à ce stade est loin d'être aussi travaillé que celui qui apparaîtra dans le film final, et il doit seulement être assez lisible pour comprendre la mise en scène. Il est donc nécessaire, contrairement aux rendus plus tard dans la production, d'avoir un rendu rapide plutôt que beau. La plupart des logiciels de 3D permettent d'obtenir ce type de rendu rapide, souvent appelé *playblast* dans les studios, d'après la terminologie de Maya.

I.1.2.2 Fabrication

Modélisation

La modélisation est la première étape où l'on passe véritablement dans la production. Elle consiste à créer les éléments volumiques à partir des documents de préproduction, qu'il s'agisse de dessins de concept, de *model sheets**, ou de références photographiques. On peut la diviser en modélisation de personnages, d'accessoires, de décor et d'environnement [8], qui demandent chacune des compétences différentes, artistiquement ou techniquement : anatomie, botanique, mécanique, etc.

Plusieurs techniques de modélisation ont été développées au fil des ans. Dans les années 1990, la modélisation par NURBS était répandue. Elle permet de contrôler des courbes pour définir des surfaces. Son modèle mathématique permet d'obtenir une pré-

8. Les deux termes « décor » et « environnement » se recouvrent largement, mais l'environnement désigne plus souvent une partie plus lointaine et moins détaillée du décor, parfois réalisée en matte painting.

cision arbitraire. Elle est toujours utilisée en CAO*, mais son usage en animation a pratiquement disparu. En 2019, la technique la plus utilisée est celle de la modélisation polygonale, qui consiste à définir des polygones dans l'espace. Ces polygones sont constitués de sommets, d'arêtes et de facettes, et leur ensemble au sein d'un objet est un maillage. Ce maillage définit la forme de l'objet. Il est souvent subdivisé, afin d'obtenir un plus haut niveau de détail et un maillage moins anguleux, tout en ayant une complexité limitée dans le contrôle du maillage.

La modélisation demande des compétences artistiques pour interpréter fidèlement les concepts et obtenir des objets crédibles ou expressifs, mais également des compétences techniques particulières, afin que la forme soit non seulement juste, mais techniquement exploitable par la suite. Il faut en particulier que l'ensemble des points, arêtes et faces qui composent le modèle, ainsi que la manière dont elles sont agencées (leur topologie), respectent des contraintes rigoureuses de densité, d'orientation des faces, de circulation des lignes de force. Par exemple, pour pouvoir animer un visage, il faut que le maillage soit suffisamment dense pour pouvoir se déformer, mais aussi que les lignes qui composent sa surface suivent les creux et les bosses, et celles selon lesquelles il sera déformé. Pour plisser le front, il faut des rides.

De plus en plus, les modeleurs, en plus de manipuler directement les éléments du maillage, peuvent utiliser une autre technique de modélisation, la sculpture numérique ou *sculpt*, qui permet de manipuler la forme de l'objet de manière plus intuitive, à l'aide d'interfaces rappelant la pratique du modelage. Ces techniques permettent dans une large mesure de s'affranchir des détails techniques de la modélisation au cours de la création, en particulier la topologie, pour se concentrer sur la forme. Une fois cette forme validée, on pourra se préoccuper de topologie dans une étape à part, la retopologie.

Rigging

Le rigging est une tâche très technique au cours de laquelle on prépare les personnages à l'animation. Contrairement au dessin animé « traditionnel », l'animateur 3D ne dessine pas chaque image, mais doit déplacer et déformer le personnage modélisé dans

une scène virtuelle. Pour déformer le personnage, qui peut être un objet géométrique composé de milliers ou de millions de points, il faut déplacer chacun de ces points. Il n'est pas réaliste d'espérer obtenir un mouvement fluide en déplaçant autant de points manuellement, à chaque image. On crée donc, pour faciliter ce travail, un ensemble de contrôleurs qui seront une interface entre l'animateur et le modèle.

Le rigging consiste d'abord à créer et placer des objets virtuels qu'on appelle des os (*bones*) dans le personnage. Ils sont grossièrement analogues aux os réels en ce qu'ils sont en général invisibles et servent de structure pour la locomotion [9]. Cette analogie, quoique pratique, s'arrête là, car les os virtuels ont des propriétés toutes différentes de leurs homologues biologiques. D'abord, comme les autres objets virtuels, ils n'ont pas de propriétés physiques intrinsèques : pas de masse, pas de résistance mécanique, pas même de volume à proprement parler.

Les propriétés qui définissent les os virtuels sont géométriques et logiques. Grossièrement, un os seul est défini par une transformation dans l'espace, qui définit la place qu'il occupe au repos. Le squelette virtuel est constitué d'un ensemble d'os reliés entre eux par des relations hiérarchiques de parent à enfant, ou par d'autres contraintes.

On trouve plusieurs types d'os dans un squelette virtuel. Le premier est le contrôleur.

Les contrôleurs sont les os avec lesquels l'animateur interagit. Ils flottent à la surface ou autour des parties constituant le personnage et devant être animées. Ils correspondent à des unités logiques de cinématique. Par exemple, on trouvera communément un ou plusieurs contrôleurs pour le torse, pour le bras, l'avant-bras et la main, etc. En déplaçant ou en tournant un contrôleur, la partie du corps qui lui correspond y est « attachée » avec la technique du skinning*, et se déplace en même temps. L'animateur peut les saisir et les déplacer pour mouvoir le personnage.

Souvent, l'histoire est plus complexe. Les contrôleurs basiques décrits précédemment ne suffisent toujours pas à obtenir les mouvements souhaités, ou sont d'utilisation encore trop complexe. On peut alors créer des mécanismes qui ajouteront des fonctionnalités au personnage. Par exemple, il faut plusieurs os pour créer une cinématique inverse [10].

9. Manière par laquelle les animaux se déplacent.
10. Technique qui facilite l'animation de la marche ou de la préhension d'objets, en permettant de

De même, un rig facial est complexe et nécessite de nombreux os.

Les « mécanismes » sont des objets qui servent à améliorer le mouvement du personnage, mais sont en quelque sorte « bas niveau », c'est-à-dire qu'ils ne sont pas censés être utilisés directement par l'animateur. Ils sont donc reliés, asservis aux contrôleurs, pour fournir à l'animateur un accès facile à leurs fonctionnalités.

On regroupe les os, qu'ils soient des contrôleurs ou non, dans un objet ou groupe d'objets que l'on désigne en général sous les noms d'armature, de squelette, ou de rig, comme dans l'animation de marionnettes en *stop motion*.

Shading

Le shading, qui se fait souvent en parallèle du rigging, consiste à paramétrer le comportement optique des matériaux à la surface des objets. Dans le monde réel, chaque matériau réagit différemment à la lumière. Un matériau est par exemple plus sombre quand il absorbe une plus grande partie de la lumière incidente, ou plus clair s'il la renvoie. Les métaux renvoient une partie de la lumière de manière spéculaire, concentrée, tandis que des matériaux comme le plâtre ou le papier la renvoient de manière diffuse, dans toutes les directions ; le verre et l'eau transmet une partie de la lumière qu'il reçoit, etc. Pour obtenir une simulation réaliste de ces phénomènes, il faut indiquer au moteur de rendu (voir plus bas) quelle proportion de la lumière reçue par une surface est réémise dans chacun de ces composants, et quelles couleurs ont les surfaces en chaque point.

Surfacing

Une des manières pour spécifier ces couleurs et comportements de la lumière est de peindre les informations à la surface des objets. Pour cela, les graphistent préparent la « surface » des modèles en la dépliant comme un patron de couture (dépliage UV) [11].

déplacer seulement l'extrémité d'un membre, plutôt que toutes les parties qui le constitue. Par exemple, on pourra positionner une main à l'endroit souhaité, plutôt que de devoir faire tourner les articulations de l'épaule, du bras et du poignet.

11. L'étape du dépliage UV est de plus en plus souvent abandonnée, grâce aux possibilités des programmes récents de peindre « directement » sur la surface de l'objet 3D, plutôt que dans un programme de peinture, sur une image qui constituera la texture.

Ils y appliquent ensuite des couleurs et des textures (cartes, ou *maps*) décrivant le comportement de la lumière. Avec ces cartes, les textures, il est possible de créer toutes sortes de matériaux. On distingue en général les matériaux réalistes, imitant le comportement physique des surfaces, des matériaux expressifs, plus plastiques et sans souci de réalisme. Par exemple, les rendus de type cartoon, c'est-à-dire évoquant l'esthétique du dessin animé traditionnel, entrent dans la catégorie des rendus expressifs.

Une autre manière de procéder est d'utiliser des algorithmes pour générer les textures plutôt que de les peindre à la main. On parle alors de textures procédurales. Cette technique permet d'accélérer la création de certains types de textures, mais n'est pas incompatible avec une peinture manuelle. Ce type de texture se développe beaucoup car il permet un prototypage rapide des textures, mais surtout leur réutilisation pour différents assets* 3D. Dans ce paradigme, popularisé notamment par les programmes Substance Painter et Substance Designer [54], il est relativement aisé d'appliquer à la surface, non plus des textures représentant une composante de la surface, mais directement des matériaux (métallique, plastique, etc.).

Animation

Étape la mieux connue du grand public, l'animation consiste, presque littéralement, à insuffler la vie aux personnages. Pour cela, l'animateur fait bouger les personnages au cours du temps, à l'aide des contrôleurs mis en place lors du rigging. On leur donne une succession de poses qui doivent être correctement placées dans l'espace, et correctement minutées, c'est-à-dire espacées dans le temps. On parle communément de *timing*.

Il faut considérer le volume, la gravité, l'inertie, la vitesse et l'accélération, le comportement physique des objets, pour finalement arriver à l'intention des personnages, à un mouvement expressif, transmettant au public l'émotion souhaitée. L'animation est proche de la comédie [12], en ce que l'animateur doit interpréter son personnage de manière crédible pour que le public puisse s'y identifier. Une différence notable est la tempora-

12. Comédie étant entendue au sens de « jeu d'acteur » et non de « forme d'expression destinée à faire rire ».

lité de la création : contrairement à la comédie, l'animation n'est pas « en temps réel », mais chaque phase du mouvement est construite patiemment, membre par membre, pose par pose, parfois image par image. Les animateurs 3D créent de l'ordre d'une à dix secondes d'animation par jour, selon le type de média, le style d'animation, et la qualité souhaitée (souvent imposée par des décisions budgétaires).

L'animation peut se faire en plusieurs phases, de plus en plus précises. On crée d'abord les positions les plus importantes et représentatives du mouvement et de la silhouette (poses clefs ou blocking). On ajoute ensuite des positions intermédiaires (breakdowns), décomposant un peu plus le mouvement, et on termine par les intervalles, qui permettent d'obtenir une animation fluide. Chaque phase peut faire l'objet d'une validation pour vérifier qu'elle est conforme à la narration et à l'esthétique de l'œuvre.

Effets spéciaux

Que ce soit dans les films en prise de vue réelle ou dans les films d'animation, en 2D ou en 3D, on ajoute souvent à l'image des effets [13] spéciaux. Ces effets se répartissent en effets visuels et effets de personnages [14].

Les effets visuels, réalistes ou non, simulent des phénomènes dynamiques, ou renforçent l'image et la narration. C'est un champ très vaste, allant de l'ajout d'effets de fumée dans un dessin animé à la création complète d'un environnement réaliste dans un film tourné sur fond vert ; du maquillage numérique d'un comédien à la simulation réaliste d'un raz-de-marée ou d'un phénomène astronomique.

Dans le dessin animé 2D, il s'agit tout comme pour l'étape de l'animation, de dessiner l'effet image par image, mais les animateurs FX se concentrent sur le phénomène représenté plutôt que sur les personnages.

Dans l'animation 3D, on utilise des logiciels spécialisés pour simuler les phénomènes

13. On emploie souvent l'abréviation « FX » pour parler des effets spéciaux, car elle se prononce presque pareil en anglais que le mot « *effects* ». On peut ainsi parler de SFX (*special effects*) pour les effets spéciaux, de VFX (*visual effects*) pour les effets visuels, et de CFX (*character effects*) pour les effets de personnages.

14. Ces deux types d'effets sont généralement créés dans des départements distincts utilisant des processus différents, mais leur finalité étant similaire, ainsi que le stade du pipeline où ils interviennent, je les place ici dans un même groupe.

physiques dans le même espace que celui de l'action, mais ici encore l'ajout des effets spéciaux se fait généralement après l'animation principale. En 3D, on trouve aussi, en plus des effets visuels, les effets de personnages (CFX, *Character Effects*), dont le but est de simuler les vêtements, cheveux et poils, c'est-à-dire des éléments réagissant à des forces physiques telles que la gravité et les collisions, selon leurs charactéristiques internes comme la masse et la rigidité.

Cette thèse évoque peu les logiciels libres permettant de créer des effets dynamiques 3D, car ce sont globalement les mêmes que les logiciels d'animation [15].

Éclairage et rendu

Une fois que tous les éléments du plan* sont prêts (les caméras sont cadrées, les personnages, les décors et les accessoires sont modélisés et animés), on procède à leur éclairage et à leur rendu.

L'éclairage est une étape où le graphiste spécialisé, le *lighter* (éclairagiste) place des sources de lumière virtuelles dans la scène 3D. Ce travail a le même but que celui de l'éclairagiste de théâtre ou de cinéma, mais ses techniques sont propres à l'animation 3D car il est possible de créer des éclairages qui seraient impossibles dans la réalité.

Le rendu est une étape de calcul au cours de laquelle les données 3D sont transformées en images, par des méthodes de simulation du comportement de la lumière.

Le moteur de rendu est le programme chargé de réaliser cette opération. Il prend en entrée, pour chaque image, les données géométriques de la scène et une description des matériaux, des surfaces et des lumières, et utilise des simulations d'illumination plus ou moins inspirées de l'étude scientifique de l'optique, du transport de la lumière, pour générer des images. C'est un des domaines de l'image de synthèse faisant l'objet du plus de recherches, dans les entreprises comme dans les universités, et de nouveaux algorithmes sont fréquemment intégrés dans les logiciels de création.

On distingue souvent le rendu photoréaliste du rendu non-photoréaliste ou stylisé.

15. Il existe des logiciels dédiés uniquement aux effets spéciaux, mais presque tous les logiciels de création 3D* proposent aussi des modules de simulation d'effets spéciaux plus ou moins complets. Dans les logiciels libres, Blender permet de faire plusieurs types de simulations physiques.

Le premier cherche à reproduire la manière dont les caméras physiques capturent les images, afin de tromper nos yeux pour nous faire croire à l'existence des objets simulées en image de synthèse. Le rendu non-photoréaliste, quant à lui, comprend tout le reste, et en particulier les rendus « cartoon », qui ne cherchent pas à être conforme aux habitudes photographiques, mais à véhiculer efficacement les intentions et émotions, à travers l'apparence plastique des images. Formellement, ils imitent souvent des outils traditionnels : crayon, plume, aquarelle, etc.

Les images rendues peuvent être séparées en couches afin de faciliter l'intégration : les éléments sont répartis en couches (décor, arrière-plan, personnages, etc.), et par composante de lumière (albédo, lumière diffuse, lumière spéculaire, ombre, etc.).

I.1.2.3 Post-production

Compositing

Les couches précédemment rendues ne sont pas les images qui apparaîtront dans le film, mais constituent un matériau de base qui sera assemblé, retraité, amélioré dans une étape appelée *compositing*. On peut y ajouter des effets « 2D » pour rendre l'image plus conforme aux demandes artistiques. Par exemple, il est fréquent dans les films de style photoréaliste de trouver des effets de particules ou d'aberrations optiques (déformation de lentille, aberrations chromatiques, facteur de flare, etc.). De manière générale, toute altération de l'image s'effectue à cette étape. Dans les films en prise de vue réelle et à effets spéciaux, c'est aussi là que les éléments virtuels sont intégrés aux images filmées, en faisant en sorte que le raccord soit invisible à l'image.

L'étalonnage ou *grading*, quant à lui, est une étape de retouche colorimétrique et d'harmonisation finale, en vue de coller au plus prêt aux intentions colorées de la direction artistique, et éventuellement aux spécifications techniques des diffuseurs. Elle permet par exemple, dans une séquence* passant du jour à la nuit, d'harmoniser des plans* aux couleurs disparates, et de créer une continuité entre eux.

Son

Le son est souvent moins considéré que l'image dans l'animation, mais il joue un rôle primordial dans l'esthétique d'un film. C'est la moitié de l'expression « audiovisuel ».

Cette négligence est assez logique dans la mesure où le son est plutôt complémentaire qu'intrinsèque à la création d'image, et demande des savoirs et compétences bien distinctes.

Je ne connais presque rien à ce domaine, ni dans son esthétique, ses enjeux narratifs et suggestifs, ni dans ses techniques de création. De plus, dans les œuvres sur lesquelles j'ai travaillé professionnellement et personnellement, je n'ai pas été en contact avec les personnes chargées du son.

Je me permettrai donc, dans cette thèse, de continuer à négliger cette partie essentielle.

Mastering

Souvent intégré au compositing et rarement considéré comme une étape de fabrication à part entière par les studios d'animation — du moins les petits — le *mastering* consiste à générer, à partir des images et de la bande son définitives, les fichiers livrables au client ou au diffuseur. En général, on confie le mastering à une entreprise spécialisée, un laboratoire cinématographique. Cependant, les studios sont parfois amenés à s'en charger eux-mêmes, notamment pour les projets à petit budget et en vidéo.

Dans ce cas, il est fondamental de respecter strictement les spécifications de diffusion, que ce soit pour le cinéma, la vidéo, la télévision, Internet, la muséographie, etc. Dans le cas contraire, la livraison serait certainement refusée par un client exigeant, ou bien être diffusée telle quelle, en courant le risque que la diffusion soit altérée, ou non conforme au résultat escompté, ou même impossible à diffuser.

Cette étape nécessite de bonnes connaissances techniques des formats de fichiers, conteneurs et codecs, d'espaces colorimétriques, de compression, etc. Dans les petits studios obligés de s'occuper de cette étape, on préfère souvent ne pas trop y penser pendant la production, et on se réveille le jour de la livraison pour lire la documentation, tester

les solutions et résoudre les nombreux problèmes qu'on peut rencontrer. Ces problèmes peuvent être anticipés en ayant connaissance des spécifications.

I.1.2.4 Gestion d'assets et suivi de production

Toutes les étapes décrites ci-dessus sont à peu près distinctes les unes des autres. Les graphistes sont en général spécialisés dans une ou deux d'entre elles et se concentrent, au cours d'un projet, sur une ou quelques tâches restreintes. Mais il existe aussi des postes transversaux, qui interviennent à toutes les étapes de la production, et s'assurent de sa bonne marche.

On peut citer les postes liés à la production (directeurs, chargés, coordinateurs et assistants de production), les postes de supervision, et les postes techniques : la R&D*, et les TD*, qui sont souvent distincts. Si tous ces postes doivent très bien connaître l'ensemble du workflow pour faire un suivi efficace, les TD doivent également connaître le dataflow (cf. section I.1.2), car ce sont en général eux qui le mettent en place.

Rappelons que si le but final d'un projet d'animation est d'avoir une œuvre prête à être diffusée, composée de nombreuses images et d'une bande son, les étapes pour y parvenir demandent la création d'un grand nombre d'éléments, représentés par des fichiers informatiques et plus rarement par des documents physiques. Ainsi, il faut conserver et organiser les scénarios, dessins de concept, storyboards et animatiques. De même, on crée de nombreuses versions des décors, personnages, accessoires et scènes 3D pour chaque plan*, et à chaque étape de production, on génère des images rendues intermédiaires, des fichiers de compositing et de montage, et finalement, les masters. Tous ces fichiers peuvent représenter une somme colossale de données, et une grande complexité. Il faut donc structurer l'organisation de la production. Pour ce faire, plusieurs outils sont nécessaires.

Le premier outil est le gestionnaire d'éléments, ou *asset* manager*[16]. Il s'agit d'un programme spécialisé qui permet d'abstraire et de consolider la gestion des versions, et le

16. On parle volontiers en français de gestion d'assets, et on emploie souvent même l'expression anglaise : asset management.

passage d'une variation à l'autre. Par exemple, dans un pipeline* primitif, la production se ferait du début à la fin dans une même scène 3D. Il apparaît rapidement que cette façon de faire est limitée pour un seul graphiste, et a fortiori ingérable en collaboration. On peut donc créer des versions à chaque modification majeure, en faisant *Enregistrer sous…*, et en renommant le fichier en cours, soit avec un numéro différent, soit carrément un nouveau nom, avec par exemple un suffixe pour le département. C'est un début d'organisation des fichiers selon une nomenclature.

Plutôt que de faire créer des versions manuellement aux graphistes travaillant sur un même projet, l'asset manager permet de faire ces opérations de renommage sans risque de se tromper. Mais l'asset manager peut aller plus loin, en évitant par exemple à plusieurs personnes travaillant dans le même département de travailler sur le même fichier en même temps, au risque d'écraser le travail de l'autre. Pour cela, l'asset manager met en œuvre un système de verrouillage des fichiers. C'est également lui qui sera chargé de créer les variations pour les différents départements : un plan* donné du film gardera son nom tout au long de la production, et plusieurs variations seront créées, pour chaque étape. Par exemple, le plan 1 de la séquence* 1 aura successivement une variation pour le layout, une pour l'animation, une pour le shading, etc. Et pour chacune de ces variations, plusieurs versions existeront, pour garder une trace des changements effectués, et éventuellement pouvoir revenir en arrière en cas de changement d'avis ou de corruption des données.

L'asset manager peut également être chargé, à chaque nouvelle version, d'exécuter automatiquement des tâches. Par exemple, vérifier que la nomenclature d'une scène est conforme à la spécification pour le projet, ou rendre des prévisualisations des plans et les envoyer aux superviseurs pour validation.

Ces différents problèmes peuvent sembler abstraits, mais ils sont omniprésents en production, et l'automatisation est indispensable.

Contrairement aux suites de création, dont une poignée se partage la quasi-totalité du marché, il existe bien quelques solutions de gestion d'assets, mais les studios créent généralement leurs propres outils internes. Le directeur technique des Fées spéciales Fla-

vio Perez explique ainsi les raisons pour lesquelles les studios développent leurs propres outils :

> « C'est bien le manque de solutions commerciales qui a poussé les studios à écrire leurs solutions. Manque de solutions lié à l'impossibilité de formaliser universellement le pipeline.
>
> « Il existe aujourd'hui des solutions mais adoptées timidement : Elara de Foundry n'est encore qu'un concept en circuit fermé ; Shotgun Toolkit est sûrement le plus utilisé ; Tactics ne perce pas malgré le fait qu'il soit ancien.
>
> « On n'utilise plus vraiment les solutions du jeu vidéo (type Alienbrain ou Perforce). Jarvis est une solution d'automatisation mais pas vraiment un asset manager. Est sorti de terre il y a peu Prisme et puis Kabaret [cf. section III.5.1.2] qui est un peu à part (un framework* pas du tout clé en mains) ; Avalon [...].
>
> « Mais on a presque fait le tour et une grande majorité de ces solutions ont moins de 5 ans, c'est très récent donc. [17] »

Il est donc particulièrement intéressant d'étudier les possibilités du logiciel libre dans ce domaine, car les studios peuvent gagner à avoir des outils en commun, le plus standard possible, et que la tâche de les développer ne soit pas redondante. En contrepartie, il faut garder à l'esprit que chaque studio, et chaque projet, est différent, et qu'un outil doit être suffisamment souple pour s'y adapter : l'inverse est rarement possible, dans la mesure où un projet d'animation a en général des singularités impossibles à prévoir.

Le deuxième outil nécessaire à la structuration des projets est le suivi de production. C'est la tâche du chargé de production, qui consiste à élaborer et surveiller le calendrier de production en fonction du budget et de l'avancée quotidienne du projet, et à répartir les tâches entre les graphistes pour optimiser le travail. Une partie du travail de chargé de production se fait en collaboration avec les superviseurs d'équipes, afin de coordonner le travail des graphistes. C'est également le chargé de production qui communique avec le client, pour programmer les validations, recueillir les corrections (retakes*), et les distribuer aux graphistes. L'outil fondamental de visualisation de la production est le

17. Entretien semi-directif avec Flavio Perez, 6 février 2019.

tableur, permettant d'établir, confronter et projeter les chiffres pour que la production soit la plus fluide possible.

Puisque ce travail repose en partie sur les chiffres, le chargé de production doit recueillir les informations à jour de tout le projet, presque en temps réel. Pour cela, il établit un dialogue permanent avec les superviseurs, graphistes et clients. C'est indispensable pour avoir des informations qualitatives sur l'avancée du projet, mais il est également important de disposer d'informations quantitatives : combien de plans* ont-ils été fabriqués cette semaine pour cette séquence* ? combien de décors, de personnages ? combien de secondes d'animation ? combien en reste-t-il ? quelles retakes sont prévues, et quel temps prendront-elles ? quel était le délai initial prévu, et comment celui-ci évoluera-t-il en fonction de l'avancée en cours ? faut-il embaucher des graphistes pour être dans les temps ? Le chargé de production est amené à répondre à ce genre de questions, et une méthode efficace pour recueillir les informations permettant d'y répondre est d'interroger une base de données construite collaborativement par les graphistes, les superviseurs et lui-même. Cette base de données est le logiciel de suivi de production.

Ce logiciel permet donc le suivi par le chargé de production, mais aussi la communication au sein du studio. Par exemple, une fois que le chargé de production a assigné des tâches aux graphistes, ceux-ci peuvent se connecter au logiciel et voir chaque jour quelles tâches ils doivent réaliser, et indiquer en retour le statut de la tâche, selon qu'elle soit seulement prévue, en cours de fabrication, en attente de validation par le superviseur ou le client, ou terminée.

Il permet à tous les intervenants de visualiser l'avancement du projet, en générant des tableaux récapitulatifs. Il peut également servir pour la validation par le client, en affichant par exemple pour chaque séquence en cours, la dernière version de la vidéo, et en entrant directement les critiques, qui pourront être ensuite consultées par les graphistes qui traiteront les retakes.

Ce logiciel est donc aujourd'hui quasiment indispensable pour qu'une production d'une certaine envergure se déroule bien. Il demande de la rigueur de la part de tous, du côté artistique et du côté production, pour remplir correctement les informations au

quotidien. Mais, correctement utilisé, il soulage le chargé de production d'une partie de son travail de recueil des informations.

Le logiciel de suivi de production peut aussi être connecté au gestionnaire d'assets, afin que les informations entrées dans l'un soient copiées dans l'autre. Par exemple, si un graphiste a terminé une version d'un plan et la publie dans le gestionnaire d'assets, cette publication peut être « poussée » dans le suivi de production en même temps qu'une image ou une vidéo, et des informations à jour sur le plan, telles que sa durée, les assets utilisés, etc.

Tout comme les gestionnaires d'assets, il existe des solutions commerciales de suivi de projet, et certains studios développent aussi leurs outils internes.

Conclusion

Les studios d'animation sont des sociétés particulières, autant dans leur activité que dans leur organisation.

Ce chapitre a permis de montrer ces particularités en abordant les différentes facettes opérationnelles des studios, en prenant comme référence le studio Les Fées spéciales : les types de contenus sur lesquels ils sont amenés à travailler, le rôle de la R&D* dans la structuration de la chaîne de fabrication. Afin de mieux définir mon champ de recherche, j'ai exposé mon rôle en tant que TD* au sein d'une telle structure, et la manière dont j'ai abordé autant des aspects techniques qu'artistiques.

J'ai également donné une vue d'ensemble du processus complexe de fabrication, de la préproduction à la post-production, demandant des compétences techniques et artistiques très variées, et utilisant plusieurs logiciels métiers spécialisés à chaque étape.

Chapitre I.2

Le logiciel libre

Introduction

Les questions d'informatique sont centrales dans la création numérique. Par définition, le point commun de tout le champ de la création numérique est de s'appuyer sur l'outil informatique comme matériau de création ou comme médium de présentation. L'outil informatique est tellement omniprésent dans les studios d'animation qu'il est devenu une évidence : on ne fait pas de film sans ordinateur.

Depuis l'écriture du scénario jusqu'à la livraison des masters, toutes les étapes peuvent se faire numériquement, en utilisant des programmes spécialisés. Ces programmes complexes doivent permettre de simplifier ou d'automatiser des tâches, et se plier à la créativité des graphistes. Pour cela, les logiciels doivent tenir compte des besoins de ces derniers, et être conçus exprès pour des applications graphiques.

On trouve dans le commerce nombre de ces logiciels, dont certains sont presque omniprésents dans les studios d'animation [1]. La plupart de ces logiciels sont créés, développés et commercialisés par des éditeurs, et on les appelle « propriétaires ». Cela signifie que leur code source* n'est pas accessible publiquement, et que les utilisateurs n'ont pas le droit d'y accéder et de le modifier. Cette interdiction est généralement vue comme la norme, puisque c'est le cas dans tous les secteurs de l'informatique. On la considère

1. Voir la section II.2.1 sur l'aquisition des logiciels existants.

nécessaire pour que les éditeurs puissent vendre leurs programmes, être rémunérés pour leur développement, et continuer à les améliorer et à les corriger.

Un autre modèle de développement et de mise à disposition des programmes informatiques existe : celui des logiciels libres. Ces logiciels permettent par définition aux utilisateurs de les utiliser comme bon leur semble, d'étudier leur code source pour comprendre comment ils fonctionnent, de les modifier, et de les redistribuer. Comme dans beaucoup de secteurs utilisant l'informatique, il existe des logiciels libres spécialisés dans celui de l'animation.

Les logiciels libres abordés dans cette thèse peuvent être soit des applications telles que Blender, Krita, etc., ou des scripts* et add-ons*. En d'autres termes, on s'intéressera de près aux méthodes, environnements, algorithmes et implémentations permettant à la production d'œuvres de procéder, et à l'artiste d'exercer ses talents.

Nous ferons donc dans ce chapitre un rapide tour de ce que sont les logiciels libres, dans la mesure où ils concernent la création artistique dans le contexte de l'animation. En effet, si leur histoire est encore récente dans la création artistique, cette histoire est riche et complexe. De nombreux concepts proviennent des champs de l'informatique, du droit, en particulier de la propriété intellectuelle, et il convient de les définir correctement.

I.2.1 Historique

Le logiciel libre a une histoire technique, idéologique et sociale complexe et cette thèse n'a pas pour vocation de l'évoquer dans son intégralité, mais seulement de donner les indications suffisantes pour en comprendre les enjeux pour l'animation, autant dans l'utilisation que font les studios de logiciels et formats libres, que dans les modalités selon lesquelles ils peuvent développer et publier eux-mêmes du code sous licence libre.

I.2.1.1 Le logiciel dans le milieu universitaire

À l'origine de l'informatique, les programmeurs étaient des savants, des mathématiciens, des logiciens, des ingénieurs. Ils ont conçu les bases théoriques de l'informatique

selon les mêmes principes que ceux de la recherche, en favorisant une libre circulation des idées à travers la publication d'articles, d'ouvrages, et à travers leurs correspondances et leurs conférences [2].

Au départ, les ordinateurs sont inextricablement liés aux programmes, et l'on conçoit les machines pour une application précise. Dès l'invention d'ordinateurs capables d'exécuter n'importe quel programme [3], on peut échanger des supports contenant ces programmes, qu'il s'agisse des sources du programme imprimées, de cartes perforées, ou de supports plus modernes comme les disques, ou plus tard la transmission directe en réseau. Avant l'arrivée de la micro-informatique et la démocratisation de l'informatique dans les entreprises, puis auprès du public, les ordinateurs sont rares et coûteux, et les programmes livrés avec les machines ne sont pas considérés comme des biens marchands à part entière. Ils sont souvent co-développés par les clients et les constructeurs, qui leur livrent directement les codes sources. La circulation des programmes continue donc à se faire comme avant : librement, sans volonté de les rendre inaccessibles. Jusqu'aux années 1970, les informaticiens des universités de sciences font circuler les programmes auprès des collègues, et par extension, ils les corrigent, les améliorent, et les redistribuent à leur tour. Cette émulation contribue à une évolution rapide de l'informatique.

I.2.1.2 Privatisation et commercialisation

À la fin des années 1960, la situation change quand le plus gros constructeur informatique, IBM, est affecté par une décision de justice très médiatisée, initiée par l'administration états-unienne afin de faire appliquer la législation sur la concurrence, la « loi antitrust ». En effet, la pratique d'IBM de livrer ses logiciels en même temps que ses machines est jugée anticoncurrentielle dans un contexte où la société domine le marché de l'informatique. La décision force donc la société à séparer la facturation du matériel et du logiciel, et le reste des constructeurs s'engage dans cette voie [9]. Le logiciel devient un bien commercial en soi, bien qu'il soit encore livré à cette époque sous forme de code

2. À l'exception des nombreuses innovations militaires, qui sont par nature confidentielles.
3. Dans la limite de ses capacités de mémoire et de puissance de calcul.

source*.

C'est à partir des années 1980 que les logiciels peuvent faire aux États-Unis l'objet d'une protection par le copyright, y compris dans leur version compilée. Le contrôle de la distribution des logiciels revient donc entièrement, comme pour les œuvres de l'esprit qui les ont précédés, à leurs auteurs. Le marché explose dans les années qui suivent, et le logiciel propriétaire devient en moins d'une décennie le modèle de distribution standard. La conséquence est que le logiciel, qui était jusqu'alors une affaire de spécialistes et manipulé presque exclusivement par des informaticiens, devient une commodité et un secret industriel jalousement gardé des éditeurs, et protégé légalement. D'autres juridictions suivent rapidement le mouvement, et dès 1985 la loi française considère les logiciels comme des œuvres de l'esprit, protégées par le droit d'auteur.

I.2.1.3 Richard Stallman et les valeurs de partage

Revenons en arrière, au milieu des années 1970. Dans les laboratoires des universités techniques des États-Unis, en particulier au Massachussetts Institute of Technology (MIT), existe depuis plusieurs années une culture, celle des hackers. Il faut entendre ce terme comme « bricoleur, bidouilleur » et pas du tout comme « pirate » [8]. Les hackers considèrent leur pratique comme un jeu : démonter, étudier, comprendre, modifier, remonter, améliorer… Le terme s'est étendu à l'informatique, et les hackers se posent des problèmes comme des énigmes à résoudre, et redistribuent le résultat sans y penser, comme on donnerait la réponse d'une charade à un copain. Dans ce contexte, il leur semble d'autant plus normal de pouvoir accéder aux sources des programmes qu'ils utilisent, y compris ceux des constructeurs.

L'invention du logiciel libre est due à Richard Stallman [1, 8], qui, devant le développement du logiciel « propriétaire » et la protection du copyright dans les années 1980, regrette l'esprit de libre échange qu'il a connu lorsqu'il étudiait et travaillait au MIT. L'anecdote dit qu'un jour qu'une imprimante dysfonctionnait, il aurait été surpris de ne pas pouvoir la réparer en modifiant les pilotes. En effet, jusque-là les constructeurs avaient toujours rendu les pilotes disponibles. Il s'est senti ce jour-là à la merci

du constructeur, privé de sa liberté de réparer un appareil qu'il avait acquis légalement, d'où l'idée qu'un logiciel qu'on ne peut pas réparer et améliorer soit « privateur ». À lui qui avait connu une situation différente, cette pratique semblait intolérable.

Conformément à son esprit et à sa culture hacker, et dans sa capacité d'excellent développeur logiciel, Stallman a donc cherché un moyen de contourner les lois du copyright afin de pouvoir créer et diffuser des programmes qui ne pourraient pas retirer aux utilisateurs leurs libertés d'utiliser, d'inspecter, de modifier et de redistribuer le code. Il travaille au milieu des années 1980 sur le projet de système d'exploitation (*operating system*, OS) libre GNU [4], comprenant tous les programmes nécessaires pour remplacer les OS propriétaires. Il crée dans ce cadre les premières licences libres pour les programmes qu'il écrit et distribue pour le projet GNU. Bientôt, les différentes licences des programmes sont unifiées en une licence unique, la GNU General Public License (GPL), qui réécrite plusieurs fois, compte encore aujourd'hui parmi les licences libres les plus utilisées.

Le fait qu'une même licence soit utilisée pour de nombreux logiciels permet de les rendre compatibles entre eux, c'est-à-dire que du code écrit pour un programme peut être réutilisé dans un autre, sans que les termes des différentes licences soient mutuellement exclusifs.

Le principe de la licence libre est né, et par la suite d'autres licences libres sont écrites et adoptées par les projets de logiciels libres et open source.

I.2.2 Fondements juridiques

I.2.2.1 Systèmes de protection : brevets et droits d'auteur

Dans les années 1980, quand les producteurs de logiciels commencent à vouloir protéger leur propriété intellectuelle, la législation états-unienne décide de deux mécanismes

4. GNU (GNU's Not Unix !) est le nom donné à un projet de système d'exploitation libre, initié par Richard Stallman et développé par de nombreux contributeurs. Ce projet n'a jamais vu le jour en tant qu'OS complet et indépendant, mais une grande partie des programmes écrits pour GNU sont utilisés dans le système GNU/Linux*.

de protection : les brevets et le copyright [5]. Ces deux mécanismes existaient déjà, dans un cas pour les inventions, et dans l'autre pour les œuvres de l'esprit. Les licences de logiciels sont écrites en termes juridiques pour expliciter les droits conférés aux utilisateurs dans le cadre des législations sur le copyright et les brevets. Il convient d'approfondir les mécanismes sur lesquels reposent les licences libres pour comprendre ce qui leur donne leur force juridique, et desquels découlent toutes les considérations techniques et économiques.

Il y a donc deux mécanismes de protection pour les logiciels : le droit d'auteur[5] et les brevets. Les brevets permettent une protection légale limitée dans le temps, et portent sur des idées, des algorithmes, des prototypes. Le brevet logiciel porte plus spécifiquement sur un logiciel, sur du code. L'obtention d'un brevet garantit, dans le pays où il est enregistré, un monopole d'exploitation de l'idée protégée. Pour qu'un concurrent ait le droit de l'utiliser dans son produit, il doit obtenir une licence, une permission de la part de la personne qui a enregistré le brevet.

Malgré des tentatives de législation dans les années 2000, suite à un intense lobbying des industriels, notamment Microsoft, et une longue lutte juridique, les brevets logiciels n'ont pas été reconnus en Europe. L'ouvrage de Perline et Noisette [6] relate notamment cet épisode, et combien il a influencé l'essor du logiciel libre en Europe. En 2019, la seule protection légale pour les logiciels en Europe est donc le droit d'auteur (ou le copyright, selon les juridictions), et les brevets sont souvent ignorés dans les discussions européennes autour du logiciel libre.

I.2.2.2 Copyright et droit d'auteur

Le copyright et le droit d'auteur sont deux formes de protection des œuvres de l'esprit. Ils sont équivalents, mais ne sont pas mis en œuvre de la même manière selon les pays. Grossièrement, le droit d'auteur a cours dans les juridictions de droit civil comme la France et la Belgique, et le copyright, celles de *common law* comme la Grande-Bretagne

5. L'équivalent du copyright en France est le droit d'auteur. Malgré des différences, j'emploierai les deux termes de manière équivalente. Voir section I.2.2.2.

ou les États-Unis. Dans cette thèse, il n'est pas besoin d'entrer dans les détails, et l'on peut retenir que ces lois ont sensiblement les mêmes fonctions et des implications très similaires.

Le droit d'auteur a pour but de protéger les auteurs d'une part, et les personnes et entités exploitant commercialement les œuvres d'autre part : les éditeurs, les diffuseurs. En France, le droit d'auteur est régi par le Code de la propriété intellectuelle. Une majorité des pays du monde reconnaissent le droit d'auteur ou des concepts similaires. En effet, la convention de Berne, ratifiée en 1886, crée une équivalence mondiale entre les systèmes de protection des auteurs, et garantit donc une protection internationale dès la publication d'une œuvre.

I.2.2.3 Les licences logicielles : droits et restrictions de l'utilisateur

Quand les éditeurs ont commencé à vouloir protéger la propriété intellectuelle de leurs programmes, le droit d'auteur[6] est une des solutions juridiques existantes, au côté des brevets donc, vers lesquelles s'est tournée la législation. Le droit d'auteur est d'abord prévu pour protéger les œuvres d'art et garantit à l'auteur une exclusivité de l'exploitation de l'œuvre. Dans le cas des logiciels, qui ont généralement un but pratique, le copyright seul ne suffit pas à préciser les droits de l'utilisateur sur le programme, comme le précisent Jullien et Zimmermann [13]. Cela conduit les éditeurs à signer des contrats avec leurs utilisateurs, stipulant dans quelles conditions ces derniers peuvent utiliser les programmes. Pour un marché où la reproduction du produit (le logiciel) est triviale et où de grands nombres d'unités peuvent être vendus, ce contrat de licence n'est généralement pas rédigé pour et signé par chaque client ou utilisateur, mais rédigé de manière générique. L'acceptation de ce contrat[7] est alors tacite de la part de l'utilisateur, et il peut soit l'accepter pour utiliser le programme, soit contacter l'éditeur pour tenter

6. Ou le copyright, selon les pays. Voir section I.2.2.2.

7. Ce type de contrat est souvent appelé « contrat de licence utilisateur final », en anglais *End-User License Agreement* ou EULA dans le contexte du logiciel propriétaire.

de signer un autre contrat d'utilisation. Il va sans dire que cette dernière option est rarissime pour l'utilisateur lambda.

I.2.2.4 Licences libres

Les licences libres et open source ne sont pas différentes des autres contrats de licence dans leur nature : elles stipulent les droits que concède l'auteur ou le détenteur des droits, à l'utilisateur. Dans le cas des logiciels propriétaires, les clauses du contrat peuvent décrire le domaine précis d'utilisation, le droit de modification et de distribution du programme, une limitation géographique ou temporelle, le nombre de machines sur lesquelles le programme peut être installé, les données personnelles que collecte l'éditeur, les modalités de mise à jour, etc.

Les licences de logiciel libre définissent elles aussi les droits et limitations de l'utilisateur, mais dans un but globalement différent : tandis que les licences propriétaires ne permettent à l'utilisateur que ce que l'éditeur a prévu, les licences libres cherchent à garantir au plus grand nombre les libertés d'utiliser, étudier, modifier et redistribuer le logiciel. Ces quatre libertés sont en fait au cœur des définitions du logiciel libre et du logiciel open source. Elles seront étudiées plus en détail dans le contexte du secteur de l'animation dans la section II.1.1.

Leurs quatre principes de base, communs par définition à toutes les licences libres, sont que tous les utilisateurs, quelle que soit leur situation nationale, puissent :

— utiliser le logiciel à n'importe quelle fin, prévue ou non par l'éditeur ;
— disposer du code source*, et pas seulement une version exécutable, et pouvoir l'étudier ;
— le modifier, afin de l'améliorer ou de le corriger, d'y ajouter ou retrancher des fonctionnalités ;
— redistribuer leurs modifications à d'autres utilisateurs.

N'importe quel contrat de licence incluant implicitement ou explicitement ces quatre conditions est une licence libre et open source. Chacun peut écrire une licence libre pour son programme, mais la plupart des développeurs de logiciels libres choisissent parmi

les licences existantes, parmi lesquelles GNU GPL et LGPL, MIT (Expat), Apache, etc. Chacune de ces licences offre des particularités, dont certaines sont étudiées dans la section II.1.3. Il y a plusieurs intérêts à choisir une licence répandue. D'abord, ses termes sont connus des autres développeurs et des utilisateurs attentifs, qui pourront connaître rapidement les droits qu'elle leur confère. Ensuite, une licence est un contrat légal liant deux parties. Une licence utilisée depuis longtemps peut être mieux étudiée par les juristes, et offrir une protection avérée [8]. Enfin, il est plus facile de combiner entre eux des logiciels disparates si leurs licences sont compatibles (cf. section II.1.3.1).

Cela peut se traduire aussi par des interdictions ou obligations pour l'utilisateur, qui peuvent être ressenties par certains comme des contraintes privatrices de liberté. Par exemple, la licence copyleft (cf. section II.1.3.1) GPL explique clairement que l'utilisateur qui déciderait de transmettre le programme est dans l'obligation d'en livrer, au minimum, la version non-compilée, le code source*. Ce point de vue est développé dans la section II.1.3.1, mais il est important de comprendre la notion de liberté comme sociale, c'est-à-dire que le but du logiciel libre est d'offrir au plus grand nombre la plus grande liberté possible.

I.2.3 Libre et Open Source : confusion et approches idéologiques

Le logiciel libre est depuis le milieu des années 1990 marqué par un débat terminologique. En effet, le terme de logiciel libre, en anglais *free software*, crée une confusion entre les concepts de gratuité et de liberté, exprimés en anglais par le même mot *free* [9]. Afin de réduire cette confusion, certains auteurs, à commencer par Raymond [7], proposent le nouveau terme d'*Open Source*, rapidement adopté et répandu dans la communauté et au-dehors.

8. Il convient pour s'en assurer de consulter un avocat spécialiste de la propriété intellectuelle.

9. Cette homonymie est fâcheuse, car comme en témoigne la tableau I.2.1, le prix n'entre nullement en considération dans la classification des modèles de licences. S'il est vrai que les logiciels libres sont souvent gratuits, et les logiciels libres souvent payants, d'innombrables contre-exemples existent.

Modèle	Utiliser	Étudier	Modifier	Distribuer	Payant
Logiciel propriétaire	X				Parfois
Source-available	X	X	X		Parfois
Logiciel libre	X	X	X	X	Parfois
Open source	X	X	X	X	Parfois

TABLE I.2.1 – Comparaison des modèles de licences

Ces deux termes sont souvent considérés comme synonymes par le profane, si bien qu'une expression rassemblant les deux, *Free and Open Source Software* ou FOSS, voire *Free/Libre and Open Source Software* (FLOSS), a été utilisée un peu plus tard. Le fait qu'il existe dans la communauté un clivage, et même chez certains une animosité, semble contre-productif. Cela s'explique pourtant par des différences assez profondes dans la manière de concevoir le mouvement. Le courant du logiciel libre est né d'une volonté idéologique de créer une économie de partage et de garantir la liberté des utilisateurs, ce dont témoigne l'adjectif « libre ». L'ouvrage de BROCA [1] développe ainsi la thèse selon laquelle ce mouvement est une nouvelle forme d'utopie sociale. L'expression « open source » est quant à elle beaucoup plus descriptive. Ses défenseurs la qualifient de pragmatique et non idéologique, et mettent en avant l'efficacité des méthodologies de développement issues du logiciel libre, plutôt que son rôle social.

Sur la forme, les deux définitions sont quasiment identiques (cf. tableau I.2.1). Dans les deux cas, elles désignent des programmes dont les sources sont distribuées selon les termes d'une licence qui les rend exécutables, accessibles, modifiables et redistribuables. Les logiciels libres sont également open source par définition, mais le contraire est plus difficile à évaluer, puisqu'il faut entrer dans les domaines de l'éthique, voire de la philosophie politique, et que les critères sont multiples, mal définis et parfois contradictoires. Richard Stallman [37] cite par exemple le cas de programmes utilisant du code open source, mais accessibles uniquement sur des plate-formes non-libres. Il soutient que ces programmes ne sont pas libres parce qu'il est impossible à l'utilisateur de les utiliser d'une manière non prévue par le constructeur, et d'accéder à toutes leurs capacités [10].

Cette distinction, souvent mal comprise, est néanmoins utile à garder en tête pour

10. Cette pratique porte le nom de « tivoïsation » d'après celui d'un produit : le Tivo.

connaître le positionnement des acteurs, notamment des développeurs et éditeurs de logiciels.

Pour brouiller un peu plus encore les définitions, il y a souvent une confusion entre logiciel open source, et logiciel dont les sources sont accessibles, ou *source-available*. Ce modèle de licence offre effectivement la possibilité d'utiliser et d'examiner les sources sous certaines conditions (souvent contre paiement), mais pas les autres libertés de les modifier, et surtout de redistribuer. Il n'a donc rien à voir, ni avec l'open source, ni avec le logiciel libre. Ce modèle sera étudié plus en détail dans la section II.1.1.4.

I.2.4 *Open data*

On assiste depuis plusieurs années à une volonté de la part des instances publiques, mais également d'initiatives privées ou associatives, de rendre accessibles des bases de données, en diminuant les contraintes d'accès. Cela s'applique donc à des bases internationales, gouvernementales ou régionales, et à des associations ou organisations non-gouvernementales dont le but est le recueil, la construction ou la consolidation de données. Il peut s'agir de données scientifiques telles que des photographies, images de synthèse, données d'expériences brutes ou traitées, modèles ; de données cartographiques comme des relevés cadastraux ou des plans d'urbanismes ; de données citoyennes comme les horaires de transports ; de données juridiques et législatives comme les lois et arrêtés, etc.

De la même manière que le logiciel libre utilise et détourne le droit de la propriété intellectuelle pour permettre aux utilisateurs de s'approprier les programmes, il existe des licences proches en esprit et spécialement conçues pour les bases de données. En effet, les bases de données ne sont pas protégées par les mêmes lois que les œuvres soumises au droit d'auteur (œuvres de l'esprit), mais elles bénéficient généralement d'une protection similaire dans le code de la propriété intellectuelle : un monopole d'exploitation est accordé au producteur de la base de données, qui peut donc décider de la rendre disponible selon les termes qu'il entend [2], et donc éventuellement choisir une licence

de base de données ouverte telle que la licence ODbL*.

Ces données peuvent être directement utiles aux studios d'animation, et j'en montrerai deux exemples. Ces deux exemples concrets ont tous deux été utilisés en production, pour des films d'animation ou des installations muséographiques, ce qui montre leur utilité pour l'animation. L'utilisation par les Fées spéciales sera étudiée aux sections III.1.2 et III.1.3. J'ai également utilisé des données ouvertes dans le projet *Music Road*, qui est décrit dans l'annexe B.

I.2.4.1 Données d'élévation de la NASA

L'Administration nationale de l'aéronautique et de l'espace ou *National Aeronautics and Space Administration* (NASA) est l'agence gouvernementale états-unienne responsable du programme spatial états-unien. Il s'agit avant tout d'une agence scientifique qui élabore de nombreuses missions pour étudier l'espace, et la Terre depuis l'espace. Le gouvernement états-unien a pour politique de placer des données d'une partie de ses agences dans le domaine public [11].

La NASA, comme d'autres agences, publie donc beaucoup de données sur ses différents sites [12]. Puisque cette agence dispose de nombreux équipements de haute technologie, notamment des instruments de mesure avancés à bord de satellites, elle est bien placée pour disposer de données uniques et extrêmement utiles à des fins de visualisation et d'étude scientifique, mais également à des fins artistiques.

Parmi ces données se trouve une catégorie très intéressante pour la fabrication d'images de synthèse : les données d'élévation terrestres (figure I.2.1). Il s'agit de mesures de la surface de la planète, se présentant sous la forme d'une liste faisant correspondre à chaque point du globe une mesure de l'élévation, c'est-à-dire la distance verticale par rapport au niveau de la mer. Concrètement, ces données peuvent être représentées sous la

11. Le domaine public est l'absence de toute protection juridique pour des œuvres, les rendant utilisables et accessibles à tous. C'est la forme la plus extrême de licence libre. En France, les œuvres ne sont dans le domaine public que plusieurs décennies après la mort de leur auteur, et celui-ci ne peut pas y placer volontairement ses œuvres. En revanche, d'autres pays le permettent.

12. En particulier le site principal de données ouvertes, `https://data.nasa.gov/` et le site de données scientifiques sur la Terre, `https://earthdata.nasa.gov/`.

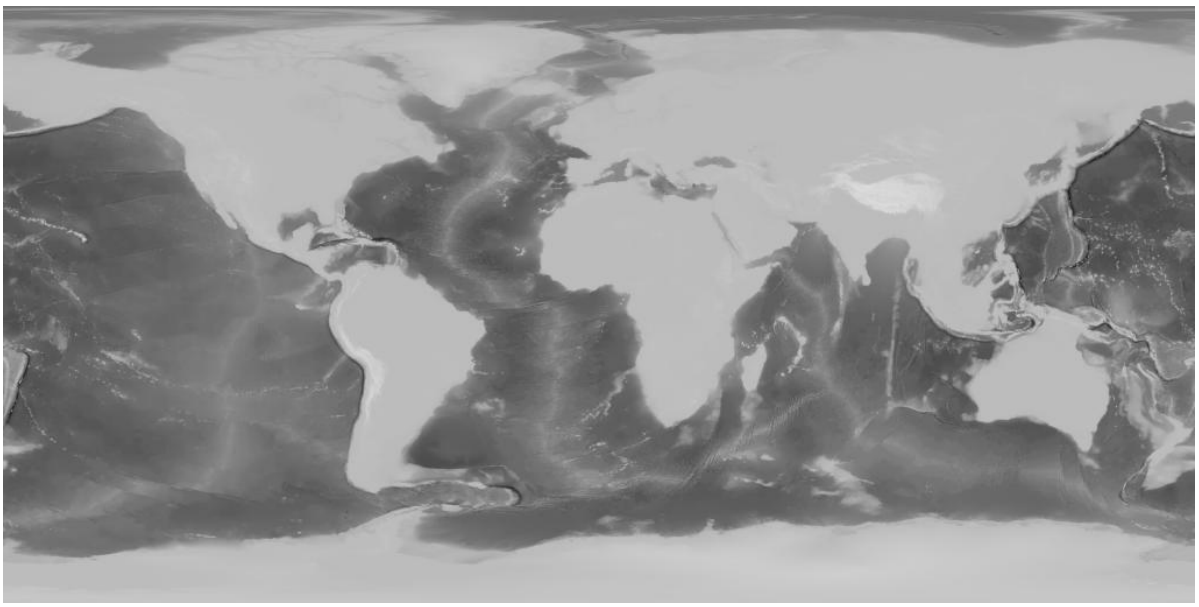


FIGURE I.2.1 – Image composite à partir de deux cartes publiées par la NASA : une carte d'élévation terrestre et une carte de bathymétrie.

forme d'un fichier texte, ou d'une image standard dans laquelle chaque pixel correspond à une zone géographique réduite, et sa valeur à son élévation. Il est donc possible d'utiliser cette image dans un programme de 3D pour créer une sphère déformée localement, avec une grande fidélité visuelle à la Terre.

Un studio d'animation peut se servir de ces données directement pour créer des visualisations, des modèles ou des animations de la Terre (cf. section III.7.1.2).

I.2.4.2 OpenStreetMap

OpenStreetMap (OSM) est une base de données cartographiques commencée en 2004 en Grande-Bretagne. Couvrant la totalité de la surface de la Terre, avec un niveau de précision inégal mais en constante amélioration, elle est le fruit des contributions de millions [13] de volontaires. Chacun peut y contribuer, en proposant des améliorations ou des mises à jour à la base de données.

13. Les statistiques du projets affichent plus de 5 500 000 utilisateurs en 2019. Voir `https://www.openstreetmap.org/stats/data_stats.html`

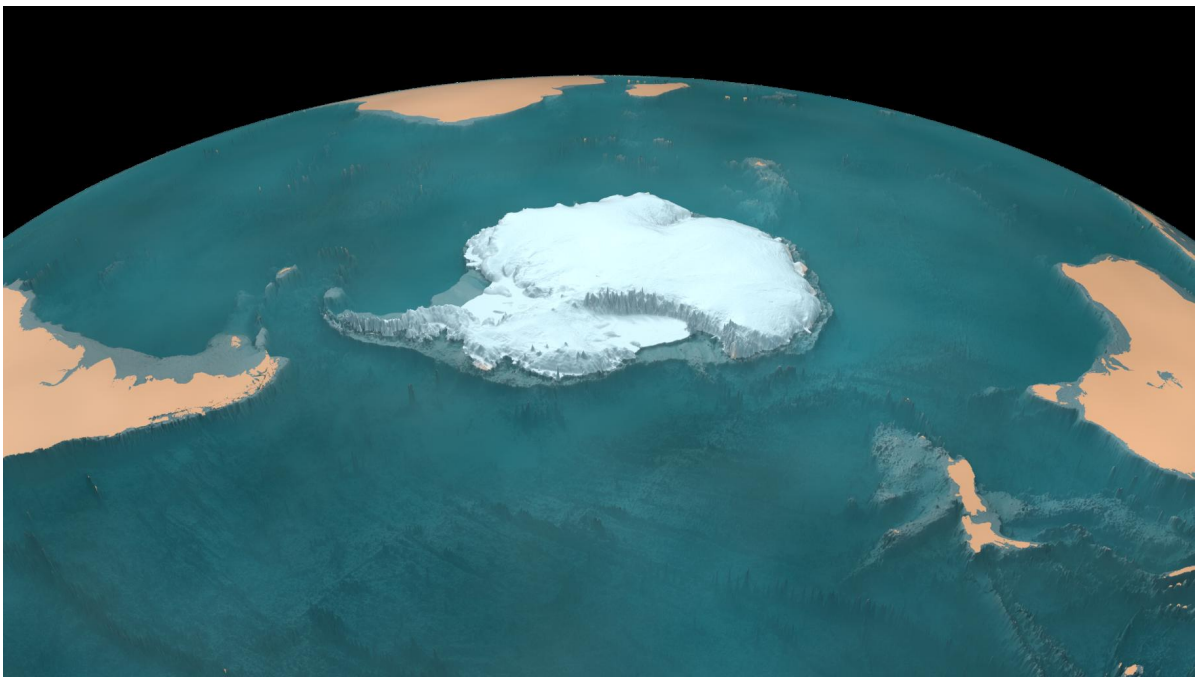


FIGURE I.2.2 – Rendu du continent antarctique avec la surface déplacée par la carte précédente.

Nature de la base de données

Il convient d'insister sur le fait qu'OpenStreetMap est une base de données, et non pas un service en ligne présentant des informations, comme les sites plus connus de Google, Apple et Microsoft (Bing) Maps[14]. Le « produit » offert par OpenStreetMap n'est pas le site web, mais une base de données colossale librement utilisable. Cette base de données peut être utilisée pour plusieurs applications, par plusieurs acteurs, et surtout avec peu de restrictions d'usage. Aussi, le site web openstreetmap.org est *une* de ces applications : il utilise la base de données pour faire le rendu de la carte qui est affiché.

14. On peut arguer que les cartes de Google, Apple ou Microsoft sont également des bases de données accessibles par d'autres moyens que via le site web, à savoir une API* pour les développeurs, permettant de créer des ressources cartographiques dérivées. C'est vrai dans une certaine mesure, mais les conditions d'utilisation du service sont très restrictives. Il est par exemple interdit aux utilisateurs d'extraire des données ou de les garder en mémoire, de créer des cartes dérivées, ou de créer et diffuser des listes de points d'intérêt, le but du service étant de dépendre de l'infrastructure de Google. Il est également interdit de mélanger des sources, en ajoutant des données extraites des cartes Google à d'autres fonds de cartes. De plus, l'utilisation des données est interdite dans certains pays ou territoires. Ces utilisations interdites sont au contraire encouragées par OpenStreetMap. Il est possible de télécharger et de garder à jour l'intégralité de la base de données, pour une utilisation hors-ligne.

Mais d'autres groupes et entreprises offrent aussi des cartes fondées sur les mêmes données. Elles peuvent avoir un autre rendu, une autre langue, utiliser d'autres technologies, viser un public différent, ou simplement offrir un service commercial alternatif, avec le support qui s'y rattache. La base de données est également utilisée pour de la visualisation scientifique, du calcul d'itinéraire, de l'aide humanitaire, ou toute autre application.

Licence de base de données ouverte

Les données d'OpenStreetMap sont publiées sous la licence ODbL* [15], spécialement conçue pour les bases de données. Comme la licence GNU GPL, c'est une licence libre à copyleft fort (cf. section II.1.3.1). On peut donc l'utiliser à n'importe quelle fin, mais à deux conditions : l'attribution et le partage dans les mêmes conditions. Si je veux présenter une version modifiée de la base de données, je dois afficher clairement l'origine des données, et je dois distribuer la version dérivée sous la même licence, ODbL. En revanche, je peux créer une *œuvre* dérivée utilisant la base de données (par exemple un plan* dans un film) en indiquant seulement l'origine des données, sans que ce film ne doive être publié sous la même licence. Cela permet une exploitation commerciale du film, selon les termes que je choisis.

Organisation de la base

Comme souvent avec le contenu libre, son organisation est au départ déroutante, à la fois dans sa mission que dans l'accès aux données. En effet, le site web du projet présente une carte ressemblant aux services en ligne comme Google Maps. Les fonctionnalités sont similaires, quoique plus faibles que celles d'autres services : navigation sur la carte du monde, de l'échelle globale à l'échelle locale ; géolocalisation ; recherche d'adresses et de points d'intérêts ; calcul d'itinéraire ; informations sur des zones géographiques ou des points d'intérêt.

15. On peut trouver la page expliquant les droits associés à la base de données ici : `https://www.openstreetmap.org/copyright`

Tout comme dans Google Maps, l'utilisateur peut créer un compte et proposer une modification de la carte pour l'améliorer, au bénéfice de tous les autres utilisateurs. La différence se trouve dans la manière dont les modifications sont intégrées à la base de données. Elles ne sont pas validées par une société, mais modérées par la communauté, c'est-à-dire par d'autres contributeurs. Tout comme dans Wikipédia où chaque article peut être modifié par tout le monde, la modification apparaît immédiatement et la construction d'OpenStreetMap est organique. La viabilité de ces projets repose sur l'idée que la majorité des utilisateurs veuille y contribuer de bonne foi — et il existe des mesures contre les modifications des utilisateurs malveillants.

Contribution

Chaque utilisateur peut contribuer à la base de données. Ils restent les auteurs de leur contribution, mais la publient sous la même licence que les autres données, et la base dérivée qui en résulte est également libre. Ce n'est pas le cas des contributions que l'on peut faire à Google Maps. Dans les deux cas, j'en reste l'auteur, et je donne une licence irrévocable d'utiliser, stocker, modifier, distribuer, et créer des œuvres dérivées de ma contribution. Mais la différence est que pour OpenStreetMap, je donne cette licence à tous les utilisateurs, tandis que dans le cas de Google Maps, comme spécifié dans ses termes d'utilisation, je n'accorde cette licence qu'à Google, et non aux autres utilisateurs. La société a donc dans les faits un quasi-monopole sur ma contribution, et la contrôle juridiquement et techniquement. Ce contrôle juridique s'applique aussi à l'exploitation commerciale de mes données.

Conclusion

Le logiciel libre est un mouvement qui a pris une ampleur considérable au cours des dernières décénnies, mais dont les principes sont encore mal compris, y compris au sein des studios d'animation qui en utilisent quotidiennement, directement ou indirectement.

J'ai rapidement abordé l'histoire du logiciel libre, pour bien garder en tête ses fonda-

tions idéologiques et pratiques ; les principes juridiques sur lesquels reposent sa diffusion et son élaboration, à savoir le droit d'auteur. J'ai montré la la manière dont les licences de logiciels libres détournent les principes initiaux du droit d'auteur pour donner la possibilité aux développeurs et aux utilisateurs de créer un bien commun, utile à tous.

J'ai enfin abordé un thème connexe, celui des données ouvertes (*open data*), qui reposent sur des principes similaires à ceux des logiciels libres, et donnent des possibilités de création inédites aux studios.

Conclusion de la première partie

Cette première partie a permis d'exposer les deux domaines disjoints qui sont étudiés, comme son nom l'indique, dans cette thèse : l'animation et le logiciel libre.

Dans le chapitre I.1, j'ai abordé les particularités des studios d'animation en tant que structure, et en tant qu'entité utilisant des processus complexes qui reposent sur l'outil informatique. Pour circonscrire ma recherche, j'ai évoqué le studio dans lequel j'ai passé la majeure partie de la durée de ma thèse, Les Fées spéciales.

J'ai également montré une chaîne de fabrication (pipeline*) typique d'un petit studio d'animation, en décrivant chaque étape de la chaîne. Ces étapes sont interconnectées, mais peuvent être indépendantes les unes des autres. Certaines peuvent aujourd'hui être accomplies en utilisant des logiciels libres, d'autres difficilement.

Le chapitre I.2 a permis de décrire ce que constituent le logiciel libre et le logiciel open source — et ce qui les distingue. J'ai rappelé leur histoire au sein de la micro-informatique et les principes juridiques qui les sous-tendent, utilisant le droit d'auteur d'une manière pour laquelle il n'avait pas été pensé antérieurement.

Ces éléments sont indispensables à la suite de cette thèse, afin de comprendre la place que peut avoir le logiciel libre dans l'animation, à la fois dans la simple utilisation de logiciels existants, et dans le développement, la collaboration et la publication au sein des studios de scripts et d'applications, que nous aborderons dans la partie suivante.

Deuxième partie

Enjeux du libre pour l'animation

Introduction

On observe depuis plusieurs années un engouement pour certains logiciels libres au sein des studios. Les raisons de ce choix sont multiples : elles peuvent être liées au coût perçu pour les structures, à la souplesse que leur confère ce modèle de développement et de publication, à la perspective d'une coopération, voire d'une entraide entre les studios, ou simplement à la performance de certains programmes.

Mais au-delà de ces raisons, les problématiques liées à l'utilisation du logiciel libre sont complexes. Une partie des questionnements est plus généralement applicable aux structures utilisant ou contribuant aux logiciels libres, et d'autres sont spécifiques aux activités des studios d'animation. Cette partie explorera ces problématiques, regroupées en plusieurs thématiques, qui porteront sur les facteurs juridique (chapitre II.1), économique (chapitre II.2), et technique (chapitre II.3).

Chapitre II.1

Obligations et libertés juridiques

Introduction

Tout comme un film d'animation, le produit d'un développement logiciel est une œuvre protégée par le droit d'auteur. Et de la même manière que les studios doivent se préoccuper des aspects administratifs et juridiques du droit d'auteur pour publier leurs productions ou simplement engager des auteurs, ils doivent avoir des notions de propriété intellectuelle pour publier ou contribuer à des programmes.

Cette connaissance juridique est d'autant plus fondamentale que dans le cas du logiciel libre et open source, c'est sur elle que reposent les principes de base du logiciel en tant que bien commun, et du développement collaboratif.

Ce chapitre exposera donc l'intérêt pour les studios d'utiliser et de contribuer au logiciel libre, ce qu'ils doivent savoir pour ce faire, et les difficultés juridiques qui y sont attachées.

II.1.1 Les quatre libertés du logiciel libre

Les définitions du logiciel libre et du logiciel open source comprennent toutes deux la nécessité de publier un programme sous une licence offrant à l'utilisateur quatre libertés fondamentales : utiliser, étudier, modifier et distribuer le code source*. Ces quatre li-

bertés peuvent être précisées explicitement dans le texte de la licence, ou implicitement, comme c'est par exemple le cas de la licence permissive What The Fuck Public License (WTFPL) [70] , qui mentionne simplement que l'utilisateur peut faire ce qu'il veut avec le programme.

Ces quatre libertés, déjà mentionnées dans la partie I, sont donc au fondement juridique du logiciel libre et open source, et je détaillerai dans cette section les problématiques associées à chacune d'entre elles pour un studio d'animation.

II.1.1.1 Utiliser

La première liberté associée au logiciel libre est d'utiliser le logiciel à n'importe quelle fin, sans restriction de but ou de domaine d'utilisation. À l'inverse, les logiciels propriétaires, à travers leur contrat de licence, interdisent souvent des utilisations, ou bien les restreignent à quelques utilisations précises.

Nombre d'installations simultanées

Dans la mesure où les conditions d'utilisation sont fixées par les éditeurs, ces derniers peuvent décider qu'une licence donnée ne puisse être installée que sur un nombre limité de postes, ou utilisée par une seule personne. Ainsi, les termes d'utilisation d'un acteur dominant du secteur de l'animation comme Autodesk [27] stipulent dans la version de mai 2018 :

> « Pour toute Offre incluant des Logiciels qu'Autodesk met à votre disposition ou Vous livre, sous réserve du respect des présentes Conditions Générales et de toutes les obligations de paiement, Autodesk Vous concède une licence non-exclusive, non-cessible et ne pouvant être l'objet de sous-licences, pour la durée de Votre abonnement, afin d'installer et d'utiliser les Logiciels (et de permettre à Vos Utilisateurs Autorisés de les installer et de les utiliser) uniquement (i) conformément à la Documentation de l'Offre et à toutes Conditions Particulières applicables, le cas échéant, et (ii) dans le périmètre de Votre abonnement, y compris le nombre autorisé, le Type de Licence, le Territoire et autres spécificités précisées pour le

type et le niveau que Vous avez choisis au moment de la souscription de l'Offre. »

Cette clause signifie que l'utilisation du logiciel est limitée dans le temps, dans le nombre d'installations, et dans la zone géographique [1]. En pratique, cette limitation est appliquée par des moyens techniques de *Digital Rights Management* (DRM), typiquement par un serveur d'identification de licence. Dans certains cas, le logiciel ne pourra pas être exécuté sans connexion à Internet, afin de l'authentifier auprès des serveurs de l'éditeur et empêcher ainsi toute utilisation non prévue par le contrat d'utilisation.

Dans le contexte spécifique du secteur de l'animation, la limitation de durée peut rapidement devenir problématique. En effet, il est fréquent que les studios souhaitent garder une compatibilité avec des fichiers créés sur d'anciennes versions du logiciel, collaborer avec des graphistes utilisant différentes versions, ou simplement terminer un projet avec la version avec laquelle il a été commencé, pour éviter des problèmes de mise à jour en pleine production. Sachant que la fabrication d'un film d'animation peut durer des années, une production peut être amenée à être en retard de plusieurs versions d'un logiciel.

Ce problème a été bien illustré en 2019, et assez largement médiatisé dans le secteur [25], quand l'éditeur Adobe a interdit l'utilisation des anciennes versions de ses logiciels Creative Cloud, les contrevenants pouvant s'exposer à des poursuites. Je n'ai pas eu connaissance de productions d'animation ayant été mises en difficulté par cette annonce, mais elle a néanmoins fait parler d'elle, dans la mesure où un grand nombre de professionnels préfèrent garder d'anciennes versions plutôt que s'exposer à des problèmes de compatibilité.

1. En octobre 2019, dans le contexte de la présidence de Donald Trump, les entreprises États-Uniennes se sont vues interdire toute relation commerciale avec le Venezuela. Une conséquence directe dans le secteur de l'animation a été l'arrêt complet de l'export des logiciels de la suite Adobe (Photoshop, After Effects, etc.). Suite à cette annonce, les utilisateurs devaient voir leurs abonnements « cloud » suspendus, sans remboursement ni négociation possible. Adobe a reçu une dérogation de son gouvernement le mois suivant pour pouvoir continuer à exporter ses produits au Venezuela, mais cette expérience a montré clairement la dépendance industrielle et économique aux éditeurs de logiciels.

Restrictions d'usage

L'utilisateur peut se voir limité aux usages prévus par l'éditeur. C'est le cas par exemple des licences d'évaluation, qui sont en général gratuites et interdisent une utilisation commerciale. Voici un autre exemple, extrait de la version de mars 2020 des conditions générales d'utilisation des produits Adobe [26] :

> « **Utilisation inappropriée.** Vous ne devez pas utiliser de manière inappropriée les Services ou les Logiciels. Par exemple, vous ne devez pas : [...]
> « (E) utiliser les Logiciels pour créer tout type de base de données ou jeu de données ; »

Cette clause vise directement à contrôler le type de contenu que les utilisateurs peuvent créer. Sachant que la limite entre une œuvre et une base de données regroupant plusieurs documents ou œuvres est difficile à déterminer en pratique, cette clause peut potentiellement restreindre la liberté de création des artistes.

Le choix de travailler avec des logiciels libres évacue toutes ces problématiques de restriction d'utilisation imposée par les logiciels propriétaires. En effet, la première liberté associée au logiciel libre est d'utiliser le logiciel à n'importe quelle fin, sans restriction de but ou de domaine d'utilisation. L'utilisateur peut copier le logiciel, l'installer sur un nombre illimité de machines, pour une durée illimitée. Cela lui garantit de pouvoir continuer à utiliser le programme même lorsque celui-ci est déclaré en fin de vie par son éditeur, voire des décennies plus tard, du moment qu'il parvient à compiler ou exécuter le programme sur sa machine. Il peut utiliser le programme pour n'importe quel usage, prévu ou non, déclaré ou non.

II.1.1.2 Étudier

La deuxième liberté offerte par les logiciels sous licence libre et open source est celle d'en étudier le code. On parle souvent de « boîte noire » pour décrire les programmes, fonctions ou modules dont le comportement est connu et éventuellement documenté,

mais dont l'implémentation ou même les algorithmes sont inconnus, voire inaccessibles. C'est le cas de la majorité du code propriétaire, à l'exception des licences dites « *code-accessibles* », qui ne remplissent pas les conditions de l'open source et du libre mais dont le code peut être consulté dans des conditions définies par l'éditeur.

Le code libre et open source est donc par définition accessible, lisible et étudiable. La licence GPL va plus loin en demandant à ce que le code source* soit distribué « sous sa forme privilégiée » [64, §1], c'est-à-dire tel que conçu par l'auteur, sans modification ultérieure ayant pour but de la rendre illisible ou inaccessible [2].

La base du développement collaboratif

Cette liberté présente plusieurs intérêts pour l'utilisateur. D'abord, il est évident que c'est une condition indispensable au travail de développement collaboratif. Les éventuels contributeurs sont bien forcés de prendre connaissance du code pour être en mesure de le corriger, de l'améliorer. L'idée énoncée par Raymond [7], selon laquelle « avec suffisamment d'yeux, tous les bugs sont superficiels », et connue sous le nom de loi de Linus, très répandue dans le monde de l'open source, découle précisément de cette liberté. En étudiant le code, les développeurs qualifiés peuvent repérer régulièrement les bugs et les corriger. Cette possibilité n'existe pas dans le logiciel propriétaire.

Il en découle logiquement les modèles de développement propres à l'open source, que nous étudierons plus en détail à la section suivante dans le cas spécifique du secteur de l'animation (cf. section II.1.1.3).

Valeur pédagogique

Ensuite, l'étude d'un code source de qualité est une manière extrêmement utile d'apprendre à développer. Le code constitue en effet un ensemble d'algorithmes, de méthodes évaluables par la machine, mais on peut également le considérer comme une œuvre des-

2. Cette pratique, l'offuscation, a pour but de se prémunir d'une utilisation non autorisée de propriété intellectuelle ou de secrets commerciaux. Dans le cas de langages compilés, les exécutables constituent en eux-mêmes une offuscation, puisque le code source est difficile à reconstruire. Dans les langages interprétés comme Python, il n'existe pas de distinction formelle entre le code source et l'exécutable. Cependant, il est possible de rendre illisible, ou difficilement lisible, du code interprété.

tinée à être lue par d'autres humains. C'est pour cette raison que les guides de développement informatique liés à des projets ou des groupes de développeurs incluent souvent des recommandations sur la lisibilité du code. Celui-ci est destiné à être lu bien plus souvent qu'il n'est écrit. La documentation a un rôle d'ingénierie, puisqu'elle permet d'expliciter le fonctionnement, l'usage et les détails d'implémentation d'un programme. Elle permet également la transmission des savoirs, décisions, desseins des développeurs à d'autres développeurs, présents ou à venir.

Pour m'appuyer sur un exemple concret, je propose de relater mon expérience personnelle en matière d'apprentissage de la programmation informatique. J'ai appris à développer des scripts* de plusieurs manières. La première, à l'université, m'a été enseignée en licence dans un cours de scripting pour Maya. En parallèle à une nécessaire introduction au langage, la méthode consiste à apprendre les fonctions de l'API* en exécutant les commandes du programme. Après chaque opération, Maya affiche dans une console la commande correspondante. Ainsi, pour reproduire ou automatiser un processus connu, on cherche l'opération dans l'interface, on récupère la commande, et on effectue une recherche dans la documentation de l'API*. On peut ainsi connaître les options disponibles, lire les exemples, et les adapter à la tâche voulue. Cette méthode, si elle est très précieuse et efficace pour le début, ne peut enseigner que des « atomes » de programmation, que sont les commandes individuelles permettant d'élaborer des manipulations de données plus complexes. Elle est très adaptée au scripting de l'API `cmds` dans Maya [3]. Pour apprendre d'autres API et des principes de programmation plus avancés, j'ai trouvé cette méthode insuffisante.

Cela m'amène à la deuxième méthode que j'ai employée pour apprendre le développement. Je l'ai utilisée lorsque j'ai commencé à scripter dans le logiciel d'animation 3D Blender, dont la documentation en matière de scripting, et les ressources pédagogiques étaient à l'époque, vers 2012, succinctes. Il existait bien quelques pages expliquant le minimum pour commencer à écrire des scripts dans l'interface de Blender, ainsi que quelques tutoriaux vidéo. Seulement, ils étaient insuffisants pour comprendre correcte-

3. Cette API Python livrée avec Maya permet de manipuler les données du programme. Elle est conçue selon un paradigme procédural dérivé du langage MEL propre à Maya [30].

ment la logique propre à l'API, sans maîtriser préalablement d'autres API organisées de manière similaire. Il fallait donc consulter les exemples mis à disposition au sein du programme par les développeurs. Ces exemples, bien documentés et commentés, permettaient d'acquérir une meilleure idée des principes, des structures de données et des moyens d'y accéder, des opérateurs, de l'interface graphique, etc. Mais pour pouvoir concevoir des scripts plus avancés, il était nécessaire, puisque je n'avais pas les connaissances préalables, de *lire*, en plus de la documentation, du « vrai » code, de vrais add-ons* fonctionnels et correctement conçus. Une partie de cette démarche se résume à recopier et adapter des bouts de code à ses besoins, mais un autre aspect fondamental est d'être exposé à des problèmes inconnus, et surtout à leur solution. Plutôt que de chercher une solution originale, il est profitable d'apprendre à mettre en œuvre des solutions déjà éprouvées, souvent plus efficaces, et même parfois intégrées au langage de programmation. Il est bien sûr aussi possible de lire intégralement les documentations du langage et de l'API, ce qui est également utile tôt ou tard, mais plutôt indigeste. Le fait de voir une solution utilisée en condition réelle permet de la comprendre beaucoup mieux, et de la retenir.

Cette manière d'apprendre est efficace dans les premiers temps, quand on apprend à programmer, et elle demeure utile plus tard, en production, quand on rencontre un problème nouveau, ce qui arrive souvent dans un secteur aux problématiques de développement très spécifiques comme celui de l'animation. Je continue donc à apprendre, et à programmer de cette manière, et cette expérience m'a fait prendre conscience du rôle joué par le libre dans l'apprentissage et la pratique du développement pour l'animation, qui m'est pratiquement indispensable aujourd'hui.

Cependant, bien qu'il soit utile de lire du code pour apprendre à en écrire soi-même il convient que le code soit de bonne qualité, autant dans sa conception que dans sa documentation.

Qualité du code

S'il est possible pour les développeurs au sein d'un studio d'étudier le code source de programmes publiés sous licence libre, l'inverse est aussi vrai : un studio qui publie son code peut le voir étudié par n'importe qui. Du code destiné à être relu et évalué par d'autres ne sera pas écrit comme du code réservé à l'intimité de l'entreprise. Il en résulte souvent un code de meilleure qualité dans deux domaines :

D'abord, en destinant du code à publication, et par extension à une éventuelle collaboration, les studios d'animation sont forcés à considérer des cas plus généraux que ceux de leur propre production, de leur propre pipeline*. En raison des conditions de production, la publication de code open source par les studios se fait généralement en deux temps. Les programmeurs écrivent d'abord rapidement des outils visant à résoudre des problèmes pratiques rencontrés en cours de production. Un outil jugé utile à un plus vaste public peut ensuite voir son code nettoyé, refactorisé en vue de sa publication. De cette manière, l'outil peut être rendu indépendant des particularités du pipeline du studio.

Ensuite, pour qu'un outil puisse être utilisé par quelqu'un d'autre que par ses concepteurs, il est nécessaire qu'il soit convenablement documenté. Bien sûr, une documentation doit être correcte pour tout type de programme, libre ou propriétaire, mais dans le cas de programmes open source, le code source lui-même peut faire partie de la documentation technique, à travers les commentaires des développeurs.

Or, dans le contexte d'un studio d'animation, la préoccupation générale est plus de publier une œuvre audiovisuelle que d'écrire des programmes. L'écriture de code clairement conçu et bien documenté n'est donc pas prioritaire, a fortiori quand on sait, ou que l'on croit, qu'on écrit du code à usage unique.

En mars 2019, j'ai mené une enquête par entretien semi-directif auprès de développeurs au sein de studios d'animation français collaborant à des projets open source, pour recueillir leurs regards sur le rôle du logiciel libre dans leur pratique du développement pour l'animation [4]. Comme le confirme l'un d'eux, TD* dans un petit studio

4. Un guide d'entretien se trouve en annexe E.

d'animation :

> « […] C'est un autre aspect intéressant de l'open source d'ailleurs : quand tu codes avec l'état d'esprit de partage, tu codes avec le retour de tes pairs en tête. Ça pousse à la rigueur. Mon code pour des trucs persos est carrément moins propre que celui des projets open ! J'aime beaucoup documenter mon code. Avant, pour clarifier les idées, et après pour valider que tout est simple et solide. Mais ma doc préférée c'est les commentaires à l'intérieur du code. C'est l'occasion de laisser des petites blagues, de mettre de l'esprit dans un code, de rendre les choses plus vivantes. Je reçois régulièrement des messages à propos des blagues dans les commentaires, parfois à propos de code que j'ai écrit il y a 10 ans à IMG [Illumination Mac Guff] ! C'est un plaisir et ça me pousse à le faire de plus en plus.[5] »

II.1.1.3 Modifier

Dès lors qu'un développeur a accès au code source d'un programme, il est en mesure de le modifier et de l'exploiter pour ses propres fins, qui ne sont pas nécessairement celles de l'auteur original du programme. Il peut réparer ce qui ne marche pas ou plus, améliorer ce qui pourrait mieux marcher, ou ajouter des fonctionnalités non prévues.

Support technique des développeurs

On constate souvent une méfiance des studios vis-à-vis des logiciels libres, en raison du fait que ceux-ci ne sont généralement pas commerciaux, et qu'ils n'offrent donc pas de support technique régis par contrat. En théorie, le fait de s'acquitter du coût d'une licence auprès d'un éditeur donne au studio le droit à un support technique, qui peut s'étendre à du développement, et à la résolution de bugs en particulier.

Un TD interrogé sur le sujet modère quant à lui cette affirmation :

> « Mais le support sur les softs historiques est en pratique inexistant de toute façon, à moins de payer extrêmement cher en plus du prix des licences. Et même là je

5. Entretien semi-directif réalisé par Internet avec Damien Coureau, 13 mars 2019.

> doute de la qualité des réponses face au bug… Personnellement je vois le [développement] comme un outil fondamental dans la structure d'un studio vfx/anim. [6] »

Lorsqu'on utilise un programme d'animation propriétaire et que l'on rencontre un bug, le seul recours est de contacter le service après-vente pour le leur rapporter, en espérant qu'il soit corrigé dans la version suivante. En attendant, il faut trouver une manière de contourner le bug [7].

On peut donc penser que les développeurs de logiciels libres n'étant pas engagés dans une relation à des clients, ils ne sont pas tenus de s'occuper de la maintenance. De fait, la plupart des licences libres incluent même une clause stipulant que le développeur ne doit *pas* d'assistance à l'utilisateur [8]. Cependant, deux éléments viennent contredire

6. Entretien semi-directif réalisé par Internet avec Damien Coureau, 13 mars 2019.

7. Interrogé à ce sujet, François Grassard, utilisateur du programme After Effects depuis la première version, mentionne le bug suivant, qui est toujours présent dans le programme cinq ans après avoir été signalé :

> « Pour la gestion du cache* disque : dans After, quand tu fais un rendu ou quand tu travailles dans la timeline*, pas forcément en rendu mais juste la RAM preview, il stocke le contenu de chaque calque dans le cache disque. C'est-à-dire qu'il stocke le résultat de rendu d'un calque. Donc il remplit le disque avec un certain nombre de Go maximum que tu définis, et il est censé réutiliser ce cache disque au moment des autres previews bien sûr, mais il est capable aussi de les réutiliser au moment du rendu. Et donc le gros challenge pour eux, c'est de savoir ce qu'ils peuvent réutiliser, et ce qu'ils ne peuvent pas. S'il y a eu des mises à jour, il va faire un rendu qui correspond aux dernières modifications que tu as faites. Et ça, depuis la 2014, depuis qu'ils ont extrait le moteur de rendu du moteur d'interface, c'est-à-dire que le moteur de rendu est un service qui tourne en arrière-plan, et qui n'est pas lié à l'interface, mais l'interface lui lance des appels. Ça permet par exemple de lancer un rendu ou une preview, et de se balader dans l'interface sans que ce soit bloquant.
>
> « Quand ils ont fait ça, ils ont retiré beaucoup de choses : le multiprocessing — même s'ils essaient encore de l'utiliser, mais on peut considérer qu'After n'utilise qu'un cœur — et surtout, le gros problème, y compris avec la version 2019, et j'ai encore eu le problème il y a deux semaines : parfois, il croit que l'image qui se trouve sur le disque correspond à une image en cache, et il se trompe. Donc, tu rends une animation et tu te retrouves avec une image qui a été lue du cache disque et qui n'a rien à voir du tout. Ça peut même venir d'un autre projet.
>
> « Donc quand tu lances un rendu, il faut purger le cache disque et la RAM, et tu n'as plus les avantages que tu avais en cours de travail.
>
> « Il y a encore des problèmes avec ça, ça traîne depuis la version 2014. »

8. Un exemple est donné en annexe C, abordant la question de la documentation publique des outils

cette théorie. D'abord, les développeurs de logiciels libres peuvent être contactés pour rapporter des problèmes, et selon les ressources du projet et la complexité des bugs, ils sont souvent traités rapidement.

Ensuite, les utilisateurs, et les studios en particulier, peuvent participer eux-mêmes à l'effort de maintenance, soit en modifiant directement le programme, soit en le faisant faire par d'autres, soit au minimum en communiquant avec les développeurs.

Réparer soi-même

Par la liberté de modifier le code, les utilisateurs de logiciels open source qui rencontrent un problème peuvent tenter de le résoudre eux-mêmes. Pour cela, on peut ouvrir et lire le code source* pour en comprendre la logique, utiliser des outils de debug, et éventuellement corriger les bugs.

Une fois un bug résolu, on peut envoyer la modification correctrice à l'équipe du programme, sous la forme d'un patch*. Les responsables du logiciel décident ensuite s'ils doivent l'intégrer, en faire modifier des parties, ou le rejeter complètement s'il est trop éloigné des ambitions du projet, ou de trop faible qualité. Dans tous les cas, même si le patch est rejeté, le studio peut garder le correctif pour son usage personnel, et le fournir à d'autres, voire mobiliser la communauté pour le faire intégrer dans le programme.

Une fois qu'un bug a été corrigé, il peut être intégré dans le code source du programme, et lorsque la version suivante sortira, elle inclura donc le correctif. Mais il peut passer du temps entre deux sorties, plusieurs mois, voire plusieurs années selon les projets. L'utilisateur lambda ne profitera donc pas immédiatement de l'amélioration, puisqu'il devra attendre la sortie de la nouvelle version pour pouvoir la télécharger. Cependant, il est aussi possible de profiter immédiatement des améliorations en compilant soi-même le programme dans sa version en cours de développement [9].

Pour citer un exemple concret, le cas de figure s'est produit dans mon travail aux Fées spéciales, pendant la production de *Dilili à Paris*. Au cours du layout, nous importions

des studios lorsqu'ils les publient.

9. Certains projets permettent aussi de télécharger chaque jour une version compilée du programme pour pouvoir utiliser les dernières mises à jour.

dans Blender la bande-son de l'animatique pour vérifier que les mouvements étaient synchronisés avec les dialogues. Pour chaque plan*, nous produisions ensuite un rendu rapide (*playblast*), avec le son. Une fois le layout de chaque séquence* achevé, nous créions un montage incluant les plans. La bande-son de chaque plan était ainsi superposée avec celle de la séquence complète, et aurait dû correspondre parfaitement. Or, il y avait systématiquement un décalage d'une image. Après rapide enquête, l'équipe technique a conclu qu'il s'agissait d'un bug. Pour en avoir la certitude, nous avons ouvert le code source de l'opérateur qui s'occupait d'exporter le son, et avons constaté en effet que la logique était mauvaise. Flavio Perez, responsable technique, a ouvert un rapport de bug* sur le site des développeurs de Blender [10]. Il y explique le problème rencontré et suggère un patch*. De fait, les développeurs n'ont eu qu'à appliquer le correctif suggéré et à tester le logiciel. Sur cette page, il y a peu de commentaires, mais on constate que le bug a été corrigé trois jours après avoir été ouvert, week-end inclus.

Grâce à cela, nous avons pu utiliser la version que nous avions compilée en interne pour continuer notre production. Comme les versions bêta, les versions en cours de développement sont généralement moins stables que les versions officielles, et leur usage n'est pas recommandé en production, mais elles peuvent néanmoins être salvatrices pendant un projet.

Dans l'exemple précédent, le bug était trivial, et nous avons pu le corriger, même sans connaissances du langage de programmation ni du code. Pour des bugs plus complexes, nous aurions été incapables de trouver nous-mêmes la modification à faire, soit par manque de connaissances, soit par manque de moyens, car il faut y consacrer un temps dont on ne dispose pas toujours en production. L'exemple est néanmoins utile pour illustrer la possibilité qu'offre le logiciel libre de résoudre soi-même ses problèmes, et d'en faire profiter les autres utilisateurs.

Il existe d'autres exemples, toujours avec Blender, de modifications plus profondes apportées au logiciel, d'abord par un studio pour ses besoins propres, puis proposées aux développeurs, acceptées et intégrées dans le programme. Quelques-unes sont décrites

10. Le rapport de bug peut être trouvé à cette adresse : `https://developer.blender.org/T49172`

dans la section II.2.2.2.

Faire réparer par d'autres

La possibilité de modifier le code et de réparer un programme entraîne souvent une critique : tout cela est réservé à une élite capable de comprendre le code et de le modifier. Mais cette critique est décalée par rapport à la proposition du logiciel libre. Celle-ci ne dit pas que tout le monde doit être capable de modifier le code, mais que tout le monde en a le droit, quitte à demander à quelqu'un d'autre de le faire.

On peut faire une analogie avec un propriétaire de voiture, qui a le droit d'en ouvrir le moteur pour l'inspecter, et de le réparer s'il en est capable. S'il en est incapable, comme c'est souvent le cas, il a également la possibilité de la mener chez le garagiste, qui pourra faire les révisions ou améliorations nécessaires. En informatique, ce droit qui semblait naturel n'existe plus avec les logiciels propriétaires puisqu'ils ne sont pas la propriété de l'utilisateur [11]. Dans un écosystème de sources fermées, il est impossible d'envisager, comme le garagiste, des prestataires en mesure de réparer, de modifier ou d'améliorer le produit.

En revanche, le logiciel libre donne cette possibilité, et de fait il existe des sociétés dédiées au développement spécifique pour le logiciel libre. Ces sociétés dites « de services en logiciels libres » et détaillées par l'April dans [61], proposent du développement, du support et de la maintenance pour des logiciels métiers libres. Il n'en existe pas à ma connaissance dans le secteur de l'animation, mais elles pourraient se développer.

Transparence du développement

Même pour les utilisateurs ayant une expérience solide de programmation, il est difficile de corriger des bugs dans un programme composé de millions de lignes de code inconnues. Il est donc plus efficace de demander aux développeurs qui connaissent déjà

11. Paradoxalement, cet état de fait s'étend aujourd'hui à d'autres produits que l'informatique, du fait de la présence d'ordinateurs dans un nombre croissant d'objet de consommation, y compris les automobiles.

bien le code de le corriger eux-mêmes. L'organisation du développement open source permet à tous de suivre le développement.

Après avoir fait un rapport de bug*, l'utilisateur peut suivre la conversation dans un fil de discussion sur le site de développement du projet, et voir à quel endroit dans le code source* le problème a lieu. On peut ainsi estimer, en fonction des discussions (ou de l'absence de discussions), si le bug sera corrigé, et si oui avoir un délai approximatif, selon la priorité du bug pour les développeurs, et sa difficulté.

Grâce au développement public et en ligne, les utilisateurs non-développeurs, en particulier les graphistes ou leurs superviseurs qui seraient amenés à faire des rapports de bugs* aux développeurs peuvent au moins bénéficier d'une plus grande transparence qu'avec les logiciels propriétaires.

Un risque pour le développeur-créateur

Un argument de développeur–créateur s'opposant à la liberté de modifier les programmes, est celui donné par Lucas Pope, développeur des jeux vidéo *Papers, Please* et *Return of the Obra Dinn*, dans son journal de développement pour ce dernier jeu. Sur le forum où il publie ce message, il explique les étapes de conception et de développement du jeu. Il utilise la version LT de Maya [12] et se plaint des limitations artificielles de fonctionnalités imposées par cette version [13], par rapport à la version complète, et les contournements qu'il est obligé de réaliser pour atteindre son but. Pourtant, quand un autre membre du forum lui suggère d'essayer Blender, qui n'a pas ce genre de limitations puisqu'il est open source, voici sa réponse [35] :

> « Je suis sûr que Blender, ou n'importe quel autre outil, a aussi des fonctionnalités manquantes. Changer de programme me ferait juste échanger les bizarreries d'un logiciel, et la manière de les contourner, contre celles d'un autre. Ce qui est important pour moi, c'est la difficulté de contourner ces limitations. Pour l'ins-

12. LT signifie *light*, légère. Il s'agit d'une version moins chère du programme, qui n'inclut pas toutes les fonctionnalités de la version complète.

13. Il mentionne par exemple une fonctionnalité manquante, l'impossibilité de transférer l'animation d'un rig à l'autre en utilisant le rig HumanIK.

> tant, Maya LT n'est pas trop mal. Mais je vais certainement réessayer Blender et d'autres programmes après ce projet.
>
> « Personnellement, Blender m'effraie un peu parce qu'il est open source et qu'il n'y a aucune limite à ce qu'on peut réparer soi-même. Au moins, avec Maya LT, je peux dire « c'est irréparable et il faut que je contourne le problème, et que je passe à autre chose ». Avec Blender, je serais capable de passer des jours à modifier les sources ou à écrire des plugins complexes. Amusant, mais contre-productif.[14] »

Ce point de vue est celui d'un créateur de jeu vidéo indépendant qui a un niveau technique élevé pour avoir été développeur d'outils dans plusieurs studios. Il crée ses jeux pratiquement seul, conçoit le game design, et réalise toutes les étapes de création artistique, y compris les plus techniques, le développement, le son et la musique. Dans ce contexte, sachant que la création de ses jeux dure plusieurs années, et qu'il élabore déjà lui-même ses méthodes de fabrication avec ses propres outils, il est logique qu'il ne veuille pas en plus consacrer du temps à développer les suites de création 3D*, sans limite comme il le dit.

De tels arguments pourraient être repris à leur compte par des créateurs de films indépendants, mais le contexte au sein d'un studio (de jeu vidéo ou d'animation) est différent, puisque les équipes sont plus nombreuses. Cela signifie que le développement d'un outil performant et réutilisable est d'autant plus profitable au projet et à la structure, que cet outil sera utilisé de nombreuses fois par de nombreuses personnes.

De plus, le développement est dans tous les cas indispensable aux studios d'animation, et le choix de développer un outil en script* ou de modifier le code du programme appartient aux TD* ou à la R&D*, selon la facilité de mise en œuvre, les connaissances techniques de l'équipe, et les bénéfices techniques espérés. Dans un contexte d'entreprise, la possibilité de se laisser entraîner à du travail intempestif est un problème de management. En d'autres termes, l'acte de modifier les sources du programme est un choix technique, stratégique et économique.

14. Traduction de l'auteur.

Dépendance de l'utilisateur au développeur

Les exemples précédemment cités sont des cas documentés où le développement de nouvelles fonctionnalités par un studio ou la correction de bugs a bien fonctionné, c'est-à-dire que le code a été réintégré dans le programme amont*, et que toute la communauté a pu en profiter. Cela implique que la proposition correspondait aux objectifs du développeur, autant en ce qui concerne le design (architecture, interface), que la qualité du code. Ce cas de figure ne se présente pas toujours, car les développeurs de logiciels libres doivent prendre en compte les besoins perçus de leur communauté, pour les utilisateurs cibles. Il s'ensuit que certaines propositions de modifications ne sont pas retenues. Par exemple, la fondation Blender a pendant des années déclaré que les utilisateurs visés étaient les indépendants solitaires, et éventuellement les petits studios [15]. Les fonctionnalités de pipeline* permettant d'utiliser le logiciel dans un contexte de plus grosse production manquaient, les développeurs de Blender ne considéraient pas de leur responsabilité de les développer. Quelle qu'en soit la raison, et quel que soit le processus de gouvernance, ce sont les développeurs qui décident des critères pour inclure une fonctionnalité.

Pour illustrer, Flavio Perez, directeur technique, raconte avoir souhaité pendant des années l'intégration dans Blender de Qt, un ensemble de bibliothèques* permettant de créer des interfaces graphiques. Qt est très utilisé par les autres logiciels du secteur, au point d'être devenu un standard de fait [16]. Son intérêt est qu'une interface développée avec Qt peut être rapidement portée dans un autre programme intégrant aussi Qt. Cela augmente donc l'interopérabilité* et permet de réduire les coûts de développement. Flavio aurait souhaité voir la bibliothèque intégrée dans Blender, mais ses aspirations n'étaient pas celles des développeurs de Blender. Il raconte sa rencontre avec eux :

> « La première fois que j'ai été à la Blender Conference et que j'ai discuté avec des développeurs et que j'ai dit « C'est super Blender, mais il y a quelques trucs

15. La Blender Foundation, dans une communication officielle au salon Siggraph en 2013, définissait ainsi Blender : « Une solution de création 3D complète, libre et open source, pour les artistes indépendants et les petites équipes. » [23]

16. On parle de « standard de fait » quand une technique n'est pas codifiée par un organisme de normalisation, mais très largement adoptée au sein d'une industrie.

> qui manquent pour que ça fonctionne en équipe [...] », je m'en suis pris plein la vue : « Ah non, c'est pas comme ça qu'on utilise Blender, il faut t'adapter à [sa] façon de faire [...] ». Et ce que je demandais, ce n'était même pas forcément une adaptation de Blender à une standardisation des autres, mais juste [...] laisser la porte ouverte à une autre façon d'utiliser le logiciel, plutôt que de la fermer. Et dire par exemple que si vous intégrez Qt dans Blender, ça veut pas dire que vous allez faire comme les autres, mais qu'on va pouvoir faire plus de choses que vous avez pas prévues. Et ce discours a été très mal entendu.[17] »

En conclusion, malgré les difficultés afférentes à la possibilité de modifier le code source, cette liberté permet aux studios d'animation de résoudre des problèmes ou d'améliorer leurs outils, soit en modifiant eux-mêmes le code pour obtenir leur propre version, soit en demandant à un prestataire de faire ce développement pour eux. Cela réduit la dépendance vis-à-vis des éditeurs, et peut donc permettre aux studios de corriger les problèmes plus rapidement, y compris en cours de production.

II.1.1.4 Distribuer

Cette quatrième et dernière liberté est à la fois celle dont les implications sont les moins souvent comprises, et celle qui donne tout leur sens aux logiciels libres et open source. En effet, sans la possibilité de publier un programme réutilisant du code, la portée en est limitée à un usage privé.

Pour qu'une collaboration entre des développeurs puisse s'établir, pour qu'ils créent ensemble un même programme, il faut que chaque développeur ait la capacité légale de redistribuer une œuvre dont il n'est pas l'auteur original. Pour que chacun puisse apporter sa pierre à l'édifice qu'est le programme, la somme des contributions, il doit exister un moyen légal de créer une œuvre dérivée. Chaque contributeur conserve le droit d'auteur de ce qu'il a écrit, mais offre aux autres, via la licence libre, le droit de l'utiliser.

17. Entretien semi-directif avec Flavio Perez, 6 février 2019.

Le *source-available* n'est pas open source

On comprend souvent l'open source comme un programme dont le code peut être simplement lu, comme du code « ouvert ». Cette condition est insuffisante pour qu'un programme puisse être qualifié d'open source. Pour bien illustrer la distinction, je parlerai d'un autre modèle, celui des programmes à sources disponibles (*source-available*), souvent confondu avec le modèle open source.

Dans ce modèle de distribution, un développeur donne également à l'utilisateur la possibilité de consulter le code source*. La licence qui lie l'utilisateur à l'auteur lui permet de les télécharger, de les étudier et même de les modifier pour son propre bénéfice. Il peut aussi réparer ce qui ne marche pas et améliorer ce qui ne marche pas assez bien. Mais lorsqu'il fait une amélioration, il est le seul à pouvoir en bénéficier ; il ne peut pas en faire profiter son voisin, ou a fortiori le monde entier. S'il décide malgré tout de redistribuer les sources modifiées, que ce soient les sources complètes, une partie du code, ou simplement un patch* permettant d'obtenir la même fonctionnalité, c'est de la contrefaçon, et il s'expose à des poursuites. Dans ces conditions, ce qui fait l'open source, à savoir un contrôle partagé de la distribution, et un développement collaboratif, est impossible.

Le moteur de jeu vidéo d'Epic Games, Unreal Engine, parfois mentionné pour la possibilité offerte aux développeurs d'accéder aux sources, est par exemple disponible selon une licence *source-available*. Dans ce cas précis, la licence d'utilisateur final [32] stipule que l'on peut étudier et modifier le code, mais que la seule entité à qui il est permis de le redistribuer est l'éditeur, Epic. Cela signifie qu'il conserve non seulement un monopole commercial sur ses développements, mais qu'il acquiert également un monopole d'exploitation sur *toutes* les contributions à son moteur.

Un autre point illustrant la différence entre les licences source-available et les licences open source est l'incompatibilité entre les modèles de licences. En effet, quand on évalue des licences logicielles pour le libre, on prend en compte leur compatibilité avec d'autres licences, afin que du code publié sous une licence libre puisse être utilisé dans un autre projet libre, et ce même si les deux projets n'ont pas la même licence. On prend souvent

comme référence la GNU GPL, qui est parmi la plus ancienne, et une des plus drastiques.

Or, une licence comme le contrat d'utilisateur de l'Unreal Engine stipule qu'il est interdit d'y intégrer les licences libres copyleft telles que GPL, ou Creative Commons Attribution Share-Alike, deux licences copyleft fort. La raison est que tout programme incluant du code sous licence copyleft fort est considéré comme une œuvre dérivée, et doit être publié sous la même licence. Dans la mesure où l'utilisateur ne possède pas le droit de redistribuer le code de l'Unreal Engine, il n'est pas en mesure non plus de le distribuer sous licence libre. En revanche, il est tout à fait possible d'inclure du code sous licence libre permissive, telles que les licences Apache, BSD ou MIT, car ces dernières ne demandent pas que le code dérivé soit publié selon les mêmes conditions. Une modification de l'Unreal Engine comprenant du code sous une de ces licences resterait donc la propriété de son éditeur, Epic.

J'ai exposé dans cette section le modèle de distribution source-available, pour mieux expliciter par contraste le modèle open source. La possibilité de redistribuer le code est fondamentale pour permettre le développement collaboratif propre à l'open source, et elle est aussi vitale pour la liberté de l'utilisateur chère au logiciel libre. En effet, sans cette liberté, les améliorations des contributeurs ne leur appartiennent pas, mais à un éditeur, et il est alors impossible d'en profiter comme d'un bien commun.

II.1.2 Pérennité du code et maintenabilité

L'accès à ses propres données

Dans tous les domaines utilisant l'informatique, la pérennité des outils est un facteur fondamental dans le choix d'une solution technique. Si j'adopte un outil dans mon workflow, j'attends de pouvoir l'utiliser pendant longtemps, si possible indéfiniment, ou au minimum que les données que je crée restent compatibles avec d'autres systèmes que je pourrais utiliser plus tard (interopérabilité). L'informatique évolue vite, particulièrement dans le secteur de l'animation, où de nouveaux programmes apparaissent régulièrement, et où d'anciens programmes cessent d'être supportés, parfois du jour au lendemain.

Un exemple typique est celui de Softimage|XSI, un logiciel d'animation très utilisé par les studios d'effets spéciaux ou de jeux vidéo dans les années 2000 – 2010. Il était originellement développé par la société Softimage, Co. Celle-ci a été rachetée plusieurs fois au cours des années 1990, et la dernière fois en 2008 par Autodesk. Suite à ce rachat, le programme a continué à être développé jusqu'en 2014, quand Autodesk a annoncé à la surprise générale que le programme ne serait plus vendu. Cela signifiait aussi qu'il ne serait plus ni développé, ni supporté, ni maintenu. La raison officielle donnée par Autodesk pour cet abandon est que la fin du programme permettrait à l'éditeur de concentrer ses efforts sur ses autres produits, Maya et 3ds Max, et en faire de meilleures solutions.

Les utilisateurs du programme se sont donc retrouvés avec la dernière version du programme, sans possibilité de le voir continuer à s'adapter au marché, aux dernières technologies, ni même de voir les bugs corrigés. En effet, Autodesk est toujours propriétaire du code de Softimage. Les studios ont donc eu plusieurs choix. Ils pouvaient continuer d'utiliser une version vieillissante et fatalement de moins en moins compatible à cause de l'érosion logicielle[18], mais cette solution n'aurait été bonne qu'à court terme, le temps d'un projet ou deux, puis serait devenue ingérable ensuite. L'alternative était donc de changer de logiciel pour les productions suivantes, sans doute selon un remplacement graduel dans le pipeline*. Il va sans dire que quand un pipeline mûr est fondé sur un logiciel donné, il est extrêmement long et coûteux de le remplacer par un autre, puisque les outils de création graphique et les processus d'échange de données doivent être en grande partie, voire intégralement repensés et réécrits, et les graphistes formés à nouveau. L'abandon du développement de Softimage a donc coûté cher à bon nombre de studios, mais également fait prendre conscience de la grande dépendance des utilisateurs envers les éditeurs de logiciels.

Il est fréquent que le développement de logiciels libres soit aussi abandonné, pour diverses raisons : par exemple, faute de financement, ou par manque de temps ou d'intérêt des développeurs. Mais la différence fondamentale est que si le développement d'un

18. L'environnement qui continue à évoluer tandis que le programme stagne.

logiciel libre s'arrête, il est toujours possible à d'autres développeurs ou aux utilisateurs eux-mêmes, de continuer à en faire la maintenance, du moment qu'une copie du code source soit encore accessible. Ils peuvent au minimum corriger les bugs et le garder compatibles, ou continuer à développer des fonctionnalités et à le faire évoluer.

Bien sûr, les logiciels d'animation sont très complexes, et en reprendre le développement en cas d'abandon serait une tâche très lourde. Il faudrait être appuyé par une solide structure financière ou une communauté très impliquée, ce qui pourrait rendre faible la possibilité d'une continuation. Mais avec du code libre, c'est du moins légalement possible.

D'autre part, même si le développement d'un logiciel libre était complètement arrêté du jour au lendemain, les utilisateurs pourraient a minima continuer à se procurer la dernière version en date, et ouvrir ainsi leurs fichiers de production pour les convertir vers d'autres systèmes de création. Même cette possibilité n'est pas garantie avec un logiciel propriétaire s'il n'est plus commercialisé, en particulier lorsque le format de fichier n'est pas libre (sans code source ni spécification publique, ou breveté). De plus, avec un modèle de distributon de type cloud, du moment que le serveur de l'éditeur ne peut plus activer le programme, il devient impossible de l'utiliser, et donc d'accéder à ses anciens fichiers, et à sa propre propriété intellectuelle.

Libération du code

Blender était à l'origine un logiciel propriétaire, développé au sein d'un studio d'animation. Une tentative de commercialisation a échoué, et le logiciel allait être abandonné au début des années 2000, quand une campagne de financement participative a permis de lever des fonds pour racheter à son éditeur les droits afin de le rendre disponible sous la licence libre GNU GPL. On pourrait donc imaginer une solution similaire pour un autre logiciel commercial.

Outre le coût de développement substantiel que représenterait la libération d'un programme comme XSI, un tel changement de licence doit être mûrement réfléchi. En effet, un programme incluant une bibliothèque* non-libre ne peut pas devenir libre sans que

cette bibliothèque ne le soit aussi. L'éditeur voulant libérer du code dans ces conditions devra obtenir qu'elle lui soit fournie sous une licence libre, c'est-à-dire en demander la permission ou négocier avec les auteurs de la bibliothèque. Or, ces derniers n'ont pas forcément envie que leur œuvre soit libre. Ainsi, le responsable de la division « industrie du spectacle » à Autodesk, Maurice Patel, répondant dans un entretien à une question sur l'avenir de Softimage et la possibilité de le vendre ou de le faire développer par une autre société, explique [65] :

> « On ne va pas se contenter de balancer le logiciel [à une autre société] avant de s'enfuir, et nous ne pouvons pas le faire. Il y a trop de choses intégrées au logiciel pour qu'on puisse le faire sans danger. On s'est posé la question de rendre open source, non seulement Softimage mais aussi d'autres applications, mais ce genre de choses n'est pas trivial. Il y a les *codebases*, de la propriété intellectuelle tierce, et il faut qu'on passe tout ça en revue pour comprendre d'où viennent ces propriétés intellectuelles, etc. Donc toute transition demandera de notre part des ressources et la question est : cela vaut-il le coup ? [19] »

Le reste de l'entretien est très éclairant sur les priorités commerciales que donne un éditeur comme Autodesk à ses différentes divisions, et les raisons qui le conduisent à prendre la décision inopinée d'en fermer une. En substance, il explique que la décision a été prise très rapidement pour répondre aux changements rapides de la haute technologie. Il ajoute que malgré tous les efforts de la part d'Autodesk pour réduire les coûts du développement, c'est-à-dire le délocaliser à Singapour, le programme n'était pas assez rentable. Enfin, qu'il était en compétition avec deux autres logiciels du même éditeur, Maya et 3ds Max, pour les ressources de développement. Il conclut sur la question de la possibilité de revendre le logiciel : il explique que la société l'avait acquis pour bénéficier de la propriété intellectuelle et de l'expertise des ingénieurs, et que les concepts de Softimage étaient aussi utilisés dans les autres programmes d'Autodesk. Pour cette raison, la société n'avait aucune intention de le revendre — même si cela provoquait des ennuis pour leurs clients utilisateurs de Softimage.

19. Traduction de l'auteur.

Sans aller jusqu'à dire que le logiciel propriétaire met systématiquement en danger ses utilisateurs, car tous n'ont pas la même politique et que ce type d'abandon est malgré tout assez rare, il n'en reste pas moins indéniable que rien ne garantit aux utilisateurs la pérennité des programmes propriétaires, ni le droit de continuer à accéder à leurs propres données. Il est intéressant de comparer avec le cas de logiciels libres comme Krita ou Blender, qui ne *peuvent* pas subir d'abandon irrévocable, du fait qu'ils soient sous licence GPL. Dût le développement cesser, quelqu'un d'autre aurait toujours le droit de le reprendre, ou bien de récupérer des parties pour les intégrer à un autre programme.

II.1.3 Choix des licences

Pour qu'un logiciel soit considéré comme libre, il faut qu'il soit publié selon les termes d'une licence libre (cf. section I.2.2.4). Parmi les nombreuses licences libres, il est difficile de savoir laquelle ou lesquelles choisir pour publier du code. La langue dans laquelle elles sont écrites est celle des juristes, et leurs implications sont rarement claires pour le profane. De plus, les plus courantes ne sont pas rédigées en français mais en anglais, et qui plus est, conçues pour le copyright états-unien et non pour le droit d'auteur français. Elles peuvent tout de même s'appliquer plus ou moins directement aux œuvres ou aux contributions françaises, mais il faut être vigilant.

Nous aborderons ici les grandes catégories de licences existantes, et celles qui sont utilisées par les créateurs de logiciels libres et par les studios.

II.1.3.1 Licences *Copyleft*

La première catégorie historique de licences libres est dite copyleft. La licence GNU General Public License (GPL) en est l'exemple le plus connu et le plus répandu. Les licences de cette catégorie reposent sur le principe de *copyleft*, qu'on pourrait à peu près traduire par « abandon du droit de copie », et qui est en anglais un jeu de mot avec le copyright, et les mots pour droite et gauche. Elles exposent ainsi la volonté de créer un modèle de propriété intellectuelle différent du modèle traditionnel dans lequel les

auteurs renoncent à une partie de leurs droits sur leurs œuvres, au profit des utilisateurs et d'autres développeurs.

Le principe de base d'une licence copyleft est que toute modification du programme doit être redistribuée selon les termes de la même licence. Cela entraîne plusieurs conséquences. D'abord, c'est une garantie légale que le code reste libre. En effet, si je publie un programme sous une licence copyleft, personne n'a le droit d'en modifier la licence [20]. Si quelqu'un modifie mon programme, et qu'il choisit de redistribuer sa version modifiée du programme, il sera donc obligé d'utiliser la même licence copyleft. Si j'ai choisi par exemple la licence GPL, cela signifie également que le programme devra impérativement être au moins distribué sous forme de code source*, et éventuellement sous forme binaire. De ce fait, les utilisateurs ont la garantie de pouvoir lire le code qu'ils utilisent, de pouvoir le modifier et le redistribuer (cf. section II.1.1).

Ensuite, si je veux inclure du code sous licence copyleft dans un programme, je ne peux le faire qu'à la condition que les licences soient compatibles, c'est-à-dire que les différentes licences n'aient pas de clauses mutuellement exclusives. Typiquement, si j'utilise du code écrit par quelqu'un d'autre et publié sous une licence copyleft, qu'il s'agisse d'une partie d'un programme ou d'une bibliothèque* liée dans une application, je dois impérativement publier également mon programme sous la même licence, puisqu'il est considéré comme dérivé du code que j'ai réutilisé. Cela signifie que je ne peux pas publier mon code sous licence propriétaire. C'était le but initial du logiciel libre de Richard Stallman, qui souhaitait empêcher l'appropriation du code. Cela entraîne aussi que je ne peux pas publier mon programme sous une autre licence open source.

Ce fait suscite des opinions divergentes sur les licences copyleft : d'une part, en empêchant l'appropriation du code, elles garantissent une liberté inaliénable aux utilisateurs. D'un autre côté, elles empêchent d'utiliser le code de n'importe quelle façon, et certains développeurs ont le sentiment d'y perdre une partie de leur liberté.

Un bon exemple est celui du moteur de jeu libre Godot. Il s'agit d'un environnement

20. En tout cas, personne ne vivant dans un pays respectant les droits d'auteur via la convention de Berne. En pratique, la question peut être plus complexe puisqu'elle dépend des systèmes juridiques et de la jurisprudence.

de développement de jeu vidéo, avec un logiciel d'édition permettant de concevoir les graphismes, le son, la logique, et le code du jeu vidéo. Les jeux vidéo sont des programmes, et donc protégés par le droit d'auteur. Ils peuvent donc, comme tous les programmes, être protégés par des licences logicielles. Or, en utilisant un moteur de jeu pour créer son jeu vidéo, le développeur intègre une partie de ce moteur dans le jeu : par exemple, les jeux vidéo utilisant le moteur Unity intègrent du code écrit et possédé par Unity. Pour Godot, cela signifie que le jeu vidéo intègre du code sous licence libre. Si la licence choisie pour Godot avait été copyleft, GPL par exemple, tous les jeux vidéo créés avec Godot seraient nécessairement sous GPL également, puisqu'ils intégreraient du code du moteur. Cela pose problème à l'immense majorité des développeurs de jeux qui souhaitent garder un monopole d'exploitation sur leur jeu, monopole impossible à obtenir avec la licence GPL.

Godot a donc été publié sous une licence permissive (cf. section II.1.3.2), une catégorie imposant comme son nom l'indique des conditions d'utilisation beaucoup moins contraignantes aux utilisateurs.

Copyleft fort et copyleft faible

Dans la catégorie des licences copyleft, il existe une subdivision entre les copylefts dits forts et faibles. La licence GPL évoquée précédemment fait partie de la première catégorie : lorsqu'un programme utilise du code sous licence GPL, il doit nécessairement adopter la même licence, même si le code dérivé n'intègre pas directement le code d'origine. C'est par exemple le cas des bibliothèques*. Une bibliothèque sous licence GPL forcera les développeurs qui feraient appel à des fonctions de la bibliothèque à publier leur code sous la même licence GPL.

La condition de redistribution sous la même licence ne s'applique pas de la même manière pour les licences copyleft faibles. En effet, elles demandent également de redistribuer une version dérivée sous la même licence, mais permettent d'utiliser des fonctions d'une bibliothèque externe sans que les termes s'appliquent à l'ensemble de l'application qui utilise ces fonctions. La Lesser General Public License (LGPL) est un exemple de

licence à copyleft faible, préconisée par ses auteurs de la Free Software Foundation pour les bibliothèques logicielles. Ainsi, si je développe une application et que je lie [21] une bibliothèque sous licence LGPL, je ne serai pas obligé de distribuer mon programme sous la même licence, et je pourrai donc choisir, par exemple, une licence propriétaire. En revanche, je devrai redistribuer le code source des éventuelles modifications que je pourrai faire à la bibliothèque elle-même.

II.1.3.2 Licences permissives

Les licences permissives, au contraire des licences copyleft, ne demandent pas la redistribution dans les mêmes conditions. Il en existe un grand nombre, dont les plus répandues sont les licences MIT (*Massachussetts Institute of Technology*), Apache, BSD (*Berkeley Software Distribution*), chacune pouvant avoir des variations.

Ces licences sont souvent plus courtes, beaucoup plus souples, et permettent en particulier de changer la licence d'une version dérivée. Cela a également des conséquences importantes pour la publication de programmes. Les sources peuvent être intégrées avec d'autres codes sources, qu'ils soient libres ou propriétaires, et devenir donc elles-mêmes propriétaires. Ces licences ne demandent pas une redistribution sous forme de code source. Elles permettent donc de modifier un programme, en général à la simple condition que le programme dérivé inclue les noms des auteurs et les termes de la licence. C'est pourquoi les logiciels propriétaires utilisant du code sous licence permissive donnent accès à une liste de licences et de noms d'auteurs [22], pour chaque composant utilisé dans le programme.

Pour revenir à l'exemple de Godot, publié sous licence MIT, les œuvres dérivées que sont les jeux vidéo peuvent être publiées sous n'importe quelle licence, y compris propriétaires, à condition que les utilisateurs puissent lire le texte de la licence quelque part dans le programme. Cela règle le problème des développeurs voulant exploiter commer-

21. Lier du code à un programme (de l'anglais *link*) est une opération réalisée à l'aide d'un programme spécialisé, l'éditeur de liens, qui crée l'exécutable final à partir des différents composants compilés au préalable. Dans le cas présent, l'exécutable inclut donc la bibliothèque préexistante.

22. Liste que personne ne lit jamais, à part peut-être certains développeurs.

cialement leur jeu vidéo en utilisant un modèle économique propriétaire habituel, et sans que leur programme puisse être modifié plus qu'ils ne le souhaitent.

Comme la majorité des programmes de 3D, Godot s'appuie sur des bibliothèques* pour proposer des fonctionnalités ou faciliter le développement. Le choix d'une licence permissive signifie aussi que les développeurs de Godot ne peuvent pas utiliser de bibliothèque sous licence copyleft sans être forcés d'utiliser la même licence, ce qui nuirait, on l'a vu, à la possibilité des créateurs de jeux de publier sous licence propriétaire.

L'exemple de Godot est intéressant parce qu'il permet d'aborder le statut d'œuvre d'art des programmes. Un jeu vidéo est plus souvent reconnu comme une œuvre d'art que la plupart des logiciels métiers, programmes industriels, ou n'ayant pas de vocation artistique. Il peut donc sembler logique de protéger la vision artistique de son créateur, comme il est rare de voir des réalisateurs accueillir avec bienveillance des changements sur leur film, ou une distribution non autorisée[23].

Il existe bien des jeux vidéo libres, mais ce sont en général des projets financés par les dons, par crowdfunding ou pas financés du tout, et qui n'ont pas pour vocation de créer un monopole d'exploitation. À l'inverse, Godot, comme la plupart des moteurs de jeu, a pour vocation d'être utilisé par des studios pour développer des produits commerciaux, et l'opinion des développeurs de Godot est que cela serait impossible en utilisant une licence copyleft, puisque le jeu serait immédiatement redistribuable par tout acquéreur.

II.1.3.3 Exemple de publication : un script pour Blender

Pour les studios d'animation, ces questions de licences peuvent sembler hors de propos, mais dès lors qu'une partie de leur activité est la publication ou la contribution à des projets open source, il est indispensable de connaître le sujet.

En effet, c'est la licence du programme qui régira les droits accordés aux utilisateurs. Il arrive souvent de vouloir publier un petit script* ou un bout de code, sur un forum par exemple, dans l'idée qu'il puisse être utile aux lecteurs. Le processus n'est pas forcément aussi évident cependant, et outre le choix de la licence, il y a une procédure pour publier

23. Distribution qui constitue légalement une contrefaçon.

du code libre.

Afin d'illustrer le processus du choix et de la publication sous licence libre, j'aborderai l'exemple d'un script* pour le programme Blender, qui présente des particularités intéressantes.

Pour qu'un programme puisse être publié sous licence libre, il faut avant tout qu'il soit protégé par le droit d'auteur, puisque les licences libres utilisent le droit d'auteur pour conférer des libertés aux utilisateurs. Le droit d'auteur s'applique par défaut à toute œuvre originale, mais il vaut mieux que le nom de l'auteur du script soit identifiable dans le code lui-même, généralement dans l'en-tête. En pratique, cette mention ressemble à ceci :

```
# Copyright (C) 2019 Les Fées Spéciales
# voeu@les-fees-speciales.coop
```

Ces lignes sont écrites à l'intérieur du script. Les croisillons `#` signifient dans la syntaxe du langage Python que les lignes en question sont des commentaires, et ne seront donc pas exécutées avec le programme.

Dans le cas d'une société, comme ici par exemple Les Fées spéciales, c'est la personne morale qui est considérée, et non pas les personnes physiques travaillant pour l'entreprise et ayant écrit le script.

Le fait de préciser la paternité de l'œuvre est insuffisante pour conférer de quelconques droits aux utilisateurs, sinon éventuellement la lecture du code. En effet, la protection du droit d'auteur auquel est soumis le code dans de nombreuses juridictions, y compris le droit français, empêche notamment la modification et la redistribution, qui pourrait être qualifiée de contrefaçon. Pour attacher une licence à du code source*, il faut procéder à deux étapes :

1. Mentionner la licence en tête des fichiers sources, et
2. dans le cas où le projet ne serait pas un fichier unique, il est d'usage d'ajouter également à la racine du dossier du projet un fichier nommé `LICENCE`, ou `LICENSE` pour un projet international (anglophone).

Il faut donc nécessairement une licence. En général, les éditeurs de logiciels propriétaires écrivent ou font écrire une licence spécifique pour un logiciel, souvent appelée EULA (*End-User License Agreement* ou accord de licence de l'utiliseur final).

Quand les Fées spéciales ont commencé à publier des scripts sous licence libre, elles ont consulté un avocat spécialiste de la propriété intellectuelle [24], qui a conseillé d'utiliser une des licences CeCILL, corédigées par le CEA, le CNRS et l'INRIA. Ces licences ont un grand intérêt pour les sociétés françaises : elles sont rédigées en français et tiennent compte du droit français, et sont donc mieux à même de faire respecter les termes de la licence devant la loi. Il en existe plusieurs variantes, dont une à copyleft fort [25] (CeCILL-A), une licence permissive (CeCILL-B), et une à copyleft faible (CeCILL-C)[25].

Au début, le code écrit en production a donc été publié sous la licence CeCILL-A. Mais un élément m'a poussé à reconsidérer ce choix, en accord avec les décisionnaires de la société, pour les scripts Blender. En effet, Blender est publié sous la licence copyleft fort GNU GPL. Cela signifie que non seulement le programme, mais également les scripts doivent nécessairement être publiés sous la même licence. La FAQ de Blender [28] explique :

> « If you share or publish Python scripts they have to be made available compliant to the GNU GPL as well, if they use Blender Python API* calls. »

« Si vous partagez ou publiez des scripts Python — s'ils utilisent des appels à l'API de Blender — ils doivent également être disponibles en accord avec la [licence] GNU GPL. » Il y a deux raisons pour lesquelles les scripts doivent être publiés sous cette licence.

D'abord, dès lors qu'un script Blender inclut la ligne de code `import bpy`, il lie la bibliothèque* `bpy` (en terminologie de Python, on parle de *module*). C'est le cas de la quasi-totalité des scripts pour Blender, puisque c'est par ce module que sont exposées les fonctions de Blender. Cela signifie qu'un script Blender est légalement considéré comme une œuvre dérivée du module `bpy`.

24. Comme devrait le faire toute société publiant du code, tant la question peut être complexe.
25. Cf. section II.1.3.1.

Ensuite, la licence GPL étant copyleft, elle inclut une clause empêchant l'utilisateur de republier le code sous une autre licence [64]. En cas de modification et de publication d'un fichier source sous licence GPL, il faut impérativement le rendre disponible sous la même licence.

Puisque le script est dérivé de Blender et que tout dérivé doit être publié sous la même licence, il découle logiquement qu'un script Blender doit être distribué dans les mêmes conditions, sous licence GNU GPL.

Cette conclusion est somme toute d'ordre légale et théorique. En pratique, il est rare que le fait d'utiliser du code trouvé sur Internet aille jusqu'à une action judiciaire. Néanmoins, lors de la publication d'un script Blender sans licence GPL, les auteurs de Blender, principalement la Blender Foundation, mais également de nombreux autres contributeurs, pourraient demander de supprimer le code ou de le publier sous licence GPL. Quoique peu probable, ils pourraient également intenter un procès.

De même, pour un morceau de code sous licence GPL trouvé sur Internet, sur un forum par exemple : si ce bout de code est clairement identifiable et attribué à quelqu'un, et que ce bout code est utilisé dans un programme, l'auteur original peut intenter un procès si le script n'est pas couvert par la même licence.

Bien que faible, le risque existe, et augmente avec l'audience d'un programme, et éventuellement ses revenus.

Une autre raison d'utiliser la GPL, outre l'obligation légale liant le développeur aux auteurs originaux, est que c'est dans l'esprit du logiciel libre : s'appuyer sur du code libre pour utiliser, étudier, modifier et distribuer de nouvelles idées au public.

Cet exemple complexe montre bien les difficultés que présente le choix de licence, et l'intérêt que peuvent avoir les studios à consulter des spécialistes dès lors qu'ils souhaitent publier du code. Chaque situation est différente, selon le pays, le type de programme (application, script, bibliothèque), l'utilisation éventuelle de code couvert par une licence libre, et la vision qu'a l'auteur du logiciel libre et open source.

II.1.4 Œuvres sous licence de libre diffusion

Une question fréquente à l'intersection des milieux de la création et du logiciel libre est celle de la publication de contenus libres, dans d'autres domaines que le logiciel. Dans une réflexion proche de celle des logiciels libres, les licences de libre diffusion ambitionnent de changer la manière d'utiliser le droit d'auteur pour protéger les œuvres, et d'en faire des biens communs plutôt que d'en conserver la jouissance exclusive à leurs auteurs.

Ce type de licences tire son nom du fait que les œuvres les utilisant peuvent être librement diffusées, sans nécessairement correspondre à la définition stricte de la licence libre. Concrètement, certaines licences de libre diffusion permettent de redistribuer une œuvre, mais pas de la modifier, ni de créer des versions dérivées ou des œuvres composites. D'autres permettent la modification, mais interdisent une utilisation commerciale. On pourrait dire que sur les quatre libertés du logiciel libre (cf. section II.1.1), les licences de libre diffusion ne retiennent que celles d'utiliser, d'étudier, et parfois de redistribuer dans certaines conditions. Les licences libres sont donc des licences de libre diffusion, mais l'inverse n'est pas forcément vrai.

Les licences de la famille Creative Commons sont les plus connues parmi celles qui visent à protéger des œuvres autres que du code source. Elles suivent un principe similaire à celui des licences de logiciels libres, mais sont recommandées pour tout type de documents (images, vidéos, sons, textes, etc.). Elles exploitent également les mécanismes de copyright pour permettre de réutiliser, remixer et modifier les œuvres. Elles peuvent en principe répondre à des enjeux sociaux tout comme les logiciels libres, mais elles ont des obstacles propres.

Le logiciel open source existe depuis maintenant des décennies et a démontré son efficacité. Les développeurs y sont acculturés, et il pénètre toute l'industrie informatique. C'est d'ailleurs de ce milieu que proviennent les licences de libre diffusion pour les œuvres. La culture open source ou libre ne s'étend pas encore généralement aux producteurs de contenus, mais des initiatives existent, y compris dans le milieu de l'animation.

La plus visible de ces initiatives est celle de la Blender Foundation, l'organisme initialement chargé de la coordination du développement du logiciel Blender, qui développe

également depuis 2005 des projets de films animés. Ces productions ont plusieurs buts. D'abord, dans le cadre du développement de Blender, d'améliorer la qualité du programme en l'utilisant en conditions réelles, c'est-à-dire pour créer un film. L'équipe des films de la Blender Foundation est en effet composée de graphistes professionnels, engagés pour le temps du projet, et de développeurs. Ils travaillent ensemble : les graphistes rapportent les bugs qui surviennent, et peuvent faire des demandes aux développeurs pour améliorer des fonctionnalités existantes ou en implémenter de nouvelles. Les améliorations faites dans le cadre du projet sont ensuite reversées dans la version publique. Ainsi, la dernière version de Blender, 2.8, a beaucoup bénéficié d'un de ces projets, *Spring*, sorti début 2019. L'équipe a commencé le film en utilisant la version précédente, et basculé en cours de production vers la nouvelle, alors que beaucoup de fonctionnalités précédemment disponibles étaient en train d'être ré-implémentées, que l'interface changeait chaque jour, et que le programme était généralement instable. Cette situation inconfortable pour l'équipe a néanmoins permis au développement d'aller à une grande vitesse, compte tenu de l'ambition de cette nouvelle version.

Ce paradigme de production, consistant à faire un film pour développer un programme, est parfois adopté par d'autres éditeurs, en particulier de moteurs de jeux comme Unity et Epic Games [26]. Il semble évident que ce paradigme n'est pas adapté de manière générale pour les studios, dont l'activité principale est de faire des films et non des logiciels de création.

Le deuxième but affiché par les projets de la Blender Foundation est de créer une vitrine technologique pour le programme, en montrant qu'il est possible de créer du contenu de haute qualité avec Blender et d'autres logiciels libres. Ce deuxième but a été largement atteint, comme le montrent plusieurs indicateurs. Les films de la Blender Foundation, s'ils ne sont pas aussi connus que les courts-métrages du studio Pixar, par exemple, ont néanmoins joui d'une forte diffusion sur Internet et au-dehors, avec des millions de vues sur les plate-formes de vidéos comme YouTube. En effet, outre leurs

26. Il existe au sein d'Unity une équipe spécifiquement dédiée à la production de films et au développement du programme dans cette optique. Cette équipe a réalisé un film en utilisant et en développant Unity, Adam [42].

qualités techniques et artistiques, ils en ont une autre, beaucoup plus rare : la licence sous laquelle ils sont publiés, qui permet leur diffusion et leur réutilisation sans devoir obtenir une autorisation. Ils ont été utilisés pour tester des algorithmes de traitement d'image ou de vision par ordinateur, pour des démonstrations d'écrans et de téléviseurs dans des salons, etc. Ils ont donc une grande valeur pour différentes industries, du fait de leur libre diffusion.

Leur financement n'est pas non plus tout à fait conventionnel. En plus de subventions publiques classiques de l'État néerlandais, les projets de la Blender Foundation ont fait l'objet de campagnes de financement participatif, soit par la prévente des supports vidéo (DVD, téléchargement), soit plus tard par un abonnement à un service de téléchargement de contenus pédagogiques, de films et d'éléments 3D, le Blender Cloud. Le financement participatif n'est pas un phénomène nouveau, ni dans le divertissement, ni en informatique, puisqu'en particulier Blender avait pu libérer son code source* en 2002 grâce à une telle campagne. Néanmoins, il est intéressant de mettre en regard ce modèle de financement avec celui de la diffusion des films, puisque, tout comme les logiciels libres, non seulement les individus ayant participé au financement du film peuvent se l'approprier, mais tous les autres aussi peuvent regarder, télécharger, modifier, redistribuer le film en vertu des termes de la licence Creative Commons Attribution.

Une autre source de financement à mentionner est celle qui a eu lieu pour le film *Sintel*, qui a consisté à *crowdsourcer* une partie du travail de modélisation d'assets*. Les créateurs du film ont demandé aux utilisateurs de Blender, via les canaux d'information de la communauté, de les aider à modéliser des éléments de décor, qu'ils ont utilisés dans le film. De cette manière, les coûts sont réduits puisqu'une partie des graphistes est amatrice et bénévole. Il est douteux que ce procédé soit généralisable, à moins de fédérer une forte communauté autour d'un studio ou d'un projet.

À l'heure actuelle, ces modèles de financement, s'ils sont très adaptés aux films de la Blender Foundation en tant que développeur de logiciel, rencontrent des obstacles dans le cadre de studios d'animations. Comme le souligne Flavio Perez, la Blender Foundation elle-même est en train de se restructurer afin de développer son activité de production

de films sous une nouvelle entité, le Blender Animation Studio, et rencontre ce même obstacle pour produire son premier long-métrage [27] sous licence de libre diffusion :

> « Il y a des initiatives. Mais là par exemple, même la Blender Foundation a renoncé [à la diffusion sous licence de libre diffusion]. Ils savent très bien que leur long-métrage ne pourra pas être libre comme ils le souhaitent. Déjà parce qu'ils achètent la propriété intellectuelle d'un personnage à quelqu'un [le personnage de bande dessinée, l'agent 327], et ensuite parce que Ton Roosendaal [président de la fondation] l'a très bien dit, si on rend le film libre dès le premier jour, personne ne va vouloir nous l'acheter pour le sortir en salle. Puisqu'il sera librement disponible sur Internet, librement redistribuable, personne ne va vouloir nous acheter les droits DVD. Personne ne va vouloir nous acheter les droits d'exploitation en salle. Même Netflix, ça ne l'intéressera pas, puisqu'ils n'ont pas l'exclusivité. […] que faire avec le contenu ? Si même la Blender Foundation n'a pas encore répondu à ces questions… je crois qu'on ne peut pas y répondre, nous, directement. [28] »

En général, dans un studio d'animation les droits d'exploitation des œuvres commanditées par les clients se négocient, et peuvent appartenir soit aux auteurs, soit aux clients. Cela implique qu'une publication sous licence libre doit être préalable à la création. Cette décision peut avoir des conséquences importantes sur la vente du contenu, et n'est pas nécessairement bien vue par le client.

Cette expérience n'a pas eu lieu au sein des Fées spéciales, car la plupart des travaux réalisés étaient en prestation pour un client. Pour pouvoir proposer une telle solution, il faudrait soit que le client soit déjà favorable à ce type de publication, soit que nous puissions le convaincre de son bien-fondé, notamment économique, et que des contraintes externes ne l'empêchent pas de prendre cette décision (producteurs et diffuseurs, pouvoir publics dans le cas d'un appel d'offre, etc.).

Il serait sans doute plus envisageable de faire une expérience limitée dans ce domaine pour les projets produits en interne par la structure, mais là encore la décision

27. Ce film, intitulé *Agent 327*, est adapté d'une bande dessinée néerlandaise de Martin Lodewijk. Un court-métrage faisant office de *teaser* et produit par la Blender Foundation est déjà sorti en 2017, et en 2019 le long-métrage est en phase de financement.

28. Entretien semi-directif avec Flavio Perez, 6 février 2019.

dépendrait de la volonté des financeurs et diffuseurs. Ce n'est pas encore la voie qu'a choisie l'entreprise, qui s'est jusque-là concentrée sur l'élaboration de pipelines* libres et la contribution au libre, mais c'est une piste d'exploration pour les nouvelles formes de production et d'exploitation.

Conclusion

Comme nous l'avons vu en début de ce chapitre, le but du logiciel libre est de garantir à ses utilisateurs des libertés lui permettant de s'affranchir de l'éditeur. En lui permettant d'utiliser le programme comme bon lui semble ; d'apprendre comment il fonctionne en l'étudiant ; de le modifier pour le rendre plus adéquat à ses propres besoins et de le garder à jour ; et enfin de redistribuer ses modifications pour contribuer à son essor au sein d'une communauté, le logiciel libre change complètement la manière de consommer le logiciel.

Le logiciel libre n'a pas pour but de fidéliser des clients, mais de rendre plus autonomes des utilisateurs, ou simplement de créer un logiciel très performant dans le cas de l'open source. Il permet donc souvent de résoudre des problèmes de compatibilité, d'interopérabilité, et de pérennité des données.

En contrepartie de ces avantages, l'utilisation du logiciel libre demande aux utilisateurs de s'habituer à sa culture, particulièrement pour les plus avancés, qui seraient amenés à contribuer du code. Pour ceux-là, il est presque nécessaire d'avoir des solides notions de propriété intellectuelle pour comprendre le principe de fonctionnement des licences logicielles, les engagements pris vis-à-vis des auteurs et des autres utilisateurs, et les droits que l'auteur renonce, en contribuant, à faire valoir sur son œuvre.

Chapitre II.2

La place du libre dans l'économie des studios d'animation

Introduction

Pour les studios, comme pour toutes les entreprises fondant leur production sur le numérique, le logiciel est fondamental dans la stratégie économique. Acquérir des logiciels commerciaux, développer ses propres solutions internes au sein d'une équipe R&D*, ou un mélange de ces stratégies, sont les solutions que peut choisir une entreprise pour s'équiper en logiciels.

Ces différentes stratégies peuvent être appliquées au logiciel propriétaire comme au logiciel libre. Ce dernier a développé des modèles économiques alternatifs, et souvent méconnus. Une des discussions les plus fréquentes autour des logiciels libres et open source concerne la manière de financer leur développement, de les rendre rentables pour les sociétés qui les développent, comme pour celles qui les utilisent, et par extension comment les rendre compétitifs avec leurs équivalents propriétaires.

Je montrerai dans ce chapitre les différentes stratégies économiques propres au logiciel libre pour les studios, autant dans l'acquisition de logiciels existants et dans l'achat de services de développement, que dans le développement interne ou collaboratif d'applications, de bibliothèques et de scripts.

II.2.1 Acquisition de logiciels existants

II.2.1.1 Le logiciel commercial propriétaire, solution par défaut

En 2019, une grande majorité des studios préfère utiliser les logiciels de création 3D* du commerce plutôt que de développer en interne des logiciels de création. Il y a plusieurs raisons à ce choix. D'abord, ces logiciels sont très performants, et ont fait leurs preuves, selon le programme, sur des centaines ou milliers de films. Comparés au développement interne, ils sont très économiques. Créer un logiciel aussi complexe et spécialisé qu'une suite de création 3D demande des milliers d'heures à des ingénieurs hautement qualifiés, et c'est un investissement long à rentabiliser, particulièrement dans un contexte très concurrentiel. Un support et une maintenance doivent être constamment fournies pour corriger les bugs qui pourraient survenir en cours de production. De nouvelles fonctionnalités doivent aussi être régulièrement développées, et une équipe R&D* doit donc travailler à plein temps. Cette possibilité n'est a priori accessible qu'aux plus grosses structures.

La deuxième raison de préférer les logiciels du commerce est que les employés, c'est-à-dire les graphistes, sont dans la majeure partie des cas formés dans des écoles. Ces dernières échangent avec les professionnels afin de former les étudiants aux logiciels les plus utilisés. Inversement, compte tenu du temps et donc du coût de formation des employés, les studios préfèrent souvent utiliser des logiciels répandus. Dans une étude de l'observatoire des métiers de l'audiovisuel parue en 2018 [67], une section est consacrée aux compétences logicielles « en tension », c'est-à-dire recherchées par les entreprises, et pour lesquelles il est difficile de pourvoir les postes. Les quatre premiers logiciels sont des programmes d'animation 2D ou 3D propriétaires, le cinquième est Blender, représentant 8 % des compétences recherchées. Une autre catégorie dans la même étude répertorie les compétences « valorisées » par les entreprises et elle est également tenue par les logiciels Autodesk et Adobe. Cet état de fait crée un cycle difficile à rompre entre les formations et les entreprises.

Cependant, on peut modérer cette affirmation, car les techniques de l'animation

évoluent très vite, et on observe des phénomènes de mode, certains logiciels étant adoptés en quelques années, et d'autres complètement oubliés en très peu de temps. Par exemple, en 2019, Houdini [1] et Substance Painter [2], commencent à être bien implantés dans les studios et les écoles, même s'ils ne sont pas cités dans l'étude. À l'inverse, un logiciel comme Softimage|XSI, qui était très utilisé pendant plusieurs années, a cessé de l'être presque du jour au lendemain, lorsque son éditeur a arrêté de le commercialiser (cf. section II.1.2).

Un exemple concret de coût

Afin de déterminer l'impact direct du coût des licences pour un studio, nous allons prendre un cas réel de prévisionnel d'investissement pour le film *Dilili à Paris*. Le choix d'utiliser des logiciels libres a été fait avant le début du projet. On compte dans le nombre de licences le nombre maximum de graphistes travaillant en même temps sur le projet, dans les différents départements. Si nous avions utilisé des logiciels propriétaires, il se serait agi, soit d'un outil de dessin et de peinture (Photoshop), soit d'un logiciel de création 3D* (Maya), soit des deux en même temps, ainsi que d'un logiciel de suivi de fabrication (Shotgun).

Logiciel	Type de licence	Nombre	Coût par licence	Total sur 18 mois
Maya	mensuelle	15	175	47250
Photoshop	mensuelle	10	30	5400
Shotgun	mensuelle	15	30	8100
Windows	perpétuelle	15	280	4200
Total				64950

TABLE II.2.1 – Estimation du coût des licences pour le projet *Dilili*.

En comparaison, le prix *minimum* à payer pour un nombre illimité de licences pour Blender, Krita et Linux* est rapide à calculer : 0 €. Un regard non habitué à considérer des questions de gérance et développement technique pourrait voir là une grande écono-

1. Houdini est un logiciel de création 3D* édité par SideFX. Au départ destiné aux effets spéciaux, il est de plus en plus utilisé pour effectuer d'autres étapes du pipeline, de la modélisation au rendu.

2. Substance Painter [54] est un logiciel de texturing permettant une approche procédurale et largement non-destructive de la peinture de surface.

mie, qui selon le budget du projet, pourrait rendre un projet immédiatement rentable. Pourtant, il y a d'autres facteurs économiques à considérer dans le choix des logiciels libres.

II.2.1.2 Le logiciel libre : l'alternative

En effet, les modèles économiques du logiciel libre ne sont pas intuitifs. La première considération souvent abordée quand on parle de logiciel libre est son prix, qui peut prendre la forme d'un achat initial ou d'un abonnement pour une période donnée. De nombreux groupes ou individus peuvent être préoccupés par le coût des programmes : les individus qui ne peuvent s'offrir de licences légitimes des programmes commerciaux ; certaines formations, pour lesquelles les tarifs pratiqués par les éditeurs ne sont pas toujours en adéquation avec leur budget de fonctionnement ; les étudiants, auxquels les éditeurs proposent cependant souvent des licences éducatives gratuites. Mais la question intéresse aussi les studios, pour lesquels la gratuité des licences peut sembler constituer une économie substantielle.

Logiciel libre, mais pas forcément gratuit

L'idée communément admise selon laquelle le logiciel libre est nécessairement gratuit dénote une incompréhension. Le calcul précédent, arrivant à une somme de 0 € pour les licences, est trompeur à plusieurs titres.

D'abord, ce calcul part du problème terminologique associé au logiciel libre, en anglais *Free Software*, que beaucoup comprennent comme signifiant « gratuit » plutôt que « libre ». Cette mauvaise interprétation s'est largement transmise en français, où on entend fréquemment l'expression « logiciel libre et gratuit ». En pratique, les licences libres accordent aux utilisateurs le droit de redistribuer gratuitement ou commercialement le code. Il est donc difficile pour les éditeurs de logiciels libres d'obtenir un monopole commercial sur leurs programmes. En revanche, rien ne les oblige à donner leur code.

D'autre part, l'équation « je ne suis pas obligé de payer » = « je ne dois rien à personne » est simpliste. Si elle est strictement vraie légalement, y compris pour un logiciel

téléchargeable gratuitement, elle ne rend pas compte du fait que le développement des logiciels est en partie financé par les dons, et que les utilisateurs en profitent directement. Les entreprises ont donc tout intérêt à donner de l'argent aux développeurs afin de leur permettre de corriger et d'améliorer les programmes qu'elles utilisent, et elles sont les mieux placées pour le faire. Une question légitime pourrait être : combien ? Il n'y a pas de consensus en réponse à cette question, mais on pourrait dire en première approche que dès que l'entreprise en a les moyens, elle devrait approcher le prix qu'elle aurait consacré à des logiciels commerciaux équivalents.

Flexibilité du nombre de licences

Outre ce coût initial, un facteur important à considérer par une production est l'estimation du nombre de licences. En effet, c'est un calcul qui fait partie du budget d'un projet, et éventuellement des frais de fonctionnement d'une société. Le nombre de postes peut beaucoup varier d'une production à l'autre, et même d'un moment à l'autre d'une production. Ainsi, lors du passage de la préproduction à la production, il peut être nécessaire d'engager beaucoup de graphistes d'un coup [3]. Le nombre de postes augmente proportionnellement, ce qui signifie que les licences doivent être commandées en conséquence.

Parfois, un imprévu demande d'engager de la main-d'œuvre qui n'était même pas prévue dans les plannings et budgets prévisionnels, ce qui a également un impact puisqu'il faut prévoir suffisamment de licences pour que chacun puisse travailler. Bien sûr, les licences ne sont pas le seul poste de dépense affecté, car il faut également prévoir les machines, les bureaux, et même simplement les locaux en conséquence. Mais c'est tout de même un facteur important pour la souplesse de production.

Le fait d'utiliser des logiciels libres réduit ce problème à néant, car même si le logiciel était payant, c'est-à-dire s'il fallait payer pour en obtenir une copie, une licence libre confère à la société le droit d'installer le programme sur un nombre illimité de postes de

3. En France, ces graphistes sont généralement intermittents, ce qui encourage la souplesse de ce type de projet.

travail, éternellement, et de le redistribuer comme bon lui semble (cf. section II.1.1.1).

II.2.2 Développement interne des logiciels

II.2.2.1 Contrôle de la chaîne de fabrication

À l'opposé de l'acquisition de logiciels existants, le choix du modèle de développement interne offre un contrôle total à la structure puisqu'elle est non seulement maîtresse des droits d'exploitation du code, mais qu'elle peut l'étendre autant que nécessaire pour répondre aux nouvelles contraintes apportées par un projet ou par le marché. En tant que secret industriel, ce modèle pouvait en outre offrir un avantage concurrentiel considérable à une époque où les studios et les éditeurs de logiciels étaient moins nombreux. Cet avantage se réduit concernant les programmes de fabrication, mais il existe encore dans certains domaines précis.

Comme l'écrit François Elie dans le contexte du logiciel libre [3] :

> « On réduit souvent l'économie du logiciel au domaine de l'édition logicielle. Pour un éditeur, les choses sont très simples : il faut investir, payer des gens pour programmer et se refaire en vendant le plus possible de licences de son logiciel. Le retour sur investissement peut être colossal (ou nul), et les marges très élevées.
>
> « Or, il s'écrit beaucoup de logiciels en dehors des circuits de l'édition. Au sein des entreprises, il peut y avoir des informaticiens qui produisent du logiciel et les entreprises peuvent recourir à du service, qui peut donner lieu à de l'écriture de logiciels. Il faut donc intégrer cet immense pan d'activité lorsqu'on parle d'économie du logiciel. C'est d'autant plus important que ce « chacun pour soi » est un gaspillage de ressources, car il conduit à refaire tous un peu la même chose. »

L'observation d'Elie est valable dans le domaine de l'animation, car même si beaucoup de studios utilisent des logiciels du commerce, la pratique du développement interne existe toujours, en particulier dans les gros studios, où les équipes R&D* sont indispensables au bon fonctionnement des productions. Ceci dit, en ce qui concerne les logiciels de création, seules quelques sociétés en France dépendent entièrement d'outils internes.

On peut par exemple citer la société BUF [17].

En s'industrialisant progressivement, beaucoup de studios élaborent leurs outils de gestion d'assets, de suivi de production, et plus généralement de pipeline*. Ils choisissent parfois des outils du commerce comme Shotgun, mais beaucoup développent leurs propres outils, pour plusieurs raisons. Ils peuvent vouloir fonctionner sans dépendre d'éditeurs logiciels (cf. section II.1.2). Les conditions d'accès aux logiciels propriétaires peuvent être trop restrictives, par exemple dans le cas d'un parc informatique très grand ou très variable, demandant de nombreuses licences ou un nombre fluctuant de licences, ce qui entraîne dans un cas une facture conséquente, et dans l'autre une gestion complexe (cf. section II.1.1.1). Enfin, ils peuvent souhaiter avoir une totale maîtrise de leurs processus, ce qui n'est pas toujours possible avec des logiciels propriétaires impossibles à modifier (cf. section II.1.1.3).

II.2.2.2 Fonctionnalités développées par la communauté

Un autre facteur économique à considérer pour un studio est le coût de développement d'outils internes, en plus des applications principales. Ces outils peuvent être des scripts* et add-ons*, comme l'exemple du rig de pantins exposé à la section III.3.1. Ce type de développement dédié à un projet est presque toujours nécessaire dans les studios, et les programmes, commerciaux comme libres, permettent ces extensions grâce à leurs interfaces de programmation. Cependant, le cas du logiciel libre est différent, dans la mesure où les développements internes peuvent être dans certains cas intégrés au logiciel principal par la suite.

Les développeurs principaux d'un projet de logiciel libre n'ont pas forcément les mêmes priorités que les studios, et ne proposent pas toujours rapidement des fonctionnalités qu'on pourrait qualifier de « standards ». Par exemple, la bibliothèque* de cache* de géométrie Alembic (cf. section II.3.2.1) a mis plusieurs années avant d'être incluse dans Blender, même si elle était considérée comme incontournable par une grande partie de l'indutrie, et même si les autres logiciels de 3D l'avaient pour la plupart déjà intégrée. Mais son intégration ne figurait pas dans la feuille de route de la fondation Blender, qui

préférait affecter ses ressources à d'autres projets. Le studio français Dwarf Animation a créé un patch* permettant d'utiliser Alembic, et l'a proposé aux développeurs de Blender [4]. Afin de répondre à la demande de nombreux utilisateurs, et puisqu'une grande partie du travail de design et de développement était déjà faite, ces derniers ont terminé l'intégration du code, et Alembic a été disponible dans les versions suivantes.

Un exemple très similaire est celui de Cryptomatte, un système permettant de décomposer une image en différentes couches, selon des identifiants d'objets et de matériaux. Cette bibliothèque a été d'abord intégrée à Blender par le studio canadien Tangent Animation pour les besoins d'un long-métrage, puis proposée, corrigée d'après les critiques des développeurs, et intégrée dans Blender.

On voit que le fait de pouvoir étendre le cœur de Blender est acquis avec l'obtention du programme, et ces extensions peuvent soit rester internes à un studio, soit servir à toute la communauté. De fait, les sociétés peuvent aussi utiliser leurs ressources internes pour leur propre intérêt, mais aussi celui de la communauté, et profiter en retour de développements d'autres sociétés.

De même, pour une société indépendante développant des programmes pour son propre usage, la publication en open source peut être intéressante. Comme le déclare le créateur du logiciel Storyboarder (cf. section III.2.1), Charles Forman, sur le site web du logiciel, Storyboarder a été créé au sein d'une société de production de film, qui « gagne de l'argent en faisant des films, et non pas des logiciels » [45]. Il ajoute que le logiciel répond à un besoin de niche, et que pour que le logiciel soit profitable, il faudrait le vendre pour un coût exorbitant. La décision de le publier sous licence libre est donc logique, puisque la société ne compte pas gagner d'argent avec le logiciel, mais peut espérer profiter d'un développement communautaire pour l'améliorer.

La mutualisation des coûts de développement est donc intéressante économiquement, pour peu qu'assez d'acteurs travaillent sur un logiciel, c'est-à-dire pour peu que la communauté soit suffisamment établie, et le logiciel adopté.

4. On peut lire les échanges entre le studio et les développeurs sur la forge de Blender, à l'adresse suivante : `https://developer.blender.org/D1783`

II.2.3 Financement du développement logiciel

Le développement informatique demande des ressources : du temps et de l'énergie. Ce lieu commun est utile à rappeler dans le contexte du logiciel libre, car il pose des questions fondamentales sur la manière dont les logiciels sont développés. On parle parfois du logiciel libre et open source avant tout comme d'un passe-temps, une passion qu'ont les développeurs pour la découverte, le partage, ou simplement la joie de la création, comme par exemple Élie [3, § Le modèle communautaire d'individus]. Beaucoup consacrent du temps à cette passion, sans attendre de compensation financière. Comme l'expliquent Jullien et Zimmermann [13], ce modèle n'est viable que pour des individus développant des petits programmes en dilettante, sans commercialisation ni support, et sans attente de la part des « clients ». Dès lors qu'un projet prend assez d'ampleur, les ressources nécessaires à son développement dépassent les capacités d'un seul programmeur du dimanche. Il faut alors y consacrer plus de temps, soit en attirant d'autres développeurs bénévoles structurés autour d'une communauté, soit en embauchant des développeurs professionnels, c'est-à-dire rémunérés pour leur travail. Pour cela, il est nécessaire de trouver un financement.

Plusieurs modèles de financement open source ont émergé, qui sont souvent complémentaires. Ils sont étudiés du point de vue du logiciel lui-même, c'est-à-dire les moyens employés par la communauté pour développer le logiciel. Nous nous concentrerons plus particulièrement ici sur l'action des studios dans ce développement.

Salariat

Beaucoup de développeurs open source sont salariés par une entreprise ou une association et développent leur logiciel dans ce cadre. Le groupe, à son tour, peut avoir différents modèles économiques. Certaines entreprises de développement logiciel vendent du service pour des logiciels libres (support, maintenance), et le développement du logiciel est alors au cœur du modèle économique, puisque les anomalies corrigées pour les clients dans le cadre d'un contrat de support sont reportées dans le code public.

Pour d'autres structures, le développement logiciel est secondaire. Ces sociétés ont une autre activité principale, et investissent dans leur R&D* pour développer des logiciels qui peuvent leur être utiles indirectement. C'est souvent le cas des studios d'animation, dont l'activité principale est la production et la fabrication de contenu audiovisuel, mais qui s'appuient pour cela sur l'informatique. Ils doivent donc allouer des ressources au développement et à la maintenance de leurs outils, soit dans le cadre d'un département R&D à part entière, soit informellement, par les graphistes eux-mêmes ou les TD*. Pour les studios qui utilisent des logiciels libres et souhaitent y contribuer, c'est-à-dire rendre publics leurs outils, proposer des correctifs ou des améliorations à l'inclusion dans le programme, la participation à des projets peut augmenter le coût de développement, car les exigences d'un logiciel public peuvent être plus élevées que celles d'un outil ad hoc développé en interne pour une production. De plus, les interactions nécessaires avec la communauté peuvent prendre du temps aux développeurs. Néanmoins, si un outil a été développé une fois pour une production, s'il est susceptible d'intéresser d'autres utilisateurs, et si son développement est pensé dès le départ pour une publication, le surcoût entraîné peut être limité.

Dons

Un autre modèle de financement des programmes open source est le don des utilisateurs ou d'entreprises. Ces projets fonctionnent alors grâce à un système de mécénat [5]. Certains projets ont une approche hybride, comme la Blender Foundation, qui salarie des développeurs avec des dons des utilisateurs, ainsi que des bourses conséquentes d'entreprises du secteur de l'animation ou du jeu vidéo [6]. Il est particulièrement important pour les studios d'animation de considérer les partenariats qu'ils peuvent établir avec les projets des logiciels qu'ils utilisent, selon leur stratégie de développement. On peut considérer qu'une bourse ressemble à une externalisation du développement, dans la me-

5. C'est le cas par exemple de Krita et de Godot, qui proposent en option aux utilisateurs de donner de l'argent pour supporter le développement, soit ponctuellement, soit régulièrement.

6. Ainsi, des entreprises comme Ubisoft et Epic Games ont annoncé en 2019 l'octroi de bourses de développement conséquentes sur plusieurs années. Ces bourses sont gérées par la Blender Foundation, qui conserve la maîtrise d'œuvre sur le développement du programme.

sure où le travail effectué par les développeurs du logiciel sera utile au studio. Il est vrai que dans les exemples cités, les entreprises donatrices n'ont pas officiellement de contrôle sur les décisions des développeurs. Cependant, dans le cas d'Epic, il était entendu que la bourse octroyée à Blender devait conduire à des améliorations dans un domaine précis [7].

Subventions

Les subventions publiques constituent une autre source de financement pour les entreprises. Dans l'animation, il existe des aides régionales à la production, pour la fabrication des films. Une partie de cette subvention peut être imputée au développement informatique pour un projet précis. Mais il existe aussi des aides spécifiquement dédiées au développement d'outils dans le cadre d'une opération de R&D. Ainsi le CNC* peut accorder aux entreprises du secteur un *soutien financier aux industries techniques*. Cette aide concerne les entreprises présentes sur le territoire, et peut soutenir plusieurs catégories de travaux, parmi lesquels le développement et l'innovation. Dans cette catégorie, l'*aide à un nouveau produit ou nouveau service* concerne spécifiquement les outils « à destination commerciale ou qui [permettent] d'engendrer une augmentation chiffrable de [la] productivité » [62]. Dans le cadre de cette aide, le taux de soutien (pourcentage des dépenses pour un projet) est plus élevé pour les logiciels « diffusés gratuitement ou en logiciel libre » que pour les autres (60 % contre 45 %).

Développement externe pour les studios

Un modèle économique souvent adopté par les développeurs de logiciels libres est celui de la prestation de service : le logiciel est libre, et son développeur ne le commercialise pas. En revanche, il monnaie du service annexe au logiciel. Il peut développer des fonctionnalités pour le compte d'une entreprise, fonctionnalités qui seront ensuite rendues disponibles pour la communauté. Il peut s'agir de support technique, de formation, de maintenance. Un exemple notable est le système d'exploitation GNU/Linux*, dont

7. L'annonce officielle de la bourse stipule : « L'Epic MegaGrant sera versée progressivement durant les trois prochaines années, et contribuera à l'initiative de développement pour professionnaliser Blender. » (Traduction de l'auteur)

il existe de nombreuses variantes appelées distributions, et dont certaines sont commerciales. La distribution Red Hat, répandue dans les décennies 1990 et 2000 a exploité ce modèle en vendant du service aux entreprises.

Ce modèle a été également adopté par des développeurs de logiciel libre dans le cinéma d'animation, et plusieurs exemples existent en France. Ainsi, Frank Rousseau, auteur du logiciel de gestion de production Kitsu [53] au sein de sa société CGWire, est financé depuis 2017 par plusieurs studios pour développer son logiciel. Il le développe en permanence, en lien avec un ou plusieurs studios à la fois, tout en faisant du support et de la maintenance [36]. Le fruit de ces développements prioritaires est reversé dans le code source* public, et les studios bénéficient des améliorations conçues avec ou pour d'autres clients.

Une autre manière de mettre en commun le développement entre studios se produit lorsqu'un même outil est utilisé et développé en interne par plusieurs studios. Par exemple, le framework* Kabaret (cf. section III.5.1.2) de Damien Coureau, a d'abord été développé pour le studio Supamonks, et publié ensuite sous licence libre. Il a été notamment adopté et adapté par Flavio Perez, directeur technique sur le pipeline* des Fées spéciales. Le principe de ce programme, en tant que framework, est d'offrir un environnement de développement permettant d'écrire sa propre solution. Il faut donc nécessairement du développement pour construire un pipeline autour, mais la solution est conçue pour faciliter ce développement en offrant une architecture et des commandes logiques et facile à étendre. Flavio avait testé une première version plusieurs années auparavant, mais n'en avait utilisé que quelques fonctions, estimant que le framework n'était pas assez mûr. Au moment de définir le pipeline des Fées, Flavio a contacté Damien pour lui demander la dernière version de Kabaret, qui était en bêta privée à cette époque. En l'utilisant pour écrire son pipeline, il a alors engagé des discussions avec le développeur afin de l'améliorer, et le rendre plus facile à adapter à des cas non prévus. Non seulement ce retour d'expérience a été précieux pour continuer le développement, mais les Fées spéciales ont également fourni directement le code de ses améliorations au studio Supamonks.

Ainsi, le développeur du studio original, en publiant son code source*, bénéficie des améliorations d'autres studios. Cela demande du temps pour former ses interlocuteurs, et fédérer une communauté de développeurs, mais in fine, d'autres personnes développent « à sa place » (cf. section II.3.3.4).

Conclusion : le coût global des logiciels

Beaucoup d'utilisateurs parmi le public et au sein des studios ont intériorisé l'équivalence *logiciel libre = gratuit*, y compris au sein des Fées spéciales. Il n'est pas rare d'entendre prononcer en réponse la phrase « Le logiciel libre, c'est trop cher » par les membres de l'équipe technique, lors de réunions de projets visant à établir le pipeline* de fabrication.

Cette formule n'est pas toujours vraie, mais elle est utile à considérer dans certains contextes, particulièrement en entreprise où les budgets imposent la rentabilité économique. Ce qui est entendu par là est que, pour exécuter une tâche précise, il est parfois économiquement plus intéressant à court terme d'utiliser un outil propriétaire. Cela dépend de la récurrence de la tâche, de l'existence et de la performance d'un outil libre conçu exprès, et du coût de développement d'un outil spécifique. En d'autres termes, il n'est pas toujours réaliste ou possible de faire l'investissement nécessaire pour développer ou faire développer un outil. De même, la formation à des outils nouveaux a un coût et prend du temps, et les entreprises n'en ont pas forcément les ressources en cours de production.

Chapitre II.3

Usages du libre : considérations techniques

Introduction

Le logiciel libre a des particularités économiques et juridiques importantes pour les studios d'animation, des contraintes et des possibilités particulières. De la même façon, le logiciel libre peut offrir aux studios des façons différentes de travailler, et de développer leurs outils et leurs projets.

Nous verrons dans ce chapitre les éléments concrets qui peuvent concerner les studios d'animation lors de l'utilisation du logiciel libre : le rôle de l'innovation dans le développement logiciel, et de quelle manière elle peut pousser à l'adoption du logiciel libre ; l'importance critique des formats d'échange pour l'animation ; et les procédures types permettant de contribuer à l'élaboration des logiciels avec leurs communautés.

II.3.1 Innovation des logiciels de création

L'innovation est au cœur du développement informatique et des industries qui utilisent des outils numériques. Les éditeurs de logiciels se livrent donc généralement une compétition pour l'innovation de haute technologie. Cette innovation a lieu à plusieurs

niveaux. Il s'agit soit d'intégrer des pratiques existantes [1] ; soit d'intégrer des algorithmes récents, pour le rendu ou le traitement de signal ; soit de créer des systèmes complètement originaux pour changer la manière de produire les images.

Dans cette compétition pour l'innovation, certains logiciels libres ne font pas exception. Ils se démarquent cependant des logiciels propriétaires à plusieurs titres. D'abord, même si les développeurs prennent de grands soins pour ne pas casser la rétro-compatibilité avec les versions précédentes, ils ont moins de difficultés à revoir complètement l'interface ou le fonctionnement. Par exemple, le logiciel Blender a publié deux versions majeures [2] cassant dans une grande mesure la compatibilité avec les versions précédentes. À deux reprises en moins de dix ans, l'interface du programme a été modifiée en profondeur, ainsi qu'un grand nombre de systèmes (moteurs de rendu, graphe de dépendance, API* de scripting). Les fichiers restent globalement compatibles, mais certaines données et fonctions sont perdues. Cette politique de rupture dans les versions majeures permet d'abandonner des parties anciennes du programme, et d'ajouter des nouvelles fonctions, qui peuvent être l'occasion d'une plus grande innovation.

Parmi les nouvelles fonctions de la version 2.80, deux exemples sont particulièrement intéressants, car ils illustrent chacun dans son domaine le caractère innovant du programme, et les particularités par rapport à des modèles de développement plus conventionnels. Le premier est Eevee, un moteur de rendu (cf. section III.3.2) utilisant les technologies de rendu dit « temps réel », c'est-à-dire développées initialement pour les applications temps réel comme les jeux vidéo. Le deuxième est le *Grease Pencil*, un outil de dessin et d'animation dans l'espace 3D, existant déjà depuis 2008 et permettant d'animer depuis 2015, mais ayant fait l'objet d'une profonde rénovation pour la version 2.80.

1. Par exemple, la réalité virtuelle, qui récemment a explosé auprès du grand public et que les éditeurs de logiciels de création 3D* commencent à intégrer pour profiter de nouveaux paradigmes de création ; ou bien le rendu physique réaliste (Physically-Based Rendering, PBR), apparu au début des années 2000 et rapidement adopté par tous les éditeurs.

2. Il s'agit des versions 2.50, sortie en 2010 et 2.80, sortie en 2019.

Eevee

Eevee est un moteur de rendu intégré à Blender et développé spécifiquement pour lui. Il a été en partie conçu par le développeur Clément Foucault. Vers 2016, ce dernier a créé un fork* de Blender et travaillé indépendamment pour améliorer le rendu interactif dans la vue 3D (*viewport*). Ses vidéos de démonstration et les versions qu'il a publiées ont suscité l'enthousiasme de la communauté et des développeurs, si bien qu'au moment où la Blender Foundation a commencé à développer la nouvelle version majeure (2.80), il a été engagé pour continuer son travail au sein de la fondation. Au lieu d'améliorer simplement la prévisualisation dans le viewport, le développement a été étendu à un moteur de rendu complet, destiné à remplacer le moteur vieillissant intégré à Blender, le Blender Internal. Cela offre l'avantage de permettre à la fois une prévisualisation interactive réaliste, plus proche du rendu définitif, et des rendus en temps différé moins réalistes qu'un *path tracer* comme Cycles, mais beaucoup plus rapides. Le développement de ce nouveau moteur n'a donc été possible qu'à cause de l'abandon de l'ancien moteur lors du passage à la nouvelle version.

Le Grease Pencil

Le Grease Pencil est un outil de dessin dans Blender, permettant de faire du dessin animé directement dans la vue 3D. Il tire son nom d'un outil de dessin traditionnel, utilisé notamment pour annoter les images sur pellicule. Développé initialement en 2008 pour la version 2.47 de Blender, en tant qu'outil d'annotation, il a peu à peu évolué en outil complet d'animation, en particulier sous l'influence de l'animateur espagnol Daniel Martinez Lara. Ce dernier n'est pas développeur, mais c'est un animateur et formateur qui dirige une école d'animation et il est très impliqué dans le développement de Blender en tant qu'expert. Entre 2017 et 2018, l'outil a été développé par le développeur Matias Mendiola en collaboration avec Martinez Lara au sein de l'école de ce dernier, dans le cadre de la production d'un court métrage utilisant l'outil, *Hero*. Il a par la suite été réintégré dans le code de Blender à l'occasion de la nouvelle version.

L'outil actuel dans Blender suscite un fort engouement du fait de la potentialité

pour des méthodes de fabrication hybrides entre le dessin animé « traditionnel » et l'animation 3D. Mais il ne faut pas oublier que cette version est l'aboutissement d'un long développement itératif, et que l'outil n'avait pas du tout été pensé pour cette application initialement.

Dans le cas de cet outil, l'innovation provient d'une combinaison de facteurs. D'abord la création d'un outil limité mais prometteur dans sa première version pour l'annotation. Ensuite, l'intuition d'un utilisateur de la capacité de l'outil à être détourné et exploité dans un but beaucoup plus intéressant, et sa volonté à voir aboutir cette intuition. Puis, la capacité qu'il a eue de faire lui-même le travail en engageant un développeur pour mettre en œuvre ses idées, avec l'aval des développeurs de Blender. Cette capacité est due à la nature open source du programme, qui lui a permis de développer un fork*, puis de réintégrer ce fork dans le code du programme.

L'innovation dans le logiciel libre n'est donc fondamentalement pas différente des autres modèles de développement, mais la prérogative n'en est pas nécessairement à l'éditeur, et de bonnes idées peuvent donc surgir de la communauté, et se voir à terme intégrées au logiciel.

II.3.2 Interopérabilité et standardisation

On l'a vu, le pipeline* d'un film d'animation se compose de nombreuses étapes interdépendantes. Chaque étape peut être réalisée par un logiciel différent, et plusieurs logiciels sont parfois utilisés pour une même étape. Pour que ces logiciels soient compatibles, il faut que leurs créateurs tombent d'accord sur des formats et composants communs.

II.3.2.1 Formats d'échange

Certains formats de fichiers* sont conçus pour représenter les données d'un programme donné. Ils peuvent suivre de près la structure de données et la logique internes du programme. À l'inverse, les formats dits d'échange sont spécifiquement conçus pour

transmettre les données d'un programme à l'autre selon une spécification plus générique, c'est-à-dire moins intimement liés à un programme précis [3].

Certains formats d'échange vont plus loin, en étant maintenus par des organisations de normalisation, ce qui incite les développeurs à respecter un standard [4], et rend donc les formats en question plus compatibles. Ce sont ces formats ouverts et standards qui sont de plus en plus adoptés par l'industrie de l'animation.

Formats propriétaires

Parmi les formats de fichiers, qu'ils soient propres à un logiciel ou des formats d'échange, beaucoup sont dits propriétaires. Cela signifie que des dispositions légales ou techniques empêchent les développeurs de les intégrer dans leurs propres programmes, et par extension, les utilisateurs de s'en servir en dehors de l'utilisation prévue. Les dispositions qui rendent propriétaire un format peuvent être le fait que sa spécification n'a pas été divulguée, ou des brevets logiciels empêchant simplement de l'utiliser.

Si un format propriétaire est utilisé sans être documenté publiquement, il est très difficile de l'utiliser dans un autre contexte que celui prévu par l'éditeur, et donc de l'inscrire dans un écosystème logiciel plus large. Cette difficulté est d'abord technique, car si les spécifications du format ne sont pas connues, et que le code source* du programme permettant de lire et écrire ce format est fermé, la seule possibilité restant aux développeurs pour comprendre le format est de faire de la rétro-ingénierie, c'est-à-dire de

3. Un exemple parlant est celui des formats d'image. On ne pense plus guère aujourd'hui, quand on envoie une image au format JPEG, qu'il n'a pas toujours existé. C'est un format standard, universel et compris par tous les ordinateurs, navigateurs, programmes de création ou d'affichage d'image. Pourtant, son cahier des charges a un jour été conçu, les algorithmes développés et implémentés, et finalement un standard a été ratifié. En respectant strictement ce standard, les différentes technologies permettant de créer et d'afficher une image JPEG garantissent à l'utilisateur que l'image sera transmise comme prévu : sans corruption, et en respectant les couleurs définies par le créateur. Cela n'évite pas tout problème de compatibilité dans la mesure où la mise en œuvre correcte du format reste toujours à la charge des développeurs. Mais contrairement aux formats propriétaires, les développeurs ont du moins accès à des documents fiables leur permettant de le faire.

4. J'emploie dans ce chapitre le terme « standard » plutôt que « norme », d'une part car c'est un mot fréquemment utilisé par les professionnels, par anglicisme, et ensuite parce que les formats de fichiers et processus informatiques propres à l'animation sont rarement normalisés par des organisations publiques, mais créés au sein de la communauté. Le cas du format JPEG est justement différent, car il s'agit d'une norme internationale.

décortiquer le fonctionnement du programme sans avoir accès aux sources. Ce processus est long, complexe et incertain. De plus, il est à la limite de la légalité dans beaucoup de juridictions. Un article de l'Electronic Frontier Foundation [31] détaille les risques de poursuite encourus aux États-Unis. Pour résumer, des précautions extrêmes doivent être prises dans la méthodologie de rétro-ingénierie pour ne pas être dans l'illégalité. En Europe, la question n'est pas complètement tranchée, mais la protection des logicielles demeure très puissante [5].

Ce problème des formats de fichiers propriétaires est récurrent en informatique. On peut considérer qu'il nuit à l'interopérabilité* et restreint les utilisateurs dans leur choix. De plus, c'est encore une dépendance des utilisateurs aux éditeurs, puisque ces derniers sont seuls en capacité à autoriser ou interdire l'utilisation du format, via la licence qu'ils concèdent. S'ils décident de ne plus supporter le format, celui-ci peut devenir obsolète très rapidement, sans recours pour les utilisateurs.

Avant d'aborder l'utilisation des formats ouverts, j'exposerai les problèmes observés avec le format propriétaire FBX, créé par Autodesk. Ce format très répandu permet de stocker des données 3D (maillages, armatures, animation, hiérarchies, etc.). Il s'agit d'un format d'échange utilisable dans une majorité des logiciels de création 3D*, puisque la plupart disposent de fonction d'import et d'export depuis et vers ce format. Cependant, la spécification du format n'est pas disponible publiquement. Autodesk livre des exporteurs et importeurs avec chacun de ses programmes de création (Maya, 3ds, etc.) ainsi qu'une bibliothèque* propriétaire permettant de lire le format, mais pas sa spécification.

Ainsi, si un développeur souhaite implémenter un importeur FBX dans son programme, il peut soit utiliser la bibliothèque d'Autodesk, mais ce n'est pas toujours possible légalement [6], soit faire de la rétro-ingénierie pour comprendre, à partir d'exemples, comment est structuré le format. C'est le cas de Blender, pour lequel les développeurs n'ont pas le droit d'utiliser la bibliothèque officielle à cause d'une incompatibilité de

5. L'article de Ciancarini et coll. [11] mentionne un jugement rendu en 2012 par la Cour européenne du justice, qui risque de faire précédent et a déclaré légale la rétro-ingénierie à des fins d'interopérabilité. Mais il estime aussi que d'autres jugements pourraient aussi bien créer une jurisprudence différente.

6. En particulier dans le cas de logiciels open source à fort copyleft (cf. section II.1.3.1)

licence. Ils ont donc dû décortiquer le format FBX pour créer leurs importeur et exporteur. Il en a résulté que leur version, quoique fonctionnelle, n'est pas aussi compatible que les versions Autodesk.

Formats ouverts

Afin de pouvoir toujours transmettre ses données d'un programme à l'autre, il est donc nécessaire de disposer de formats standards et ouverts. Comme le montrent Jullien et Zimmermann [13], les standards informatiques sont un des deux moyens de créer une interopérabilité entre logiciels, l'autre étant l'open source, avec lequel ils sont complémentaires.

Un format ouvert [7] spécifie publiquement la structure des données enregistrées dans un fichier. À partir d'une spécification ouverte, les développeurs de logiciels peuvent écrire des fonctions d'import et d'export vers ces formats. Pour être viables, ces exporteurs et importeurs doivent respecter strictement la norme, c'est-à-dire qu'un fichier écrit par un programme donné puisse être ouvert par un autre programme, et que toutes les données présentes dans le fichier soient interprétées comme prévu par la spécification, qu'il n'y ait ni perte, ni corruption de données.

L'animation est un domaine dans lequel les types de données à échanger sont nombreux, lourds et complexes [8]. Aussi, il est indispensable de disposer de formats standards pour l'animation 3D. À cause de cette complexité et de cette variété, il est difficile de concevoir un format qui soit adapté à tous les usages courants et à venir.

Je mentionnerai ici deux formats ouverts, utilisés en 2019 sur un grand nombre de productions de films d'animation 3D, et illustrant des points importants sur l'usage des formats ouverts pour les développeurs et les utilisateurs au sein des studios.

7. Certains formats de fichier ont d'abord été créés en tant que formats propriétaires avant d'être standardisés. C'est le cas par exemple du format de document PDF, créé par Adobe en 1993 et contrôlé par la même société jusqu'à sa normalisation en 2008. Depuis, la spécification du format est maintenue par l'Organisation internationale de normalisation (ISO).

8. Par exemple, les données de géométrie, de relations entre les objets et de rigging, de texture, de shading, d'animation ; de simulation de fluides, de particules, de poils, de volumes ; de lumières, de caméras, etc.

OpenEXR : spécification et implémentation

OpenEXR est un format d'image bitmap développé au départ par la société de post-production états-unienne Industrial Light & Magic (ILM). Il a été conçu pour stocker des images dans un contexte de production ou d'archivage, et non pas de diffusion [9].

Concrètement, comme beaucoup de formats de fichiers, OpenEXR est composé de deux éléments fondamentaux : une spécification précise et librement accessible du format [14], et une implémentation de référence sous la forme d'une bibliothèque open source [51]. De cette manière, un développeur voulant utiliser le format peut lier directement la bibliothèque dans son programme, et pourra la mettre à jour lorsque de nouvelles versions sortiront. Il peut aussi décider de créer ses propres routines de lecture et d'écriture du format, puisque celui-ci est documenté. De fait, il existe plusieurs bibliothèques compatibles capables de lire ou d'écrire le format, certaines plus limitées que d'autres, ou destinées à d'autres usages que la bibliothèque canonique [10].

La licence open source de la bibliothèque lui permet d'être incorporée dans des programmes libres ou propriétaires [11]. Il est cependant important de noter que certaines parties de la bibliothèque ne peuvent pas être considérées comme logiciel libre, dans la mesure où elles sont protégées par des brevets dans certains pays. Ainsi, deux algorithmes propriétaires de compression avec pertes, DWAA et DWAB, créés par le studio Dreamworks Animation, sont distribués avec OpenEXR. Ils sont très performants, mais

9. Il permet de représenter les images de manière compressée ou non, avec une précision plus grande que les formats d'image grand public et génériques tels que JPEG ou PNG. En effet, les données sont représentées sous la forme de nombres à virgule flottante d'une profondeur de 16 ou 32 bits, plutôt que des entiers. Cela signifie que des grandes valeurs peuvent être représentées avec précision, y compris des valeurs supérieures à 1 ou inférieures à 0. Le format a aussi la particularité d'être prévu pour représenter les valeurs de lumière de la scène et non celles de l'affichage, et aucune transformation n'y est donc appliquée. Plutôt que de stocker une image composée de plusieurs canaux (typiquement rouge, vert, bleu et alpha) comme la plupart des formats d'image, les fichiers OpenEXR peuvent contenir un nombre arbitraire d'images, chacune pouvant avoir plusieurs canaux. Ces différentes images constituent des passes, et ce dispositif est inestimable pour transporter efficacement les nombreuses passes nécessaires à un compositing avancé.

Sa spécification rend le format particulièrement adapté aux applications de compositing physiquement correct. Un espace colorimétrique est associé au fichier pour interpréter correctement les couleurs.

10. Par exemple, la bibliothèque tinyEXR est destinée à être légère et facile à intégrer dans un programme, en sacrifiant en échange certaines fonctionnalités avancées.

11. OpenEXR est publié sous la licence permissive BSD 3 clauses, qui est une licence permissive. Voir section II.1.3.2.

ne peuvent légalement être utilisés que dans la bibliothèque officielle. Cela signifie qu'un développeur souhaitant créer sa propre implémentation en est empêché par cette restriction [12]. Il convient donc d'être vigilant lorsqu'on utilise un format ou du code présenté comme open source, car il ne permet pas toujours toutes les utilisations.

Alembic : un pas vers l'interopérabilité 3D

Le format ouvert Alembic [13] a été initialement co-développé en 2010 par Sony Pictures Imageworks, Inc. et Industrial Light & Magic. Il a depuis été implémenté dans la plupart des logiciels de création 3D* et il est maintenu par une communauté de développeurs, dans et en dehors des studios.

Contrairement à OpenEXR, le format Alembic est défini uniquement par son implémentation de référence, et non prescrite par une spécification exacte. Une documentation existe [14], mais elle ne décrit pas le format assez précisément pour pouvoir créer une autre implémentation. Pour cela, un développeur devra nécessairement lire le code de la bibliothèque officielle, qui fait référence mais peut être amenée à changer.

Comme beaucoup de nouveaux formats — dont OpenEXR — Alembic est né d'un besoin au sein de studios. Plutôt que d'être conçu par un éditeur de logiciel, il a évolué en réponse aux nécessités concrètes de la production. Il peut à la fois permettre des pipelines* nouveaux à l'intérieur d'un studio, et faciliter la collaboration de plusieurs studios sur un même projet.

Par ailleurs, s'il a rapidement été adopté par beaucoup de logiciels au cours des années 2010, le format Alembic a aussi illustré le problème exposé à la section II.1.1.3. En effet, dans le cas de Blender, le support du format a mis longtemps a être même envisagé par les développeurs, qui le considéraient d'abord superflu pour la cible de ses utilisateurs (les indépendants et les petits studios), puis non prioritaire. Le développement croissant

12. Ce cas de figure s'est produit dans le programme FFmpeg, dont l'implémentation d'OpenEXR ne permet pas de lire ni d'écrire des images utilisant les compressions DWA, en raison du brevet.

13. Alembic est un format de cache permettant de transférer des animations de maillages, NURBS, particules, cheveux et simulations physiques d'un programme à l'autre. Il évite aux utilisateurs de se soucier des détails de hiérarchie, de rig et de skinning, ni de calcul de simulation. Son utilisation elle-même sera expliquée plus en détail dans la partie III (section III.5.2).

14. Cette documentation est dispersée sur plusieurs sites web et semble incomplète.

des relations entre les développeurs et les studios a permis un changement de mentalité, mais celui-ci a pris un temps de plusieurs années, au cours desquelles les utilisateurs pouvant faire usage des procédures permises par Alembic ont dû se replier vers d'autres solutions moins efficaces.

II.3.2.2 Organismes de standardisation

Pour que les programmes utilisés dans un pipeline soient interopérables, il faut donc qu'ils intègrent des capacités d'import et d'export dans des formats standards, et qu'elles les respectent scrupuleusement, sans quoi les fichiers seraient de fait incompatibles. Il en est de même des autres composants communs aux logiciels. Deux initiatives récentes se sont lancées afin que les éditeurs et les studios puissent mettre leurs ressources en commun pour améliorer et rendre compatibles les logiciels qu'ils développent et utilisent.

VFX Reference Platform

La première de ces initiatives est la VFX Reference Platform. Ce groupe a été mis en place en 2014 par la Visual Effects Society [15] et a pour but de définir une liste des composants (bibliothèques et formats) intégrés aux logiciels, pour améliorer leur compatibilité, particulièrement sur les systèmes Linux* dont les versions des logicielles peuvent être différentes d'une distribution à l'autre.

Cette liste, mise à jour chaque année, stipule de plus les versions à adopter. Elle recommande rarement la dernière version, son but n'étant pas d'être à la pointe, mais compatible. Même si deux logiciels de création 3D* permettent de lire et d'écrire un fichier dans un format standard, encore faut-il qu'ils s'entendent sur la version à implémenter. D'un côté, ils ont intérêt à adopter des versions récentes pour bénéficier des dernières fonctionnalités et corrections de bugs ; d'un autre côté, il peut être important de garder plus longtemps une version ancienne [16]. Cette liste est disponible publiquement

15. Une organisation internationale de professionnels du secteur des effets spéciaux. Cette organisation n'a pas de rôle de normalisation, mais elle regroupe des acteurs influents du milieu, et son opinion est assez bien suivi par les studios.

16. C'est le cas du langage de programmation Python, le plus utilisé pour le scripting dans les secteurs

sur le site de l'organisation [39].

Academy Software Foundation

L'Academy Software Foundation est un groupe établi en 2018 par l'académie états-unienne du film et par la fondation Linux. Son but est de coordonner le développement et la maintenance de bibliothèques* open source dont l'usage est crucial pour le film. Les projets logiciels qu'elle coordonne ont d'abord été conçus par des studios [17]. Compte tenu du fait que ces programmes sont devenus indispensables à tout l'écosystème de l'animation et des effets spéciaux, tous les acteurs s'accordent sur l'importance de leur développement. Afin d'éviter qu'ils ne soient abandonnés faute de moyens, et pour faciliter la contribution des développeurs de toute part, cette fondation prend à sa charge une partie de la complexité de gestion de projet, notamment l'infrastructure d'hébergement et le cadre juridique.

En plus de l'académie du film et de la fondation Linux, de nombreuses entreprises privées sont impliquées, en tant que sponsors financiers ou en tant que développeurs. Parmi ces groupes se trouvent des studios, des éditeurs de logiciels, et des entreprises de technologie généralistes. Les décisions budgétaires sont votées par les membres de la fondation, qui ont donc un rôle moteur dans le développement des projets qui y sont hébergés.

II.3.3 Contributions à un projet libre

L'activité principale d'un studio d'animation n'est généralement pas de développer des logiciels, mais de fabriquer du contenu audiovisuel, éventuellement en utilisant des logiciels. Mais une des utilisations des logiciels, dans le cadre de la R&D* est précisément de développer des processus ou du code, soit pour étendre les fonctionnalités du pro-

de l'animation et des effets spéciaux. Alors qu'une version 3 existe depuis 2008, ces secteurs utilisent en 2019 la version 2, en raison du coût global pour mettre à jour tous les scripts et programmes existants. Devant la fin du support de Python 2 annoncée par ses développeurs, la VFX Reference Platform a à son tour annoncé l'adoption de la version 3 pour 2020.

17. Parmi lesquels, OpenColorIO, OpenVDB, OpenEXR, OpenCue et OpenTimelineIO.

gramme, soit pour l'intégrer dans sa chaîne de fabrication. Les studios publient parfois une partie de leurs outils, et pour un studio utilisant le logiciel libre, il est logique de le faire sous licence libre également. En effet, des logiciels publiés sous licence libre peuvent être utilisés par toute la communauté, dans un esprit de contribution : rendre à la communauté une partie de ce qu'elle a donné sous la forme des logiciels. Ces contributions peuvent être très utiles à des projets de logiciel libre, car elles proviennent d'utilisateurs expérimentés au sein des studios, et constituent donc un retour d'expérience précieux.

Il existe de multiples façons de contribuer à un projet de logiciel. D'abord, en utilisant le logiciel et en le faisant connaître, d'autres utilisateurs peuvent être amenés à l'utiliser également, et éventuellement à y contribuer.

Ensuite, les utilisateurs de logiciels open source peuvent d'une certaine manière participer à l'assurance qualité du logiciel, en rapportant aux développeurs les bugs et problèmes rencontrés et en faisant des retours sur leur utilisation.

Les projets logiciels ont toujours besoin d'autres compétences pour améliorer, non seulement le programme lui-même, c'est-à-dire le produit distribué, mais aussi le projet plus vaste, comprenant la documentation, la traduction, le design, la communauté. Ces activités sont souvent pratiquées, soit par des volontaires, soit par des salariés d'entreprises ou d'associations organisées au sein d'une communauté de développement.

Enfin, la manière la plus directe de contribuer est de développer le programme lui-même, en écrivant du code ou en résolvant des bugs. Je me concentrerai dans cette section sur ce type de contribution, en exposant les contributions que j'ai faites au logiciel Blender en tant que salarié d'un studio, en particulier à travers la publication d'add-ons*.

Après avoir évoqué les différentes manières de publier un add-on, je reviendrai sur les problématiques de publications des add-ons sous licence libre, indépendamment du logiciel lui-même, mais s'insérant dans un plus vaste écosystème communautaire. J'expliquerai ensuite le processus d'intégration du système de pantins prototypé pour le film *Dilili* dans Blender, c'est-à-dire dans la version officielle du programme, téléchargeable par tous. Enfin, j'expliquerai le processus de contribution directe au code de Blender, en

tant que mainteneur d'un add-on écrit par un programmeur indépendant, et qui n'était plus supporté depuis plusieurs années.

II.3.3.1 Des manières de publier un add-on

Les add-ons* sont des petits programmes intégrés dans une plus grande application, utilisant son API* pour y ajouter des fonctionnalités. Il existe plusieurs moyens pour publier un add-on, et ils seront évoqués dans le contexte de l'utilisation de Blender.

Add-ons intégrés

Certains add-ons sont développés et maintenus par les mêmes développeurs que le programme principal. Ils sont livrés avec, et lui sont donc liés. Ainsi dans Blender, les fonctions fondamentales d'import et d'export de données vers des formats non-natifs sont contenues dans des add-ons livrés avec, et l'utilisateur peut décider de les activer quand il en a besoin.

Ces add-ons intégrés suivent le même processus de développement que le reste du programme, et les développeurs décident quels nouveaux add-ons développer, ou quels anciens add-ons plus compatibles doivent être retirés. Beaucoup d'add-ons intégrés au programme ont d'abord été développés indépendamment, par des individus ou des sociétés ayant besoin de fonctions non-prévues, et proposé ce code à l'inclusion dans le programme principal.

Magasins d'add-ons

Beaucoup de programmeurs créent aussi leurs propres fonctions sous la forme d'add-ons, et les publient sur des forums, des magasins en ligne moyennant paiement, ou sur leurs sites web, sans volonté que ces add-ons soient intégrés au programme principal. C'est le choix qu'a fait initialement la société en développant ses outils pour ses productions. En effet, la publication d'outils peut demander plus ou moins d'engagement de la part du développeur. Blender est utilisé par assez d'utilisateurs pour qu'il puisse exister un marché pour ces add-ons indépendants, et donc des magasins en ligne tels que

Blender Market et Gumroad. Bien que les add-ons pour Blender soient obligatoirement sous licence libre du fait de la licence copyleft GPL (cf. section II.1.3.3), un développeur peut décider de vendre son add-on, et de fait, quelques-uns d'entre eux parviennent à en tirer un profit non-négligeable. Mais dans ce cas, le fait de vendre un produit demande une maintenance et un développement réguliers, et un support aux utilisateurs devenus clients. De plus, la vente de programmes est une activité commerciale supplémentaire, qui n'est pas facile à intégrer avec la stratégie commerciale d'un studio d'animation, puisqu'elle demande des ressources substantielles, en développement et support donc, mais aussi en marketing et administration.

Publication indépendante

Le choix restant, qui est aussi le plus courant, est de publier ses add-ons gratuitement, sur une plate-forme indépendante du site du logiciel, et selon l'habitude des programmes sous licence libre, « tels quels », c'est-à-dire sans garantie de fonctionnement ni de support.

C'est la façon la plus simple de publier du code, puisqu'il suffit de le poster sur un site, blog, forum, ou, de préférence, forge de développement afin de faciliter le développement collaboratif.

II.3.3.2 Publication indépendante d'add-ons

Contraintes de la publication

Un programmeur écrivant un outil pour une production précise ne travaille pas de la même manière que s'il doit publier l'outil pour un usage plus général : il doit répondre d'abord à des problématiques de production précises, dans des contraintes de temps souvent difficiles, et le résultat est donc rarement un beau script* bien documenté et dénué de bugs, mais une solution ad-hoc prévue pour le pipeline* et les problématiques du studio.

La publication de ce code demande donc un travail de nettoyage, de documentation,

de test, et il faut bien évaluer avant de le faire le gain possible, pour les utilisateurs, et pour la société. Si l'outil est trop attaché à la société, soit parce qu'il s'agit d'un processus de niche, soit parce qu'il est indissociable du pipeline, il est probable que l'outil ne soit pas utile à grand-monde.

De plus, une fois le code publié, certaines étapes sont nécessaires pour que le code soit vraiment utilisable. Il faut en écrire la documentation, afin que les fonctionnalités du programme soient correctement décrites, ou du moins, que l'utilisateur ait une idée de son intérêt. Il faut ensuite faire un minimum de maintenance pour corriger les bugs trouvés par les utilisateurs, et intégrer le code éventuellement proposé par eux.

Construire avec la communauté

Alors, de la même manière qu'une communauté peut s'assembler autour d'un logiciel de création, un studio peut avec assez d'efforts, parvenir à fédérer autour d'un add-on* une communauté d'utilisateurs et de développeurs, qui peuvent à leur tour y contribuer.

Ils peuvent ainsi aider le studio dans le développement de ses propres outils, en l'utilisant, en rapportant des bugs ou en suggérant des fonctionnalités, voire en développant eux-mêmes des nouvelles fonctionnalités qui pourront y être intégrées directement. C'est l'argument évoqué par le développeur de jeu vidéo Jon Manning, du studio australien Secret Lab, dans une conférence à la Game Developers Conference [21]. Il explique avoir développé le système de dialogues Yarn Spinner [47] pour le jeu vidéo *Night in the Woods*, et avoir décidé de le rendre open source en cours de production. Les problématiques qu'il observe sont multiples. Il explique que l'intérêt pour un studio de faire cette démarche est d'avoir des utilisateurs qui peuvent utiliser, tester et rapporter des problèmes sur le programme, mais également développer directement, en proposant leurs améliorations, comme dans tout projet open source. Mais il souligne aussi l'importance de créer une communauté autour du projet, en communiquant avec les contributeurs potentiels, en les accueillant, en acceptant leurs propositions, même les plus imparfaites, et en engageant une dynamique de développement collaboratif. Il explique que la gestion d'une communauté est une activité radicalement différente du développement informatique,

mais qu'en cas de succès les bénéfices pour un projet ou une société sont plus grands que l'investissement en temps que cette gestion représente.

Je partage l'analyse de Manning, et je mettrai l'accent sur la condition de succès d'un tel projet open source, qui n'est pas seulement l'investissement des développeurs dans la communauté et de la publicité qui en est faite, mais également le potentiel du projet d'attirer les contributeurs, selon son utilité pour tous. Les sociétés doivent donc estimer l'intérêt de structurer une communauté de développement avant de se lancer dans une telle entreprise.

II.3.3.3 Maintenance officielle d'add-ons

Une des caractéristiques des projets open source est qu'ils accueillent les contributions externes. Ce principe est plus ou moins appliqué selon les projets, en fonction du modèle de gouvernance, de l'exigence technique et des ressources du projet. Quoi qu'il en soit, les utilisateurs sont généralement encouragés à contribuer ponctuellement, puis éventuellement à se voir confier des responsabilités au sein du projet, telles que la maintenance d'une partie du code, la communication avec la communauté, etc. Cet encouragement à la contribution peut s'étendre à des développeurs dans les studios.

Ainsi, j'ai été amené à corriger des bugs sur des add-ons officiels et à ajouter des fonctionnalités [18] dans le cadre de mon travail en entreprise. Ces changements n'ont d'abord pas été proposés à Blender, mais gardés au sein du studio, faute de temps pour créer un patch* nettoyé et facile à intégrer. Entre deux productions, j'ai pris le temps de proposer les patchs aux développeurs, tout en corrigeant les incompatibilités. Ils ont été acceptés et donc intégrés à Blender. Ce qui partait d'un développement à l'usage exclusif du studio a donc été logiquement reversé dans le commun.

La proposition de correctifs se fait via une forge, une interface dédiée au développement, sur le site web de Blender. Cette interface permet des discussions sur le design et

18. Par exemple, j'ai contribué à l'add-on AnimAll de Daniel Salazar, qui permet dans Blender d'animer des éléments normalement inaccessibles tels que les positions des sommets des maillages. À l'origine, pour chaque image de l'animation, l'add-on créait une clef d'animation pour tous les points en même temps, et j'ai ajouté la possibilité de ne créer la clef que pour les points sélectionnés, ce qui rend la gestion des clefs d'animation beaucoup plus claire.

l'implémentation du code, et à l'occasion d'un patch assez conséquent sur l'add-on AnimAll, le coordinateur du développement d'add-ons m'a proposé de faire officiellement la maintenance de l'add-on concerné. J'ai accepté, sachant que l'add-on était assez mature pour ne pas me demander un investissement de temps colossal.

Le responsable a donc engagé une procédure pour me faire devenir mainteneur. Cette procédure consiste à faire une demande officielle sur la liste de diffusion des développeurs, et à répondre aux questions d'un des chefs développeurs de Blender concernant ma motivation et mon expérience. La responsabilité de maintenance implique d'avoir accès au code source*, et le développeur m'a expliqué avec bienveillance que cette responsabilité était rarement confiée à des débutants n'ayant proposé que quelques contributions, puisque des mauvaises contributions pouvaient nuire à la qualité du code. J'ai pu lui expliquer que mon activité professionnelle m'avait conduit à développer et publier plusieurs add-ons indépendants, et à implémenter la modularisation de Rigify décrite plus bas. Il m'a confié l'accès au dépôt de code afin que je puisse y verser directement mes contributions, sans avoir besoin à l'avenir de créer une demande sur le site web. J'ai par la suite mis à jour, restructuré et corrigé des bugs dans plusieurs add-ons.

La responsabilisation des utilisateurs dans le développement d'un programme open source permet dans une certaine mesure de répartir les coûts de développement, qui ne sont alors pas assurés intégralement par un éditeur, mais engagés par des bénévoles de la communauté, ou par des développeurs au sein de studios ou d'autres entreprises. Ces développeurs utilisent une partie de leur temps pour contribuer au logiciel open source qu'ils utilisent eux-mêmes. En étant responsabilisés, les utilisateurs peuvent s'impliquer directement dans le développement, et apprendre par la même occasion les pratiques du développement collaboratif. Une telle démarche demande des ressources de la part de l'équipe centrale de développeurs, car l'accueil et le support des nouveaux venus, la critique de leur code et même la modération des discussions prennent du temps. Il y a nécessairement des problèmes relationnels à régler dans une telle collaboration, mais la confiance accordée aux développeurs peut créer une motivation à contribuer au projet, et à faire profiter de ses améliorations à la communauté.

II.3.3.4 Harmonisation du code entre contributeurs

Un problème particulier peut survenir lors de la contribution décentralisée de plusieurs acteurs à un même code. En effet, dès que deux développeurs modifient en parallèle un programme, des conflits peuvent apparaître entre leurs versions, et ils devront les harmoniser s'ils veulent les intégrer dans la version principale. C'est un problème classique de développement informatique qui apparaît dans toute équipe de plus d'une personne, et des méthodologies existent pour le pallier. Mais le problème est exacerbé quand les développeurs travaillent en complète indépendance sur des branches distinctes, que l'on appelle forks*. Les studios peuvent être confrontés à ce problème quand ils forkent un programme pour répondre rapidement à des problèmes rencontrés en production, et souhaitent ensuite proposer leurs changements à la communauté, en passant par les développeurs du logiciel.

Concrètement, le problème d'un fork est qu'il est difficile à maintenir. En effet, si la version officielle est modifiée, la version dérivée devra également appliquer ces modifications pour rester compatible. Si un studio veut proposer sa version dérivée, les utilisateurs voulant l'utiliser devront la substituer à la version officielle, et ne pourront donc pas utiliser les deux versions en même temps. Pire, si plusieurs développeurs créent des forks incompatibles entre eux, l'utilisateur devra réinstaller l'add-on* à chaque fois qu'il voudra utiliser des fonctions de l'une ou l'autre version, ce qui le conduira à terme à rencontrer des problèmes d'incompatibilité insolubles.

Intégration des pantins dans Blender

Pour illustrer la manière dont ce problème peut se poser, et une solution adoptée pour le résoudre, j'aborderai l'exemple d'un outil de rigging développé au cours de plusieurs projets dans l'entreprise Les Fées spéciales.

Cet outil est fondé sur un add-on* existant et livré avec Blender, Rigify (cf. section III.3.1). Il a été largement modifié pour inclure les fonctionnalités nécessaires aux productions de la société. Ce faisant, l'add-on est devenu incompatible avec la version d'origine. Ce code incompatible a néanmoins été publié en cours de production, après

quelques mois de développement.

Tout comme j'ai décrit la façon dont je m'occupais de la maintenance de plusieurs add-ons livrés avec Blender en tant que membre de la communauté, Rigify était à l'époque maintenu par le studio italien MAD Entertainment. Cette maintenance était relativement indépendante de la Blender Foundation, l'organisme qui coordonne le développement de Blender.

J'ai rencontré les mainteneurs lors de la Blender Conference en 2017, à l'occasion d'une présentation de leurs développements sur Rigify, et d'une présentation que j'ai donnée de la production du film *Dilili à Paris* et des outils de pantins. La discussion qui s'est engagée lors de cette rencontre a porté sur la possibilité d'harmonisation entre les forks existants et futurs de Rigify. Puisque les studios sont toujours amenés à créer des fonctionnalités très différentes, il était nécessaire qu'ils puissent le faire sans créer d'incompatibilités. Une solution pour cela était de restructurer Rigify, pour lui donner une architecture modulaire et extensible.

Cela implique que l'utilisateur de l'add-on puisse facilement créer, distribuer et installer des modules de fonctionnalités supplémentaires, équivalents à des add-ons de l'add-on. Par exemple, le système de création de pantins peut être réécrit pour être un module pour Rigify plutôt qu'une modification incompatible. Ainsi, l'utilisateur peut installer plusieurs modules, et les mélanger au sein d'un même personnage.

Difficulté du développement collaboratif

Ce travail de modularisation a demandé un temps de développement de plus d'un an, et a été réalisé conjointement par les mainteneurs et moi, en se fondant sur des allers et retours du code et des discussions en ligne. Si la décentralisation du développement a des avantages sur une gouvernance centrale, elle a cependant des conséquences sur la bonne marche du projet.

En effet, les développeurs au sein des studios travaillent en priorité sur le pipeline* interne, et ne sont donc pas forcément disponibles pour la collaboration avec une communauté open source. Cette indisponibilité a créé des tensions lors du développement,

quand les mainteneurs, prudents, ont laissé passer des mois avant de me donner les retours sur mon code nécessaires à une intégration dans Blender. Ils l'ont finalement fait après qu'un nouveau développeur les a pressés d'intégrer mes modifications, car ce retard l'empêchait de travailler sur une autre partie du code.

Finalement, le code de modularisation a été intégré avec la version 2.80 de Blender, et j'ai alors pu modifier le système de pantins pour qu'il soit distribué sous forme de module. Il n'est donc pas directement intégré dans Blender, et il est toujours développé indépendamment au sein du studio, mais il est maintenant compatible puisqu'il ne crée pas de fork. De plus, cette modularisation donne la possibilité aux riggers-scripteurs de tous les studios de travailler plus proprement, et de publier leurs rigs.

Conclusion

Tout comme pour ses particularités juridiques et économiques, le logiciel libre et open source peut offrir des avantages techniques aux studios : logiciels performants, fonctionnalités novatrices, développement rapide, support technique efficace, interopérabilité et pérennité. Mais il peut aussi être source de frustrations lorsque les logiciels sont non-conventionnels, mal supportés, mal conçus ou simplement abandonnés.

La possibilité de chacun de développer le programme en collaboration avec la communauté, si elle permet des utilisations nouvelles des programmes, demande de revoir la façon dont on considère l'outil de production, en le voyant moins comme un produit que comme un bien commun en perpétuelle évolution. Cette évolution peut demander une implication inhabituelle des utilisateurs, parmi lesquels les studio.

Conclusion de la deuxième partie

Cette partie a abordé concrètement la rencontre entre les logiciels libres et les studios d'animation.

Le chapitre II.1 a montré le rôle juridique du logiciel libre, fondé sur les quatre énoncés qui font partie de sa définition : utiliser, étudier, modifier et distribuer le programme sous sa forme de code source. Cette analyse a permis d'aborder les points suivants :

— l'importance pour les studios, comme pour n'importe quelle société faisant usage de logiciels, d'avoir accès au code pour se garantir la pérennité et la maîtrise de ses propres données et processus ;

— les différents types de licences libres existantes, et la difficulté que représente le choix des licences logicielles pour les studios ;

— le rapport qui existe entre le mouvement du logiciel libre et celui des communs, autour des licences de libre diffusion telles que les licences Creative Commons, et en quoi le choix de ces dernières est délicat pour les studios à l'heure actuelle.

Dans le chapitre II.2 nous avons vu quelles conséquences économiques l'usage des logiciels libres pouvait avoir dans les studios :

— les modalités d'accès aux logiciels, et comment le logiciel libre n'est pas qu'une alternative gratuite aux logiciels propriétaires ;

— les raisons qui peuvent pousser les studios à écrire leurs propres programmes, et le rôle du logiciel libre dans l'accès à cette pratique ;

— les modes de financement possible du logiciel libre, autant au sein du studio qu'envisagé dans un réseau plus large de développeurs.

Enfin, le chapitre chapitre II.3 a abordé trois facettes concrètes de l'usage des logiciels

libres :

— le rôle de l'innovation technique des logiciels libres de création, et leur place parmi les éditeurs ;

— la fonction déterminante de la standardisation des programmes et formats pour les studios, et de quelle manière elle repose sur le logiciel open source ;

— et enfin la manière pour les studios de contribuer eux-mêmes au développement des logiciels qu'ils utilisent, au sein des départements R&D* ou TD*, et les bénéfices que peut en tirer la communauté mondiale des utilisateurs de logiciels libres pour l'animation.

Troisième partie

Vers un pipeline libre

Introduction

L'utilisation des logiciels libres dans le secteur du cinéma d'animation est relativement récente et minoritaire, mais comme dans d'autres secteurs, leur adoption tend à s'accélérer, et de nombreux studios commencent à les considérer comme des solutions viables.

Cette partie s'attachera à décrire l'utilisation des logiciels libres que j'ai pu observer et développer au sein du studio Les Fées spéciales pendant ma convention Cifre, en groupant ces usages par département. Selon les cas, deux approches complémentaires seront adoptées : une description des logiciels utilisés pour résoudre un problème spécifique, et une analyse des outils et processus développés pour les projets, et publiés sous licence libre. Cette double approche permet d'étudier, d'une part, les qualités propres de certains programmes libres de création, et d'autre part le besoin constant de développement au sein des studios, et les bénéfices qu'ils peuvent tirer de la mutualisation propre au logiciel libre.

Chapitre III.1

Projets pour l'émergence d'un pipeline libre

Introduction

Cette thèse s'est déroulée en entreprise, dans le cadre d'un contrat Cifre (cf. section I.1.1). C'est principalement en travaillant en production sur les projets que j'ai pu observer les implications de l'utilisation du logiciel libre pour les studios, mais également dans une moindre mesure, pour un artiste développeur indépendant. Avant de développer les utilisations concrètes du logiciel libre en production, il faut avoir une idée des problématiques de production liées aux projets. J'exposerai donc dans ce chapitre quatre des projets sur lesquels j'ai travaillé : un long-métrage d'animation, *Dilili à Paris* (section III.1.1), une exposition et deux documentaires formant le projet *Antarctica* (section III.1.2) et une exposition permanente du musée de Lodève (section III.1.3).

III.1.1 *Dilili à Paris*

Dilili à Paris est un film d'animation du réalisateur Michel Ocelot, auteur notamment de *Kirikou et la Sorcière* et d'*Azur et Asmar*. Sorti fin 2018, il raconte l'histoire de l'héroïne éponyme, petite fille kanak venue à Paris pour une exposition de zoo hu-

main, à la Belle Époque. Tout en visitant de nombreux lieux célèbres de la ville, et des personnalités de l'époque — artistes, scientifiques, politiques — elle devra élucider de mystérieux enlèvements de fillettes qui terrorisent la population.

Ce film a été en préparation longtemps avant sa mise en fabrication. En effet, outre l'écriture du scénario qui demande un temps long, le film a dû passer par des étapes de recherche de financement, de storyboard et d'animatique. Ces étapes avaient donc commencé depuis plusieurs années lorsque les Fées spéciales ont commencé leur participation, début 2016.

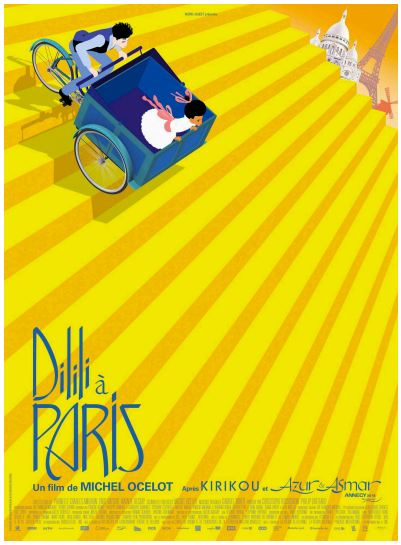


FIGURE III.1.1 – Affiche du film *Dilili à Paris*.

L'organisation de ce projet était complexe, parce qu'elle impliquait quatre équipes différentes : deux à Paris, travaillant dans les mêmes locaux, un troisième à Montpellier, plus une équipe du premier studio, à Bruxelles. Le studio parisien, Mac Guff, et son antenne belge étaient chargés de la modélisation et de l'animation des personnages principaux, du rendu et des effets spéciaux, et du compositing. Les décors ont été réalisés par une équipe engagée pour l'occasion par la société de production Nord-Ouest. Les décorateurs devaient tenir compte de la volonté de Michel Ocelot d'utiliser des photos de la ville, prises par l'auteur lui-même dans le Paris d'aujourd'hui. La majorité des décors du film a donc été retouchée pour enlever les éléments anachroniques, et retrouver une esthétique et une impression 1900. D'autres décors, comme la tour Eiffel, ont été imaginés et entièrement créés en 3D, ou dans certains cas peints sans base photographique. Les Fées spéciales, à Montpellier, ont pour leur part créé le layout et les nombreux figurants.

Dilili à Paris est le premier projet sur lequel ont travaillé les Fées spéciales. C'est donc pour ce film qu'a été conçu la première version du pipeline* de fabrication du studio. Dans ce contexte, j'ai travaillé sur un système d'animation 3D de personnages en papier découpé, qui sera détaillé dans la section III.3.1.

Le pipeline du film a été spécialement conçu pour accommoder les échanges entre les différents sites. En particulier, le layout et l'animation des foules réalisées par les Fées devaient être utilisées par Mac Guff, et se retrouver telles quelles dans leur propre pipeline. Des outils spécifiques ont dû être écrits, et ils seront détaillés dans le section III.5.2.

III.1.2 *Antarctica*

Les Fées spéciales ont travaillé sur deux projets fondés sur une même expédition en Antarctique, organisée en 2015-2016 par le documentariste de cinéma Luc Jacquet (*La Marche de l'empereur*, *Il était une forêt*), accompagné par le photographe et plongeur Laurent Ballesta. L'expédition avait pour but d'étudier la vie présente sur le continent Antarctique, par des plongées sous-marines dans des conditions extrêmes, ainsi que des phénomènes naturels ayant lieu sur et autour du continent. De nombreuses images en ont été rapportées, et plusieurs livres et films ont été édités, ainsi qu'une exposition au musée des Confluences à Lyon en 2016.

Cette exposition est le premier projet muséographique sur lequel ait travaillé le studio Les Fées spéciales. À cette occasion, nous avons créé des films animés illustrant des phénomènes comme les courants marins et les particularités de la plongée des phoques de Weddell.

Le deuxième projet est un ensemble de deux films documentaires diffusés sur Arte début 2017. Ces films montrent des images prises lors de l'expédition, et des plans* animés les accompagnent pour contextualiser ou illustrer le propos des auteurs.

III.1.2.1 Exposition au musée des Confluences

Cette exposition illustre la vie sur le continent Antarctique. Surtout constituée de vidéos prises lors de l'expédition, les Fées spéciales y ont contribué en créant quatre animations diffusées sur des écrans disséminés au sein de l'exposition.

Les images prises sur place mettent en scène les animaux sous l'eau et au-dessus. Des plongeurs de l'équipe de Laurent Ballesta ont rapporté des images nouvelles d'espèces

autochtones et inconnues jusque-là : algues, étoiles de mer, poissons, crustacés, espèces non identifiées et incroyables. Deux espèces sont particulièrement traitées : le manchot empereur et le phoque de Weddell.

Le temps de production de ce projet a été court, mais la diversité des programmes l'a rendue très dense. La production muséographique, si elle utilise les mêmes techniques que celles du cinéma, s'attache moins au volume, et les éléments produits sont donc beaucoup plus différents les uns des autres. Au total, nous n'avons fabriqué que quatre plans, mais chacun était si unique qu'aucune réutilisation n'était possible.

FIGURE III.1.2 – Affiche de l'exposition *Antarctica*.

Chacun de ces quatre plans se fonde sur une charte graphique établie par la designer graphique Marie Saby et par le directeur artistique de la société, Eric Serre. La charte se fonde sur les informations et dessins fournis par l'équipe scientifique du musée ou de l'exposition, une bible scientifique. Bien souvent, ces informations ne sont pas suffisantes pour créer de toute pièce un plan, et il faut alors demander des données complémentaires, ou trouver une autre source.

Ces plans sont diffusés sur des écrans dans l'exposition, illustrant des phénomènes observés sur le continent, en regard avec le reste de la muséographie. Par exemple, une des premières salles de l'exposition reconstitue une pièce de la base scientifique, avec les équipements de plongée, des instruments de mesure, etc. Dans cette salle, nous avons conçu un programme montrant le trajet en avion effectué par l'équipe pour rejoindre la base.

Chaque programme devait boucler sur lui-même afin de recommencer directement après avoir terminé. Mes recherches de master sur la boucle d'animation m'ont été bien utiles pour trouver des solutions créatives à ce problème.

III.1.2.2 Films documentaires télévisés

Antarctica, sur les traces de l'Empereur et *Les secrets des animaux des glaces* sont deux films documentaires diffusés sur Arte en 2017. Très liés à l'exposition du musée des Confluences, leurs images sont issues de la même expédition en Antarctique, et montrent la diversité biologique du continent, tout en illustrant des phénomènes climatiques mondiaux, et propres à ce continent. La majorité des films est en prise de vue réelle, y compris des séquences de plongée sous-marine le long des côtes du continent, un exploit inouï et inédit. Quelques séquences animées sont dispersées dans les films, illustrant divers phénomènes. Ce sont sur ces séquences que nous avons travaillé.

La production a été beaucoup plus longue, et plutôt plus laborieuse que pour l'exposition, en raison de la différence de contexte et d'interlocuteurs. En effet, si la personne en contact sur l'exposition était le réalisateur Luc Jacquet, chargé du commissariat d'exposition, deux personnes différentes étaient à la réalisation des films : Marianne Cramer et Jérôme Bouvier. Les deux films avaient des durées différentes (52 et 90 minutes), et racontaient des histoires différentes. Certains plans que nous avons fabriqués devaient être utilisés dans les deux films, mais les réalisateurs n'avaient pas systématiquement les mêmes visions. Le fait d'avoir deux interlocuteurs et deux validations à obtenir pour les mêmes images complique singulièrement le travail de la production.

De plus, une contrainte inédite pour nous, et qui peut sembler anecdotique mais a entraîné de lourdes conséquences, est celle du format, en 4K (Ultra HD). Bien sûr, les difficultés techniques liées aux nouveaux formats plus lourds sont loin d'être nouvelles, au point de devenir un poncif avec l'apparition régulière de nouveaux standards (numérique, HD, stéréoscopie, 4K, HDR, etc.), mais c'est la première fois que nous en étions frappés directement, ce qui nous a demandé des ressources et des compromis non anticipés.

Beaucoup de plans fabriqués pour ces deux projets, l'exposition et les films, n'ont pas présenté de particularités par rapport au sujet d'étude, mais certains, par leur caractère de visualisation scientifique, ont permis d'explorer des problématiques originales concernant l'utilisation de données scientifiques ouvertes. Je parlerai de ces problématiques dans le chapitre III.6.

III.1.3 Exposition pour le musée de Lodève

Entre août 2017 et juillet 2018, la majeure partie du temps de production de l'entreprise a été consacrée à son deuxième projet de grande envergure : la réalisation de programmes audiovisuels et la conception d'objets pour le musée de Lodève, spécialisé dans l'archéologie, la géologie, et l'œuvre du sculpteur Paul Dardé. Lodève est une petite commune d'environ 7 500 habitants située dans le département de l'Hérault, à une cinquantaine de kilomètres de Montpellier. Malgré sa petite taille, la commune possède un musée de rayonnement national du fait de ses expositions temporaires. Fortement ancré dans le territoire, il montre les nombreuses richesses du Lodévois environnant. Les pièces de ses collections sont d'époques diverses : les plus anciennes, des restes fossiles de plantes et d'animaux, ont des centaines de millions d'années, tandis qu'un vaste pan de la collection a été fabriqué par des humains ayant vécu dans la région il y a plusieurs millénaires, pendant la préhistoire.

Cette grande variété est propice, non seulement à l'étude, qui est le travail quotidien des scientifiques du musée, mais également à l'exposition pour le grand public. À l'occasion d'ambitieux travaux de rénovation ayant duré quatre ans, le musée a fait un appel d'offre pour concevoir une nouvelle exposition. Cela signifie non seulement restaurer le bâtiment, concevoir une nouvelle identité visuelle, de nouveaux lieux d'exposition et une nouvelle muséographie, mais également, et c'est là que nous sommes concernés, des programmes animés permettant d'illustrer, d'expliquer, de vulgariser des connaissances scientifiques au travers des pièces du musée. Les Fées spéciales ont répondu à l'appel d'offre pour la dernière période de la rénovation, alors que le gros œuvre se terminait et que les ouvriers s'activaient aux installations électriques et mobilières.

L'appel d'offre portait sur les trois domaines d'étude du musée : l'archéologie, la géologie et les beaux-arts. Ces trois domaines sont organisés dans le musée selon trois parcours, chacun dans une aile du bâtiment. Nous avons donc travaillé avec les responsables de ces départements pour concevoir les programmes. Ceux-ci sont d'une grande variété, allant de courts-métrages animés montrant la vie des habitants de la région à la préhistoire, à des pièces volumiques fabriquées selon des dessins 3D, en passant par

des schémas explicatifs sur les phénomènes géologiques, projetés sur des cartes en relief installées sur des tables dans l'exposition. La multiplicité des supports a demandé un pipeline* souple, voire laxiste, pour travailler en parallèle.

Les vidéos réalisées pour le parcours de beaux-arts n'ont pas demandé de 3D, mais seulement du montage et du motion design, aussi je n'en parlerai pas plus avant, pour me concentrer sur l'archéologie et la géologie. Je présenterai maintenant les différents programmes pertinents pour cette thèse, et exposerai dans la partie III les problématiques de ce projet et les développements nécessaires dans le pipeline pour le mener à bien.

III.1.3.1 Archéologie : films animés

Le parcours d'archéologie de l'exposition permanente du musée, *Empreintes de l'Homme*, expose l'environnement et la vie d'hommes préhistoriques dans la région au néolithique. Chaque salle explique une technique ou une pratique particulière de ces peuples, en présentant des artefacts des collections ainsi que des courts films animés mettant en scène des personnages de l'époque.

Pour ce programme, nous avons fait neuf films animés, durant chacun entre une et quatre minutes, et utilisant la technique des pantins, déjà éprouvée sur *Dilili à Paris*. Les films sont constitués de saynètes muettes, mettant en scène des personnages du néolithique dans leurs villages, ou en dehors, dans la région. On les voit accomplir leurs activités quotidiennes (chasse, pêche, fabrication d'outils, puisage d'eau, inhumation des morts, fabrication d'un dolmen, etc.).

III.1.3.2 Modélisation et création d'objets physiques

En plus des animations du parcours archéologique, les Fées spéciales se sont occupées de la conception numérique de pièces volumiques pour une fabrication 3D. D'autres pièces ont été conçues pour le parcours géologique, intitulé *Traces du vivant*. Une autre

entreprise, Sequoia MD [1], s'est occupée de la fabrication. En effet, nous ne disposions ni des compétences, ni de l'outillage nécessaires. Nous avons cependant collaboré avec la société afin d'être certains à chaque étape que les fichiers que nous leur envoyions étaient conformes à leurs demandes, et réalisables avec leur matériel, dans les délais imposés. Nous avons donc pu étudier les contraintes de fabrication. Plusieurs pièces différentes ont ainsi fait l'objet d'une conception numérique.

Pour toutes ces pièces, nous avons utilisé Blender pour ses capacités de modélisation 3D, mais d'autres programmes ont aussi fait partie du processus de fabrication, selon les besoins. Je parlerai dans le chapitre III.7 des pièces sur lesquelles j'ai personnellement travaillé.

Conclusion

Les projets exposés dans ce chapitre ont eu lieu sur une période de presque trois ans. C'est au cours de ces projets que j'ai exploré les possibilités et les pratiques du logiciel libre dans le pipeline de fabrication d'un studio d'animation, qui seront l'objet des chapitres suivants.

Durant cette période, j'ai également élaboré un projet personnel, qui n'est pas strictement en lien avec les studios d'animation puisqu'il avait pour objet la création d'une application interactive pour le web. Il est néanmoins éclairant pour comprendre la méthode que peuvent adopter les TDs* lors de la phase de veille pour développer des scripts* au sein des studios. Ce projet sera présenté en annexe B.

1. Cette entreprise basée dans les Yvelines est spécialisée dans l'agencement et la création numérique pour des expositions, des collectivités, et des clients privés.

Chapitre III.2

Préproduction

Introduction

Après cette présentation des projets ayant alimenté mon étude du logiciel libre en production, j'aborderai maintenant chaque département pour lequel le logiciel libre a été utilisé, ou du moins fait l'objet d'une étude. Je commencerai logiquement par les premières étapes de la chaîne, que constitue la préproduction.

La préproduction [1] est une phase précédant la fabrication. Il s'agit pour partie d'un travail de production [2], et pour partie d'un travail de conception artistique et de recherche technique. J'aborderai dans ce chapitre les problématiques propres à l'élaboration du storyboard, de l'animatique et du layout (cf. section I.1.2), indispensables à la bonne marche d'un projet d'animation.

III.2.1 Storyboard et animatique

J'ai décrit ces différentes étapes dans la première partie (cf. section I.1.2.1). Parmi ces étapes préparatoires de conception et de mise en scène, la première est le storyboard. Elle

1. Il convient de lever l'ambiguïté entre les deux sens du mot « production » : les métiers de production (producteur, assistant de production, etc.) et la phase pendant laquelle le film est fabriqué. Je parlerai plutôt de fabrication dans ce dernier sens.
2. Financement, co-production, etc.

n'est pas aussi technique que les étapes ultérieures de fabrication, mais gagne néanmoins à être considérée dans le cadre du pipeline* dans la mesure où ce document sera réutilisé et transmis aux autres départements.

Pour les petits films d'animation, l'équipe réalisant le storyboard est réduite, entre un et quelques graphistes, et la mise en scène ne demande pas de nombreuses itérations ni une gestion rigoureuse des versions. Ces projets ne nécessitent donc généralement pas de pipeline complexe. Les graphistes s'organisent comme ils le souhaitent, et le choix des logiciels au regard du pipeline est moins important que sur une grosse production.

Qu'il s'agisse d'une petite ou d'une grosse production, le storyboard peut bénéficier d'outils numériques adaptés pour faciliter sa conception et son intégration dans la chaîne de fabrication.

III.2.1.1 Fonctionnalités essentielles des logiciels de storyboard

L'observation de la pratique des storyboarders permet de dégager les fonctionnalités essentielles à intégrer dans les logiciels de conception.

Du storyboard à la nomenclature de projet

Le storyboard est traditionnellement un document papier qui expose les indications de mise en scène et d'autres données qui représentent l'organisation structurelle de l'œuvre et de sa fabrication : séquences, plans*, images, raccords, etc. (cf. section I.1.2.1). Dans l'équivalent numérique, ces indications peuvent être extraites pour former la base de l'ensemble du pipeline.

En effet, la nomenclature de montage est un ensemble de métadonnées qui se reflète directement dans l'organisation informatique à la base du pipeline. L'ajout d'un plan au storyboard influence la structure du film et donc l'organisation informatique (arborescence de fichiers, suivi numérique de la fabrication, etc.).

De plus, les vignettes qui constituent le storyboard peuvent en être extraites et intégrées à l'animatique. Dans ce cas, le storyboard n'est plus un document isolé, mais bien le point de départ de la chaîne de fabrication.

Double niveau de lecture

Le storyboard permet de visualiser deux informations simultanément : la composition de l'image dans le plan et l'enchaînement des plans dans la continuité de la séquence. Dans un storyboard traditionnel, on dessine sur une bande les vignettes l'une après l'autre, et on les voit dans leur continuité, chacune entourée des plans précédent et suivant. Il s'opère donc un aller-retour entre le plan représenté par une vignette, et la séquence composée de plusieurs plans.

Le numérique permet d'afficher un dessin en plein écran pour plus de précision, mais il est important de retrouver cette possibilité de changer d'échelle de lecture, en affichant également le strip. Ce changement doit pouvoir s'opérer de manière interactive et instantanée, et le graphiste doit pouvoir passer facilement d'une vue à l'autre.

Storyboard et animatique : deux vues des mêmes informations

Puisque le storyboarder crée la mise en scène du film, il gagne à déterminer une durée approximative pour les plans avant l'étape du montage. En effet, le film se perçoit dans la durée, et le rythme est une composante essentielle de son expression. Un minutage précis participe donc à la narration.

Or, contrairement à la conception traditionnelle du storyboard, proche de la bande dessinée, les logiciels de storyboard permettent de mettre en place la durée des plans en même temps que la composition de l'image. Les storyboarders peuvent donc avec l'aide du programme concevoir l'animatique en même temps que le storyboard.

De fait, dans un programme permettant de paramétrer les durées des plans que les vignettes sont dessinées, il est facile de générer l'animatique ou le storyboard plusieurs fois en cours de production. Les deux documents n'ont pas les mêmes finalités, mais représentent tous les deux le film, l'un sous la forme de planches, l'autre sous la forme de vidéo qui constitue un montage provisionnel.

III.2.1.2 Solutions existantes

Les logiciels de storyboard commerciaux les plus utilisés en animation en 2019, ToonBoom Storyboard Pro et TVPaint permettent tous deux les fonctionnalités décrites précédemment.

ToonBoom Storyboard Pro n'est conçu que pour le storyboard. TVPaint est un logiciel d'animation 2D, mais intègre un module dédié au storyboard. La manière de créer un storyboard dans ces logiciels est proche de celle employée pour animer des plans, mais plus grossière puisque le minutage n'est pas à l'échelle du plan mais de la séquence, ou du film.

Pour cela, ces logiciels disposent d'une forme de timeline*, sur laquelle on organise les vignettes des plans. L'utilisateur peut dessiner chaque vignette indépendamment, l'annoter avec les dialogues, modifier la durée, ajouter du son, animer une caméra virtuelle, et faire un premier travail de montage en réordonnant les plans, tout en ayant une interface dédiée au dessin. Dans TVPaint, les plans et séquences ainsi créés pourront servir de base pour les plans du film, et une arborescence reflétant l'organisation du film pourra être générée d'après le storyboard.

III.2.1.3 Storyboarder

Du côté des solutions libres, Storyboarder [45] est un programme open source développé depuis la fin 2016 au sein de la société de production Wonder Unit. Il est uniquement conçu pour le storyboard et l'animatique. Il peut également être intégré dans une chaîne de fabrication de film.

Dans ce programme, le processus de conception est beaucoup mieux intégré que ce que permettent les logiciels purement dédiés au dessin comme Photoshop ou même dans une certaine mesure Krita, qui permet aussi de faire de l'animation. Une première différence fondamentale se situe au niveau des métadonnées. En effet, il permet bien sûr de dessiner les vignettes du storyboard, mais également de les annoter comme décrit précédemment (action, dialogue, etc.), d'y ajouter du son, de les réordonner et de les

minuter précisément. Une seconde différence concerne les entrées/sorties : Storyboarder, puisqu'il n'est prévu que pour la création de storyboard, prévoit plusieurs manières de faire passer les données depuis et vers les autres départements.

Intégration dans le pipeline

Il est possible de construire le storyboard à partir d'un scénario rédigé dans un format spécialisé [3], et ainsi obtenir la structure complète du film, déjà annotée avec les dialogues et l'action.

Ensuite, tout comme les programmes décrits plus haut, Storyboarder permet l'export vers un fichier PDF pour échanger le document, ou vers une animation, et ainsi obtenir directement une animatique. Mais il est également possible d'exporter les images individuellement et de les traiter dans un outil à part, soit pour générer le storyboard avec une autre mise en page que celle proposée par l'application, soit pour en faire tout autre chose, par exemple les importer dans un outil de suivi de production ou de gestion d'assets.

Enfin, le format de fichier lui-même utilise JSON*, ce qui permet d'intégrer facilement Storyboarder dans un pipeline de fabrication. Par exemple, le fichier de storyboard peut être utilisé pour créer des fichiers de montage, l'arborescence des plans sur le serveur de travail, ou les informations de suivi de production (cf. section I.1.2.4).

L'intégration de ce logiciel n'a pas encore été tentée au sein des Fées spéciales, d'abord faute de temps pour développer le pipeline, puis faute d'une œuvre d'envergure suffisante pour susciter ce développement. Mais dans la perspective d'un pipeline entièrement libre, ce logiciel semble la piste la plus intéressante à développer car il pourrait très bien s'intégrer au pipeline, sans que celui-ci ne dépende d'un seul logiciel pour toute la production comme c'est le cas avec les solutions propriétaires clef en main.

3. Storyboarder lit le format propriétaire Final Draft et le format ouvert Fountain.

III.2.1.4 Storymatic, une solution interne aux Fées spéciales

En 2015, Flavio Perez, directeur technique aux Fées spéciales, a créé un petit programme permettant d'assembler les vignettes d'un storyboard en strips et d'obtenir un storyboard complet et à jour à tout moment. Il présente cet outil, baptisé Storymatic, dans son mémoire de master [20].

À l'époque, deux raisons ont motivé ce développement : d'une part, il n'y avait pas de solution libre complète pour le storybourd, puisque Storyboarder n'existait pas encore. D'autre part, sur les deux projets où ce programme a été utilisé, les storyboards avaient été dessinés sur papier par le réalisateur, ce qui excluait l'utilisation d'un logiciel de dessin, et a fortiori d'un logiciel de storyboard. Il était donc nécessaire de traiter ce document pour en extraire les informations de mise en scène et les intégrer au pipeline.

Difficultés de création sans outil dédié

Le but de cet outil est de permettre au storyboarder de dessiner ses vignettes et de les annoter indépendamment, pour ensuite générer le storyboard. Pour expliciter l'intérêt d'un tel outil, j'exposerai d'abord le processus mis en œuvre sur une production qui n'en disposait pas[4] : les animations pour le musée de Lodève (cf. section III.1.3.1).

Pour cette production, la graphiste chargée du storyboard, Léa Cluzel, a dessiné les vignettes dans le logiciel libre de peinture Krita. Elle rapporte que les storyboards ont été dessinés et assemblés vignette par vignette, et planche par planche, en ajoutant manuellement les annotations :

> « Quand je parlais d'export à la main, c'était pour la première étape de fabrication du storyboard. On créait un fichier par plan, et on exportait à la main les images (il pouvait y avoir entre 1 et 10 images par plan). Du coup les retakes* étaient pénibles car le moindre changement nous faisait faire de nombreuses manipulations : passer d'un fichier à l'autre sans arrêt, renommer le fichier si le numéro de plan avait

4. La production ayant souffert de problèmes d'organisation, Storymatic n'a pas pu être mis en place par l'équipe technique et les graphistes ont donc utilisé des méthodes plus primitives et moins automatiques.

> changé, réexporter chacune des images… et refaire les planches des storyboards, avec la numérotation et les annotations à la main. [5] »

Les vignettes des plans étaient donc, dans ce processus, organisées en fichiers, et à chaque version du storyboard, chaque fichier devait être rouvert, exporté, nommé, importé dans le fichier de planche, puis mis en page et annoté avant d'obtenir le document.

Ce processus avait aussi l'inconvénient majeur que les métadonnées associées aux plans, telles que leur numérotation et leurs durées, n'y étaient pas formellement attachées, mais ajoutées manuellement à chaque export, en les collant sur l'image. Le logiciel Krita n'étant pas conçu pour le scénario, il était impossible de réaliser cette opération automatiquement.

Automatisation du processus

Afin de simplifier ce processus, Storymatic permet d'assembler les vignettes de storyboard en bandes et en planches, à partir de dessins soit numérisés, soit directement issus d'un programme de dessin. En effet, une mise en page manuelle du storyboard est un processus non seulement très long, mais aussi source d'erreur, et surtout, destructif : la suppression ou l'ajout d'une vignette demande de refaire toute la mise en page, puisqu'elle dépend de l'ordre et du nombre de vignettes précédentes. Or, un storyboard est un document amené à évoluer de nombreuses fois au cours de sa conception ou au cours de la fabrication, au gré des allers et retours avec les commanditaires et interlocuteurs. Ajoutons qu'il faut que les dialogues et indications de mise en scène figurent au bon endroit pour chaque version du storyboard. Un outil d'édition automatique devient donc vite incontournable pour simplifier la tâche des graphistes.

Dans Storymatic, le nommage des images est à la charge du graphiste, et détermine automatiquement leur ordre dans le storyboard. Le graphiste commence donc par nommer ses vignettes selon la place du plan* dans le film, par exemple `S02-P08.png` pour le huitième plan de la deuxième séquence. Un fichier texte à part permet de renseigner les métadonnées associées au plan, telles que dialogue, action, décor, transitions, etc.

5. Entretien avec l'auteur, 24 juin 2019.

À partir de ces indications, le programme Storymatic crée automatiquement un fichier PDF du storyboard, imprimable et échangeable.

Cette manière de procéder a très bien fonctionné sur le long-métrage *Dilili à Paris* et pour la série animée *Bonjour le monde*. Elle est simple et demande peu de formation, seulement de la rigueur dans la création des éléments. De plus, dans la mesure où elle n'est pas spécialement attachée à un type d'œuvre particulier, comme l'est par exemple TVPaint avec l'animation 2D, elle peut être utilisée sur différents projets d'animation 2D, 3D ou hybride. Elle ne demande pas d'acquérir ou d'apprendre un programme complexe pour cette tâche techniquement assez simple, et peut facilement s'adapter à un pipeline.

Publication hasardeuse

Ce programme a été publié par les Fées spéciales en 2016 [52]. Cependant, cette publication a souffert de deux problèmes : son rôle d'outil maison et son manque de licence.

En effet, comme souvent dans les studios d'animation, l'outil a été développé au cours de projets précis, et non pas prévu dès le départ pour une publication. Il en résulte que le logiciel n'est pas conçu pour les usages d'autres équipes. Il est mal documenté, et n'a pas été testé en dehors du studio.

Le second problème, à la fois plus grave et plus facile à résoudre, est que le logiciel est propriétaire, et donc inutilisable en l'état. En effet, son code source est accessible, mais aucune licence n'y est attachée, ce qui le met par défaut sous la protection du droit d'auteur. La publication simple ne confère pas aux utilisateurs potentiels les droits attachés au logiciel libre (cf. section II.1.1). Aucune bonne raison ne motivait ce choix de ne pas utiliser de licence, c'était un simple oubli de la part de l'auteur, dont je me suis aperçu à la rédaction de cette thèse. Mais cet oubli illustre que le choix de publication par défaut, qui est de ne pas choisir, ne permet presque aucune utilisation du programme.

III.2.2 Layout

Le layout (cf. section I.1.2.1) est une étape cruciale dans la production d'un film d'animation 3D, puisque c'est le premier passage à la 3D à proprement parler, après l'animatique. C'est à cette étape que l'on prend une grande partie des décisions de mise en scène, et que l'on crée des scènes 3D qui se retrouveront en aval dans le pipeline*.

Comme nous l'avons vu dans la section précédente, l'architecture du projet peut déjà être envisagée ou même établie à la création du storyboard et de l'animatique. C'est cependant au layout qu'elle est le plus souvent créée. C'est là que sont fabriqués les premiers assets*, les scènes 3D, les fichiers de montage et les premiers rendus. Il faut donc que dès cette étape le pipeline soit établi, ou au moins solidement ébauché, et que les plans* de layout puissent être réutilisés. Cela implique que le programme utilisé pour mettre en place les caméras, les décors et animations provisoires, soit compatible avec les départements suivants.

L'étape du layout est différente de celle de l'animation dans la qualité des animations, et dans la finition générale des plans, mais elle utilise techniquement les mêmes outils. Il n'existe donc pas de programme spécifiquement dédié au layout, et on utilise en général le même que pour l'animation. Le logiciel de création 3D* libre le plus utilisé est Blender, y compris pour le layout. Il a toutes les fonctionnalités nécessaires pour mettre en place le layout : placement et animation de caméras, modélisation, rendu rapide (playblast).

III.2.2.1 Outils spécifiques pour le layout des Fées spéciales

Comme à toutes les étapes de la création 3D, des outils spécifiques doivent souvent être conçus pour traiter des problématiques propres à un studio ou à un projet précis. Pour développer la manière de fonctionner du studio concernant le développement et la publication d'outils open source, j'exposerai deux outils développés en interne pour répondre à des problèmes très spécifiques de layout, et diffusés ensuite à l'extérieur.

Il faut rappeler que le pipeline développé par la société Les Fées spéciales pour certaines de ses productions a pour particularité l'utilisation de pantins et de décors

plans* dans un espace 3D. Cela permet de bénéficier de la facilité de conception et de mise en place de la 2D, et de la puissance de mise en place de la 3D. Les décors en question sont plans, comme les décors de théâtre (figure III.2.1). Chaque niveau de décor est peint sur un calque à part dans un logiciel de dessin, et ils sont ensuite importés dans la scène 3D. Ce processus est beaucoup plus rapide que la création de décors intégralement modélisés en 3D, mais offre en contrepartie des possibilités de mise en scène bien plus limitées, puisque la caméra ne peut pas se déplacer comme bon lui semble dans la scène, mais doit rester parallèle au décor. En fait, ce procédé est directement inspiré du banc-titre et de la caméra multi-plans*. On le retrouve tel quel dans les logiciels d'animation 2D, mais on peut obtenir un effet similaire dans un espace 3D en y plaçant des plans avec des textures.

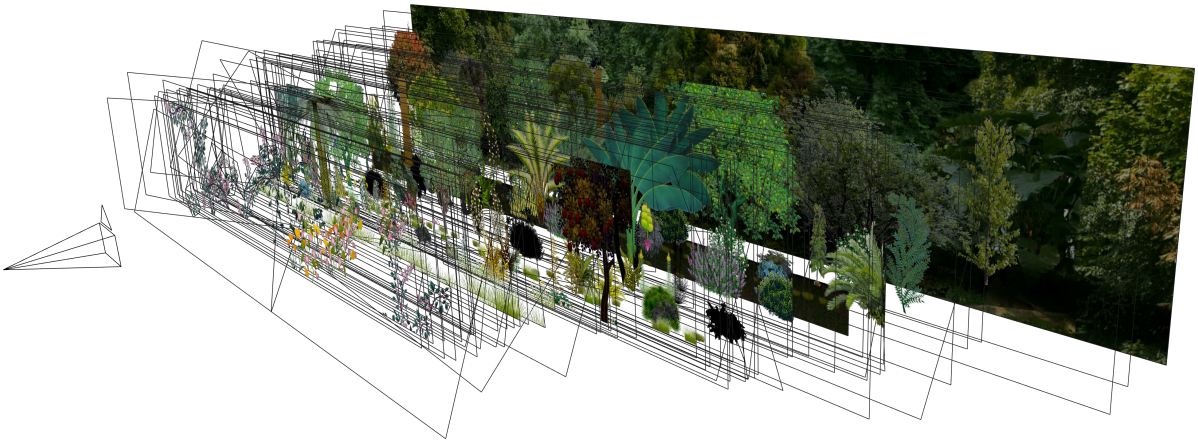


FIGURE III.2.1 – Plan du film *Dilili à Paris*. Chaque rectangle noir constitue un plan du décor.

Du fait de la spécificité de ce pipeline, les outils exposés ne sont donc pas utiles à toutes les productions d'animation 3D, mais le processus de développement et de publication n'en est pas moins révélateur.

Camera planes

Le premier de ces outils a été co-développé par le directeur technique Flavio Perez et moi, et baptisé *Camera planes*[6]. Il a pour but de faciliter la mise en place des décors en 3D. Sans notre outil, cette mise en place serait inutilement difficile car il faudrait importer et placer les décors dans la scène 3D, pour qu'ils apparaissent non seulement à la bonne taille dans la vue de la caméra, mais que les différentes couches soient placées à la bonne distance et que les éléments animés puissent y être insérés. Sachant qu'un décor de film peut être composé de dizaines, voire de centaines de couches devant l'action (*overlay*) et derrière (*underlay*), il est infaisable de le faire de cette manière.

Pour simplifier cette mise en place, cet outil expose une interface à l'utilisateur, permettant de choisir les fichiers de décor, et de les placer automatiquement dans la pyramide formée par la vue de la caméra, dans le même ordre que les calques du logiciel de dessin. L'idée est donc de retrouver strictement la même disposition. Les plans seront ensuite espacés dans la profondeur, comme un décorateur le ferait sur une scène de théâtre.

Il est ainsi facile de séparer la conception des décors de celle de leur mise en scène dans l'espace 3D. Une fois les décors placés dans la scène, la caméra peut être animée afin d'obtenir des mouvements similaires à ceux d'une caméra multi-plans*.

Rig de caméra 2D

Un autre outil a été créé pour faciliter la mise en scène dans un décor plan. Il s'agit d'un rig de caméra 2D, c'est-à-dire un objet permettant de manipuler la caméra de manière non seulement plus intuitive, mais également bien plus précise que celle qu'offrent par défaut les logiciels de création 3D*. En effet, les contrôles d'une caméra 3D sont logiquement adaptés à un layout en 3D : animation de la position, de l'orientation, de la focale, de la profondeur de champ, etc. Pour animer la caméra, il faut donc modifier ces paramètres un par un et les animer : les faire varier en fonction du temps. Mais pour

6. Notons ici que le nom de l'outil est à la fois en anglais, pour pouvoir plus facilement être diffusé auprès d'une communauté internationale, et peu imaginatif, témoignant du peu de soin apporté au choix du nom des outils lorsqu'on est en pleine production.

un film dans des décors plats, ces contrôles ne sont pas intuitifs du tout : il est plus logique de décider du cadre en déplaçant les coins de la caméra, comme on le ferait en plaçant un cadre autour d'une photographie. C'est ce que permet cet outil.

Les contrôles standards de caméra sus-mentionnés suffisent pour pouvoir mettre en œuvre cette logique de cadrage différente. Ainsi, grâce à la souplesse de l'API* du programme, nous avons compensé le manque de fonctionnalité en développant un outil spécifique. Il ne modifie pas le modèle de caméra, mais crée une série de contrôles simplifiés pour l'utilisateur. C'est donc un niveau d'abstraction supplémentaire qui ne change pas le fonctionnement du programme de base, mais y ajoute une couche pour cacher une partie de sa complexité.

III.2.2.2 Publication sous licence libre

Comme souvent, ces outils partent du constat d'une fonctionnalité manquante, et de la capacité interne de concevoir et développer cette fonctionnalité. Ces deux outils ont été publiés sous licence libre GPL (cf. section II.1.3). Ils sont donc disponibles à tous pour les utiliser et les adapter à leurs usages, et bien qu'ils soient restés assez confidentiels, sans doute par manque de communication de la part de la société, ils n'en ont pas moins reçu un bon accueil de la communauté.

Une telle publication peut soulever des questions pour les studios, dans la mesure où elle comporte des contraintes supplémentaires pour le développement des outils. Ils devront aussi être maintenus pour rester utiles à tous et ne pas devenir obsolètes. Ces problématiques sont abordées plus en détail dans la section II.3.3.2.

Chapitre III.3

Fabrication

Introduction

Les différentes étapes de la fabrication (cf. section I.1.2.2) sont la suite logique du layout, et on utilise généralement les mêmes outils pour le layout et pour l'animation. C'est à ce stade que le plus gros du film est créé par les graphistes, et il est indispensable, peut-être plus encore qu'aux autres stades (cf. chapitre III.2) d'avoir des outils performants, et un pipeline* solide. Ces outils dépendent du type d'œuvre, mais pour les projets sur lesquels j'ai travaillé, les logiciels Krita et Blender ont été au cœur des processus de fabrication.

Ce chapitre explorera les processus mis en œuvre dans une production d'animation utilisant le logiciel libre, pour permettre aux graphistes de travailler efficacement. Je me concentrerai d'abord sur le rigging (cf. section I.1.2.2) et l'animation (cf. section I.1.2.2), simplement du fait que j'ai passé presque un an à développer un système de rig automatique et d'animation dans Blender, puis j'aborderai les moteurs de rendu (cf. section I.1.2.2) libres utilisés en production, et les programmes de compositing (cf. section I.1.2.3).

III.3.1 Rigging et animation de personnages : les pantins

III.3.1.1 Initiation et historique du projet

Le rigging et l'animation sont deux des étapes les plus complexes et les plus longues à mettre en œuvre sur une production d'animation. Le rigging, parce qu'il demande des connaissances techniques pointues, et l'animation parce qu'elle est simplement lente : les animateurs font en général entre deux et dix secondes d'animation par jour, selon le niveau de qualité et le médium*. Pour pallier en partie cette complexité, les Fées spéciales ont choisi de développer des outils permettant d'utiliser une technique d'animation particulière, celle des pantins en papier découpé, parfois appelée « *cutout* ». J'expliquerai les intérêts de cette technique pour une production de film, et les difficultés de mise en œuvre au sein d'un pipeline* libre où les outils disponibles ne répondaient pas à nos besoins.

Dès la création du studio Les Fées spéciales, le choix a été fait d'utiliser cette technique pour ses propres productions, et si possible pour les prestations également. La technique tire son nom du médium utilisé, développée elle aussi au cours du XX^{e} siècle, par des auteurs comme Lotte Reininger. La raison première de ce choix par la société est la collaboration, pour le premier projet du studio, avec le réalisateur Michel Ocelot, qui est adepte de la technique et l'a utilisée dès ses débuts sur des films et séries comme *Les Trois Inventeurs* et *La Princesse sensible*. Plus récemment, il l'a aussi utilisée, en version numérique, pour des longs métrages, dont *Les Contes de la nuit*.

Dans sa variante historique, la technique consiste à dessiner un personnage sur du papier, puis à en découper les parties du corps et les accessoires, qui seront articulés à l'aide de ficelles ou d'attaches parisiennes. Les personnages sont ensuite placés sur un banc-titre [1] et animés image par image. Le personnage est filmé, soit à contre-jour et apparaît ainsi en silhouette noire sur blanc, soit éclairée du dessus, laissant apparaître

1. Appareil permettant de filmer l'animation image par image, consistant en une table lumineuse filmée par une caméra mobile accrochée au-dessus.

les détails, les ombres et la texture du papier.

Tout comme dans les autres techniques de l'animation, l'outil numérique remplace rapidement les objets physiques pour faciliter la création et l'animation. En utilisant l'ordinateur, on peut accélérer et automatiser beaucoup d'étapes, tout en conservant la même latitude expressive et les mêmes principes d'animation que sur papier.

Sur le film *Dilili à Paris*, l'animation principale des protagonistes du film était faite en 3D, mais l'auteur souhaitait faire apparaître des foules de personnages dans certaines séquences* pour donner de la vie à l'environnement du film. La création en 3D de centaines de figurants satisfaisant les ambitions artistiques du réalisateur aurait été bien trop chère pour le budget du film. Il a donc été décidé de reprendre la technique des pantins utilisée par Michel Ocelot sur ses projets précédents. C'est en partie pour cette raison que les Fées spéciales ont été contactées : pour pouvoir prendre en charge le travail de conception et d'animation des pantins, ainsi que le layout.

La production du long-métrage *Dilili à Paris* a été donc été l'occasion de développer des outils spécifiques pour l'animation de pantins. Il existait déjà des solutions, mais elles ne pouvaient pas être intégrées au pipeline de fabrication 3D du film (cf. section III.3.1.3).

Inspirations

Lors du précédent long-métrage de Michel Ocelot, *Les Contes de la nuit*, dont l'esthétique reprenait celle des films en papier découpé mais dont la fabrication était numérique, un système de création de personnages assisté avait été conçu par l'animateur Jean-Claude Charles. Il permettait de découper et rigger les personnages rapidement, ce qui est crucial pour une œuvre mettant en scène de nombreux personnages. Cet outil fonctionnait dans le logiciel utilisé pour l'animation de ce film, Autodesk Maya (figure III.3.1).

Pour la fabrication de *Dilili*, les Fées spéciales avaient pris la décision dès la création d'utiliser le logiciel de création 3D* Blender. Il était donc impossible, même dans l'hypothèse où les outils Maya avaient été disponibles, de les utiliser directement dans notre chaîne de fabrication fondée sur Blender, les API* de Blender et de Maya étant tout à

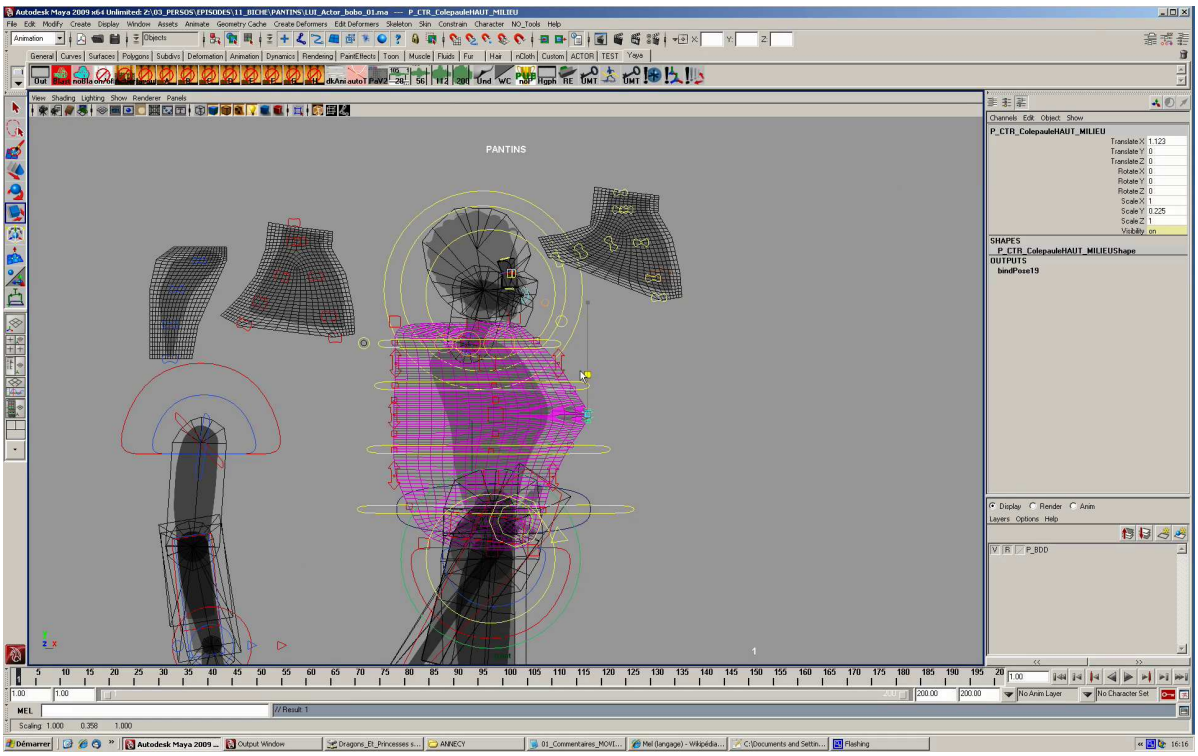


FIGURE III.3.1 – Interface développée dans Maya pour *Les Contes de la nuit*.

fait incompatibles. Il fallait donc écrire de nouveaux outils répondant à un cahier des charges similaire. Pour nous aider, Jean-Claude, qui a aussi travaillé sur *Dilili* en tant que premier assistant réalisateur, a transmis au début de la fabrication (mars 2016) un ensemble de vidéos illustrant le fonctionnement du système de rigging automatique, et les fonctionnalités d'animation des pantins eux-mêmes (figure III.3.1). N'ayant que peu d'expérience d'animation, ces vidéos ont été une inspiration précieuse pour la conception et le développement de ce qui allait devenir mon outil de pantins. De même, j'ai analysé des rigs de pantins existants, comme celui de Pepper, utilisant le dessin d'un personnage de bande dessinée créé par David Revoy [2], et riggé par l'animateur et rigger russe Nikolai Mamashev. Une autre inspiration a été le travail de rig mené sur le personnage de Tomoe Gozen et son cheval, dessinés par mon collègue Eric Serre et riggés par le graphiste Philippe Giffard. Tomoe, riggée entièrement à la main, est dotée de systèmes

2. Cette bande dessinée en ligne est une des rares à être diffusée sous une licence de libre diffusion (Creative Commons Attribution 4.0, cf. section II.1.4), permettant ainsi à n'importe quel créateur d'utiliser les personnages ou même directement les textes et images pour créer des œuvres dérivées.

très proches de ce que je souhaitais obtenir avec l'auto-rig.

III.3.1.2 Contraintes

La création d'un système d'auto-rig n'est pas triviale, et il faut tenir compte avant de commencer des contraintes du projet et de la structure, concernant les possibilités artistiques, techniques, budgétaires, et de production.

FIGURE III.3.2 – Concept art de Michel Ocelot, avant le début de la fabrication.

Délais de fabrication

Le premier critère du système était que les pantins soient rapides à fabriquer. En réalisant des prototypes de rigs, j'ai estimé le temps de fabrication d'un pantin sans outil d'automatisation à une semaine, d'un bout à l'autre, dessin exclus. Il faut garder à l'esprit que des centaines de personnages devaient habiter les scènes du film, et que pour

la plupart, ils étaient équipés de rigs similaires, mais à chaque fois légèrement différents. Il était donc impératif d'automatiser la création. C'est un cas où l'automatisation ne permet pas seulement de faire gagner un peu de temps et d'efficacité à l'équipe de graphistes, mais, simplement, de terminer le film.

Une autre contrainte importante était celle du calendrier. En effet, j'ai commencé à travailler sur ce projet dès mon arrivée dans la structure, et la fabrication du film a démarré aussitôt. Je n'avais que très peu de battement pour arriver à un programme fonctionnel. Les premiers pantins du film ont donc été riggés par moi et d'autres graphistes, « à la main », sans automatisation du processus, en parallèle de la conception du prototype de rig complet, puis du développement des outils de rigging.

Fidélité du design

Une des demandes artistiques principales était la fidélité du personnage au design original décidé par l'auteur. En effet, ce dernier souhaitait retrouver tels quels dans le film les personnages qu'il avait dessinés (figures III.3.2 et III.3.3). Cela signifie que le processus de fabrication des pantins devait s'appuyer sur les images numériques envoyées par Michel Ocelot, et qu'un contrôle rigoureux de la fidélité au dessin original soit opéré à chaque étape [3].

Les pantins sont considérés comme des figurants. À ce titre, ils apparaissent en général en arrière-plan et sont cadrés plus large que les personnages principaux. Ils sont moins détaillés, et ne doivent pas offrir des possibilités d'animation aussi complètes que leurs homologues 3D. Néanmoins, ils doivent avoir une gamme de mouvements suffisant à donner l'illusion de leur présence parisienne, à leur offrir une individualité, en un mot à proprement les animer. Techniquement, plusieurs besoins ont émergé de mon étude des rigs de pantins de Jean-Claude, et de conversations avec l'animateur des pantins, Léo Silly-Pelissier.

3. Voir aussi l'annexe C et le point de vue d'un artiste sur la polyvalence artistique de la technique des pantins.

FIGURE III.3.3 – Une image du layout correspondant au concept précédent.

Systèmes d'animation

Les rigs devaient être dotés de contrôleurs (figure III.3.4) relativement standards : membres en FK et IK* avec bascule ; *foot roll* ; plusieurs articulations dans le torse ; tête et membres détachables pour bouger le point d'articulation ; membres étirables pour donner l'illusion d'un dessin en raccourci ; variations colorées des éléments (visages, habillements) pour augmenter la variété d'un personnage sans le redessiner et rigger ; variations animables pour différents éléments (par exemple, animation de main pour effectuer une action, ou animation des yeux pour le clignement). Les personnages devaient de plus pouvoir se retourner, c'est-à-dire être tournés, soit vers la droite, soit vers la gauche. Il devait être facile d'intégrer des accessoires à un personnage donné, comme cannes, bouquets, parapluies, chapeaux, volets de jupes, landaus, etc. Et bien sûr, il n'y avait pas deux morphologies identiques dans les dessins de Michel Ocelot.

Les rigs de pantins numériques peuvent être conçus pour permettre d'animer les personnages de manière fine, avec des déformations pour créer l'illusion de rotation ou de raccourci. Dans la mesure où les pantins du film étaient des figurants assez peu détaillés, que le style d'animation souhaité par le réalisateur ne requiérait pas ce type

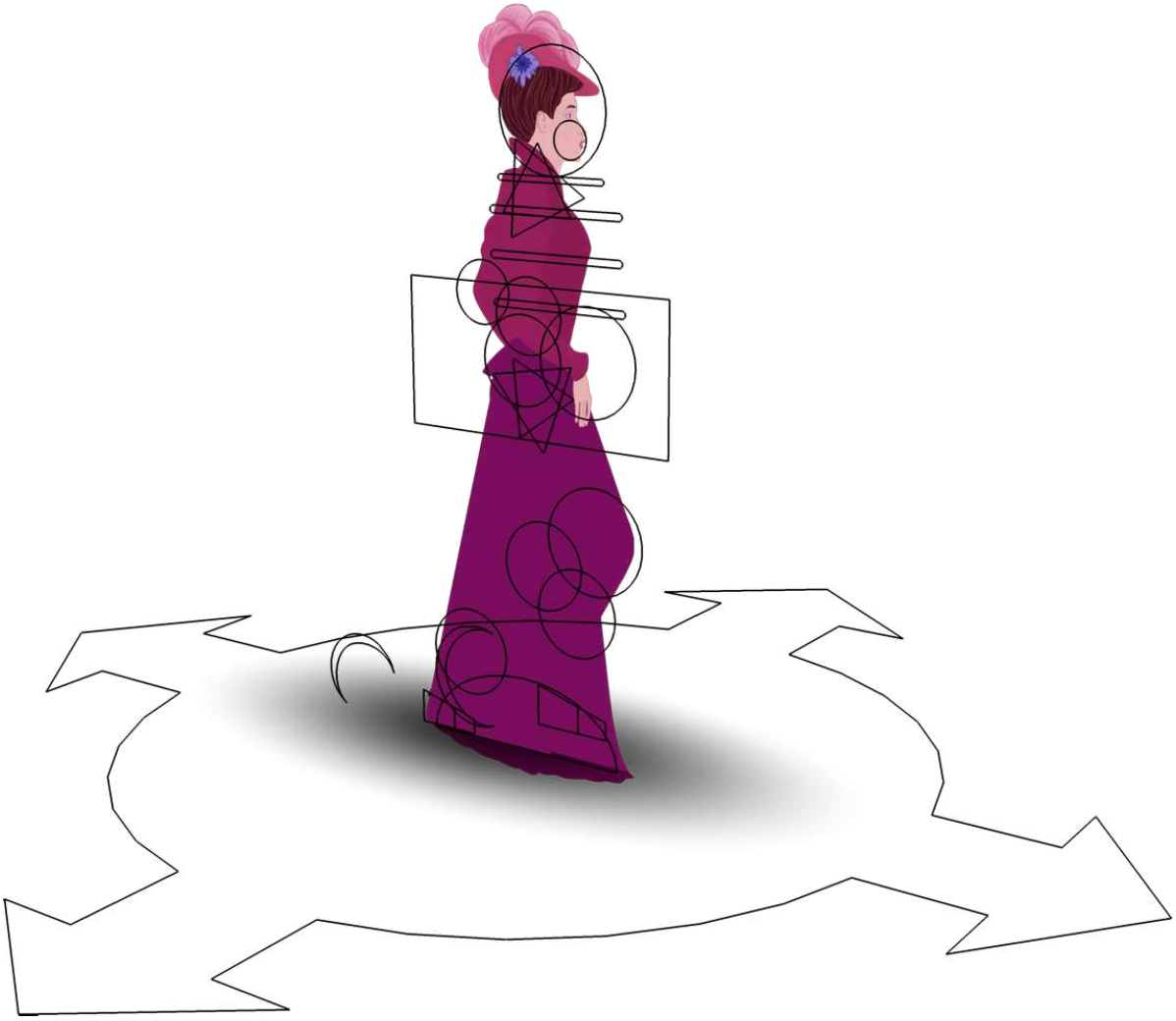


FIGURE III.3.4 – Rig de personnage pantin dans *Dilili à Paris*. Les formes géométriques sont les contrôleurs accessibles aux animateurs, et la majeure partie de la complexité du rig est cachée.

de contrôles, et que leur mise en œuvre aurait complexifié le travail des riggers comme des animateurs, je n'ai pas créé de système de déformation pour les pantins de ce film.

Outre les considérations de rig et d'animation, des contraintes de pipeline propres à la production sont apparues. En effet, comme expliqué dans la présentation du projet (section III.1.1), le layout et l'animation des foules étaient faits par les Fées spéciales, puis exploités par le studio Mac Guff. Je reviendrai plus en détail sur ces questions dans le section III.5.2, mais il faut retenir pour le moment qu'elles ont imposé des contraintes supplémentaires pour le système de pantins.

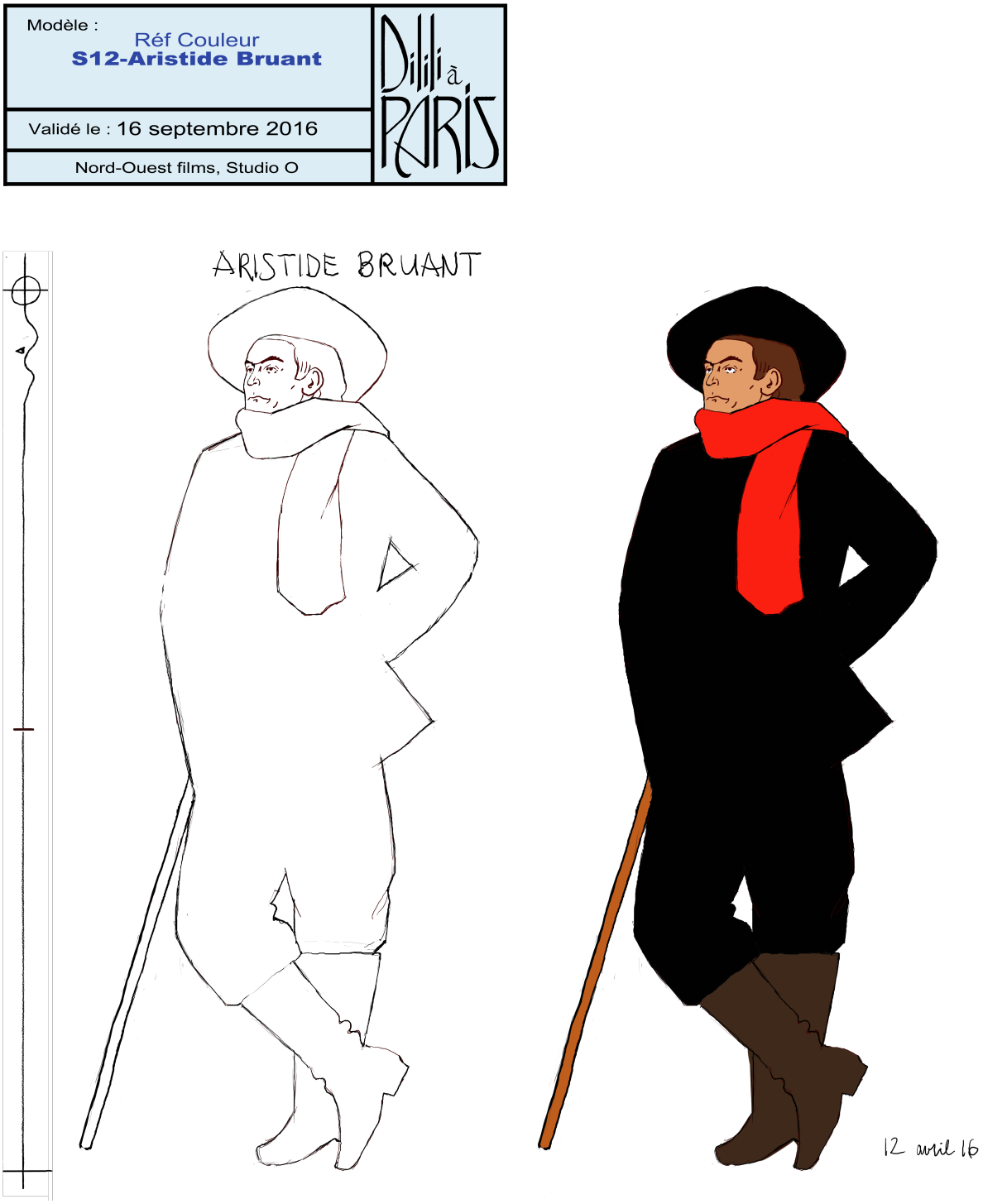


FIGURE III.3.5 – Aristide Bruant dessiné par Michel Ocelot. Ici, la couleur du personnage est également fournie, mais beaucoup de pantins ont été colorisés par l'équipe pantins des Fées spéciale. On voit en haut à gauche le cartouche de Michel Ocelot, et à gauche la toise permettant de dessiner les personnages à l'échelle.

Passage du dessin à d'animation

Concernant la manière de créer un personnage, il fallait d'abord créer une passerelle entre le logiciel de dessin (Photoshop ou Krita) et le logiciel d'animation 3D (Blender). Chaque membre [4] étant dessiné sur un calque à part (figures III.3.5 et III.3.6), ce dessin et sa décomposition en calques devaient se retrouver à l'identique dans Blender. Les points d'articulations devaient aussi pouvoir être placés dans le logiciel de dessin pour permettre à l'équipe Pantins de concevoir la dynamique du personnage en même temps qu'elle le dessinait.

Une fois le dessin du pantin correctement importé dans la scène Blender, il fallait que la création du rig soit simple et prévisible. Dans l'idéal, le personnage aurait été riggé à l'appui d'un seul bouton, mais un tel système aurait requis, soit trop de contraintes sur le rig, soit d'être beaucoup trop avancé pour être mis en œuvre. Le système que j'ai conçu est donc plutôt un assistant au rig qu'un autorig complètement autonome. Néanmoins, il réduit considérablement les étapes de création et assure une cohérence entre les différents pantins, grâce à une nomenclature standard des membres.

III.3.1.3 Solutions existantes

Après plusieurs semaines d'étude et de prototypage des systèmes de rig, les contraintes n'étaient pas formalisées dans un cahier des charges technique, mais elles étaient assez clairement définies pour que j'engage le développement d'outils. J'ai commencé par rechercher des outils existants pour créer des pantins. Il existe de nombreux logiciels propriétaires, et quelques logiciels libres. Je n'en ferai pas ici une liste exhaustive, mais j'en mentionnerai deux qui ont été considérés, et les raisons qui m'ont poussé à développer un nouveau système.

D'abord, j'avais connaissance depuis quelques années de DUIK, un plug-in* de pantins pour Adobe After Effects dont l'efficacité a été démontrée par son utilisation sur de

4. On entendra par « membre » un élément mobile du rig, correspondant à un calque dans le logiciel de dessin, à une plaque en 3D, et à un os dans le rig. Ainsi, un avant-bras peut être membre, tout comme un orteil, une jupe ou un parapluie.

nombreuses séries animées et autres œuvres [5]. Je l'avais vu à l'œuvre lors de la fabrication d'un court-métrage d'Eric Serre, *Herakles, aux origines de la Crau*, en 2014. Il avait servi à rigger tous les personnages, y compris des chevaux. Ce programme, malgré sa grande qualité et bien qu'il soit libre [6] et développé en France par la SCOP Rainbox, a cependant été éliminé, pour des raisons de licence et d'utilisation. En effet, si le plug-in est lui-même libre, il demande à être exécuté dans un environnement non-libre, dans After Effects sous Windows ou Mac OS, or les Fées spéciales utilisent GNU/Linux*. De plus, le pipeline* requérant d'être en 3D et fondé sur Blender pour pouvoir être transmis à Mac Guff, une solution After Effects ne convenait pas.

Dans Blender, l'add-on* COA Tools (Cutout Animation Tools) [44], développé par Andreas Esau, existait et répondait à certaines contraintes de la production. C'est un outil permettant d'accélérer la création de rigs, en fournissant par exemple des solutions pour découper les plaques, dessiner rapidement le squelette et opérer le parentage et le skinning* plus facilement qu'avec les fonctions standards de Blender. Il offre également un système d'échange avec les applications de dessin Adobe Photoshop et The GIMP pour pouvoir échanger les calques entre le logiciel de dessin et le logiciel de création 3D*, ainsi que leur position et leur nomenclature. Cependant, cet add-on, très utile pour des graphistes seuls et expérimentés, n'était pas adapté à l'échelle de la production (pour rappel, des centaines de personnages similaires), et à l'équipe de rig, dont nous savions qu'elle serait en partie débutante et pour laquelle le processus devrait être simplifié au maximum. De plus, il ne permet pas de développer un rig qui ait à la fois des contrôles avancés, et soit facilement reproductible.

III.3.1.4 Développement de l'outil de rig de pantins

Même s'il existait des solutions répondant à certains de mes critères, aucune n'était entièrement satisfaisante. Il fallait donc, soit partir de zéro et concevoir une solution originale, soit modifier une solution existante. Connaissant un système de rig modulaire

5. L'entretien présenté en annexe C présente un autre point de vue sur la facilité d'utilisation de DUIK.

6. Il est disponible sous licence GNU GPL, cf section II.1.3.1.

intégré à Blender, j'ai choisi la deuxième solution.

J'ai utilisé et détourné l'add-on Rigify pour créer le système de génération de rigs. Le système d'autorig est composé de plusieurs sous-systèmes permettant la fabrication des pantins, de leur découpage à leur rendu. Je décrirai maintenant le fonctionnement de ces sous-systèmes.

Dessin des plaques

Comme expliqué en introduction de ce chapitre, l'intérêt artistique de cette technique est de retrouver à l'identique le dessin de concept dans le personnage animé définitif. Pour cela, le dessinateur peut utiliser des outils physiques traditionnels, auquel cas le dessin sera numérisé avant d'être retraité numériquement, ou directement un logiciel de dessin. Dans tous les cas, le concepteur n'est pas nécessairement l'animateur, ni le rigger du personnage. Ainsi, il n'est pas nécessaire d'avoir de quelconques connaissances techniques pour que son personnage se retrouve à l'identique dans le film. Pour *Dilili* par exemple, l'auteur Michel Ocelot a dessiné la quasi-totalité des figurants, sans jamais utiliser Blender.

Entre le dessin du personnage, sur papier ou numérique, et le début du rig dans Blender, une étape intermédiaire dans un logiciel de dessin est nécessaire. Les pantins étant animés, les différentes parties de leurs corps doivent nécessairement bouger l'une par rapport à l'autre. Il faut donc les « découper » en éléments, qui seront reliés entre eux au moment du rig. Dans la technique traditionnelle, cette découpe est littérale : on coupe chaque partie du corps avec des ciseaux, et on les attache ensemble avec de la ficelle ou des attaches parisiennes. Dans la version numérique, on part du dessin et, pour chaque partie, on la sélectionne et on la place dans un calque à part en prenant soin que les zones d'articulations soient couvertes par les deux parties. Par exemple, au niveau du genou, où la cuisse et la jambe se rejoignent, on trace un cercle, et on découpe autour sur les deux parties.

Sur *Dilili*, les dessins étaient fournis au trait, c'est-à-dire sans couleur. Ils étaient donc d'abord colorisés dans Krita avant d'être découpés. Pour le système d'autorig, j'ai

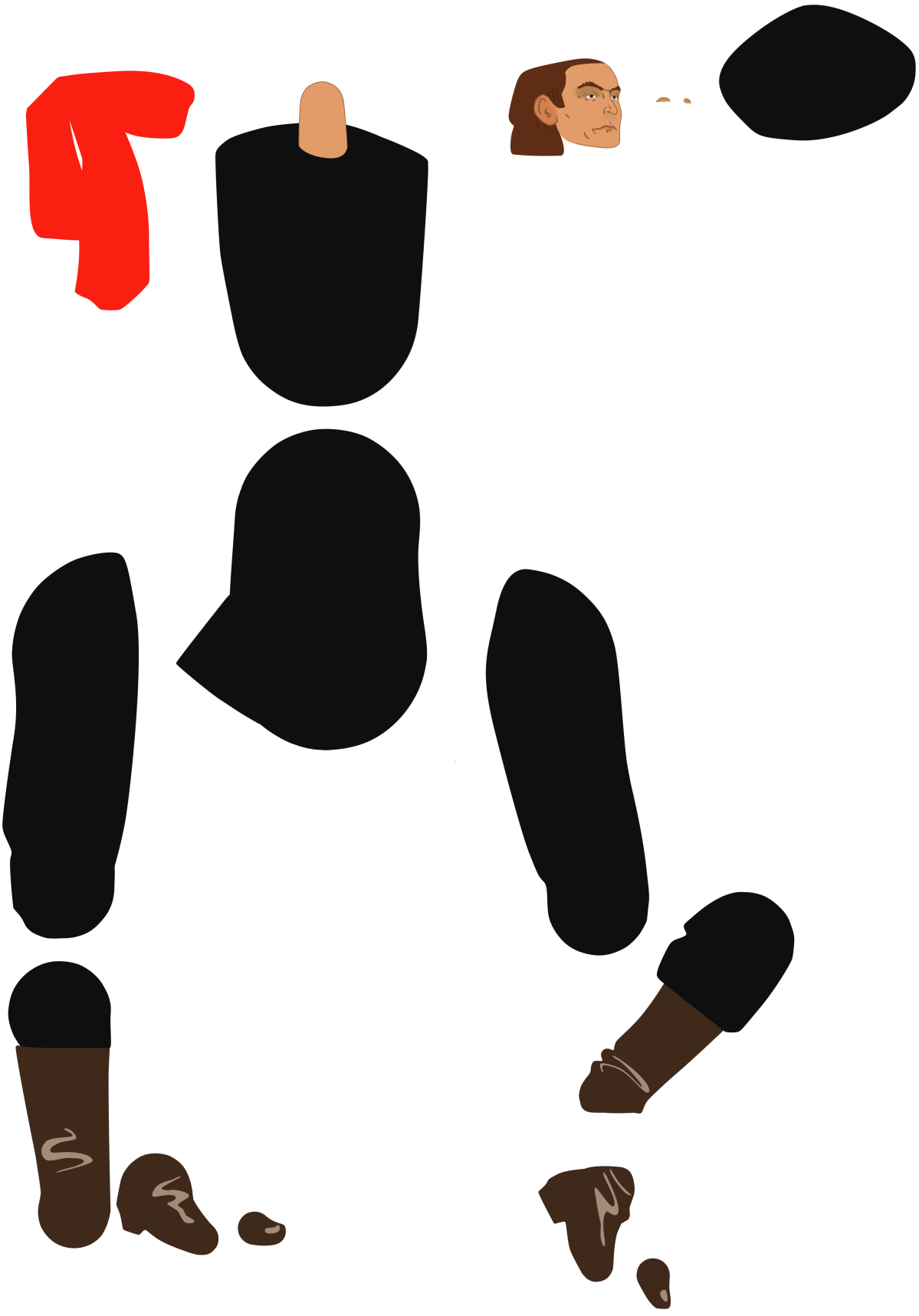


FIGURE III.3.6 – Vue éclatée d'Aristide Bruant, découpé par l'équipe pantins.

conçu un modèle de fichier contenant déjà tous les calques vides, correctement nommés et organisés selon une nomenclature figée. L'équipe des pantins pouvait donc copier chaque partie du dessin original, et la coller dans le calque correspondant. Cette nomenclature standardisée, en contraignant les graphistes à une forme toujours identique, a plusieurs avantages. Elle leur évite de devoir organiser et nommer tous les calques pour chaque pantin. Elle permet aussi, dans le logiciel de 3D, d'identifier automatiquement les calques pour les parenter correctement, comme nous le verrons plus loin.

Dans les cas où plusieurs parties du corps se superposent, par exemple si la main est dessinée le long du corps, il faut faire en sorte qu'elle ne se retrouve que dans le calque correspondant, et donc effacer la main du calque du torse.

Import des plaques dans le logiciel de création 3D

Une fois le pantin dessiné et découpé, chaque partie du corps indépendante se trouve dans un calque dans le fichier de dessin. Avant de pouvoir commencer le rig dans Blender, il faut pouvoir l'y importer. Cette partie du processus a été assez rapide à mettre en œuvre, car elle s'appuie sur un add-on* existant. Développé par Jasper van Nieuwenhuizen [55], un importeur de fichiers Photoshop permet d'utiliser ce format et de retrouver dans la scène 3D les calques, avec leur ordre, leur hiérarchie (groupes), et leur position. Certaines informations, comme les modes de fusion et options de calques sont perdues, mais les données de pixel sont suffisantes pour notre besoin.

L'add-on* est publié par son auteur sous licence GPLv2. J'ai donc pu y apporter les modifications nécessaires pour l'usage du projet (correction de bugs, préréglages, et différentes options concernant la nomenclature et la création des matériaux). Ces modifications ont depuis été réintégrées par l'auteur dans le code principal.

Il est à noter que le format utilisé est celui de Photoshop (PSD), mais que les pantins sont découpés sous Krita. En effet, quoi que le format PSD ne soit pas normalisé, une spécification assez précise a été publiée par Adobe [59], et les développeurs de Krita l'ont implémentée dans leur logiciel. Il est donc utilisé dans la chaîne de fabrication plutôt que le format natif de Krita, simplement à cause de l'outil d'import de pantins dans Blender.

Ce type de fichier est complexe, et du fait de la disponibilité de la spécification du format PSD, des bibliothèques* Python sont disponibles pour le lire et l'écrire. C'est sur une de ces bibliothèques que s'appuie l'add-on de Jasper [7]. Un analyseur des fichiers Krita pour Python aurait été trop coûteux à mettre en œuvre à l'époque. Depuis, Krita intègre la possibilité de scripter le programme et une solution d'échange direct Krita-Blender est envisagée, mais n'a pas encore été développée, le bénéfice étant trop mineur par rapport au temps de développement.

Rigify

Rigify est un système d'auto-rig modulaire et extensible. C'est un add-on* officiel, c'est-à-dire maintenu par les développeurs de Blender, et distribué avec. Il est utilisé pour créer rapidement des personnages 3D. On construit un rig à partir de « briques » ayant chacune un jeu de fonctionnalités propres. Par exemple, on peut ajouter dans son rig un module *torse*, des modules *bras*, des *jambes*, etc.

La création d'un rig avec Rigify se fait en deux étapes. Les différents composants sont positionnés de façon à correspondre à la morphologie du personnage, puis le rig est « généré », c'est-à-dire que les mécanismes et contrôleurs sont créés par le programme d'après cette morphologie (figure III.3.7). L'intérêt est de séparer la forme du personnage de ses fonctionnalités. Par exemple, on pourra créer le rig de deux personnages ayant des mécanismes identiques, mais des morphologies très différentes. Ou bien, on pourra ajouter facilement des membres supplémentaires au personnage, sans devoir rigger ces membres individuellement. Un rig pouvant être constitué de centaines d'éléments logiques (os, contraintes, *drivers*) devant correspondre à une nomenclature rigoureuse, et chacun de ces éléments prenant un temps non négligeable à mettre en place, on voit la rapidité et la flexibilité que peut offrir un tel système.

Les possibilités de l'add-on correspondaient assez bien aux contraintes de *Dilili*, mais le système n'était pas du tout conçu pour des personnages de type pantin. Quoique flexible pour faire des personnages 3D, Rigify ne permet pas par défaut de créer n'importe

7. La bibliothèque Python utilisée est psd-tools [57].

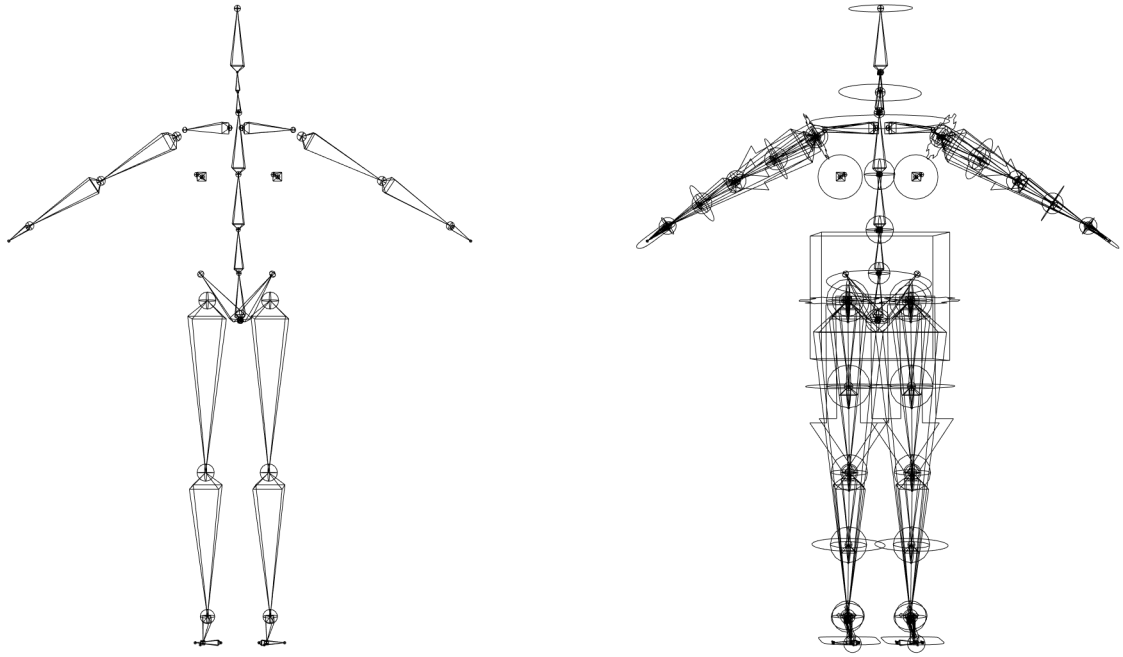


FIGURE III.3.7 – Squelette généré par l'add-on Rigify. À gauche, le squelette de base, avec 29 os. À droite, le squelette généré automatiqument, avec 217 os et des centaines de contraintes et de drivers.

quel rig, la génération des personnages étant prévue en dur dans le code, c'est-à-dire impossible à modifier pour l'utilisateur lambda. Pour ajouter des fonctionnalités, il fallait modifier l'add-on. Aussi, j'ai dû étudier le code pour déterminer comment y ajouter des modules supplémentaires, qui utiliseraient les mêmes principes, sans nécessairement être compatibles avec le système de base, car les deux utilisations diffèrent radicalement.

En tant que système modulaire, Rigify est constitué de trois types de scripts* fondamentaux :

— le système de génération, qui constitue le cœur du programme,

— les différents modules de génération automatique, ou ‹rigs› [8],

— des modèles de personnages prédéfinis constitués de ‹rigs› : les ‹metarigs›.

Mon idée était de garder le cœur tel qu'il existait (en corrigeant et signalant éventuellement les bugs découverts), et d'ajouter des modules de ‹rigs› et de ‹metarigs› adaptés

8. Terminologie de Rigify. Les mots propres à cet outil et dont la définition diffère de l'habitude seront par la suite entourés de ‹guillemets simples›.

aux besoins de *Dilili*. Ainsi, ma version exploiterait les fonctions de génération, en proposant des modèles de personnages très différents. Cette version n'est que partiellement compatible avec l'add-on original, car les nouveaux types de ‹rigs› ne peuvent pas être mélangés avec les types originaux au sein d'un même personnage.

Ordre des plaques dans l'espace

FIGURE III.3.8 – Si l'ordre des plaques dans la profondeur n'est pas défini, il apparaît des conflits.

Par défaut dans Rigify, les os sont positionnés dans l'espace, avec une position et une orientation en trois dimensions. À l'inverse, les pantins, héritage du papier découpé, ne se déplacent qu'en plan, dans deux dimensions. En théorie, retirer une dimension simplifie la conception d'un personnage, puisque les membres ne peuvent plus tourner que selon un seul axe. Dans la pratique, d'autres considérations apparaissent. Lorsqu'on découpe des morceaux de papier et qu'on les lie ensemble, on décide nécessairement lequel passe devant l'autre. On peut aussi changer en cours d'animation. Par exemple, si un personnage danse la gigue, un pied restera au sol et l'autre passera alternativement devant et derrière la jambe fixe. Or, dans un rig en 2D, tous les os sont dans le même plan. Il faut donc pouvoir donner au moteur de rendu des instructions pour dessiner les membres dans le bon ordre, sans quoi on peut se retrouver avec un personnage de profil ayant le cou devant la tête, ou les deux bras devant le torse, dans une interprétation cubiste de la perspective, en plus d'erreurs graphiques (figure III.3.8).

Ce problème est résolu facilement dans les logiciels de dessin 2D, grâce aux calques.

Les calques forment une pile, et sont dessinés l'un après l'autre du bas au haut de la pile, comme des feuilles transparentes empilées sur une table lumineuse (d'où le nom approprié de calques). C'est le cas par exemple dans DUIK, où on peut décider de l'ordre des plaques en réordonnant les calques.

Dans un espace 3D comme celui de Blender en revanche, toutes les plaques sont par défaut dans le même plan. Cela crée des conflits d'affichage, le moteur de rendu ne sachant pas distinguer ce qui vient devant. Il faut donc légèrement décaler ces plaques dans la profondeur. Le premier instinct est de prendre chacun des objets-plaques, et de les décaler jusqu'à ce qu'ils soient tous sur un plan différent, dans le bon ordre. Seulement, cette approche pose rapidement problème dès que l'on veut que le pantin se retourne dans l'autre sens. Si l'on fait faire demi-tour au pantin, l'ordre des plaques est inversé ; des plaques qui devaient être derrière se retrouvent devant, et inversement. Les membres eux-mêmes sont inversés : le bras qui était devant se retrouve derrière.

FIGURE III.3.9 – Espacement exagéré des plaques dans un pantin. Les os sont éloignés les uns des autres, et les plaques qui leur sont parentées paraissent donc dans le bon ordre.

Il faut donc trouver une solution plus évoluée pour résoudre ce problème. Celle que j'ai mise en œuvre consiste à ce que ce décalage se fasse à l'intérieur du rig, avant le parentage de la plaque à l'os, mais après les mécanismes qui donnent à l'os sa position. J'ai donc décidé de décaler dans la profondeur les os de déformation. Une interface permet au rigger de choisir l'ordre des membres et

des plaques (figure III.3.9). Cet ordre est stocké dans une structure de donnée définie exprès, et les plaques sont ensuite décalées à l'aide de *drivers* mus par cette structure de données. D'une certaine manière, ce système assez complexe permet d'émuler dans Blender un système de calques propre aux logiciels de dessin, pour obtenir une interface intuitive, mais assez flexible pour que l'équipe pantin puisse changer l'ordre comme bon lui semble.

Parentage automatique

La notion de parentage, rapidement évoquée précédemment, désigne un lien qui unit deux entités dans une scène 3D (objets, os). C'est une relation dans une direction, où une entité enfant suit les transformations de son parent. Géométriquement, un parentage se fait en multipliant la matrice du parent par la matrice locale de l'enfant, pour obtenir la transformation définitive. Ce principe est omniprésent en 3D pour créer une hiérarchie dans une scène ou au sein d'un ensemble d'objets, comme un personnage.

Dans l'animation 3D, les personnages sont généralement déformables, et sont liés à leur squelette grâce à un système de skinning*. Dans le cas des pantins, la technique traditionnelle du papier découpé et articulé ne demande pas de déformation des membres. Le skinning n'est pas strictement nécessaire et un parentage simple peut suffire. Pour se faire, on dessine les articulations de manière à ce que les deux parties se superposent. Par exemple, on découpe le membre inférieur en deux : la cuisse et la jambe, qui se superposent au niveau du genou pour former un raccord invisible lors de la rotation.

Si toute la fabrication avait été faite dans un même programme (Blender), il aurait été envisageable de créer des déformations de plaques avec un système de skinning*, mais plusieurs obstacles ont empêché de le faire sur *Dilili*.

D'abord, s'il est relativement aisé de faire passer des transformations simples de Blender à Maya, les déformations étaient beaucoup plus complexes à transférer au début de 2016, c'est-à-dire à un moment où le format Alembic n'était pas encore intégré à Blender. Je décrirai les problématiques d'échange dans la section III.5.2, mais pour l'heure il faut savoir que les animations de personnages 3D layout sont justement exportées avec

des caches. On aurait pu utiliser un tel système pour les pantins, ce qui aurait donné la possibilité de déformer les membres. Mais les opérations de génération, d'export, de conversion de ces caches sont très coûteuses, et leur poids non négligeable, ce qui était un élément à considérer pour choisir comment échanger les données entre plusieurs sites.

Ensuite, le processus de skinning est long, difficile à maîtriser, et demandait des recherches supplémentaires pour permettre la déformation de plaques. Le processus est aussi difficile à automatiser, car des tests et retouches doivent être effectuées pour obtenir un skinning correct, ou alors développer des algorithmes spécifiques, ce pour quoi je n'avais ni les compétences, ni le temps.

Enfin, comme dit précédemment, le réalisateur souhaitait une esthétique proche des pantins découpés, ce que permettait très bien de se contenter d'un parentage simple. Nous avons donc décidé, sauf rares exceptions « hors pipe[line] [9] » de nous contenter de plaques indéformables.

L'opération de parentage incluse avec Blender assez simple, mais elle doit être répétée des dizaines de fois pour chaque personnage, pour les centaines de personnages du film. Pour faciliter ce travail, un outil spécifique a été conçu. Sachant que la nomenclature des os et plaques des pantins serait toujours similaire, il était logique de construire des modèles, à la fois dans Krita pour le découpage, et dans Blender pour le rigging. Dans Krita, un fichier contenant une hiérarchie de calques est établi, permettant de dessiner les membres d'une manière unifiée, à la fois dans le nommage des calques et dans l'ordre des plaques (cf. section III.3.1.4). Du côté de Blender, l'utilisation des modèles de rigs garantit une nomenclature standard — sauf modification par les graphistes.

Variations d'animation

Les dessins des pantins sont statiques. Contrairement à l'animation 3D ou au dessin animé, où la forme peut changer à chaque image, on ne peut pas faire prendre n'importe quelle position aux plaques. Par exemple, si la main a été dessinée ouverte, on pourra la bouger en déplaçant le bras, mais il est impossible de la faire se fermer.

9. C'est-à-dire imprévues dans le système automatique, et à la charge des graphistes.

Il a donc fallu trouver un moyen de permettre à une plaque donnée (par exemple, la main ou l'œil) de pouvoir prendre en cours d'animation plusieurs apparences différentes. Si on fait l'analogie avec l'animation *stop motion*, les mouvements généraux du corps correspondent au rigging, et les variations correspondent aux différentes parties interchangeables donnant une gamme de formes. On s'en sert pour faire cligner les yeux des personnages, ou pour la synchronisation labiale.

Dans Blender, ces variations prennent la forme de multiples plaques superposées et parentées au même os. Ainsi, si la main de notre personnage doit avoir trois positions différentes, mettons : ouverte, fermée et pouce tendu, chacune de ces positions est dessinée dans Krita. Au moment de l'import, le pantin se retrouve avec trois mains superposées. Un script* spécialisé permet de considérer ces trois mains comme des variations d'un même membre, et de n'en afficher qu'une à la fois. Les autres sont cachées automatiquement. Cela implique, d'une part, que ce système de variations soit exposé à l'animateur à travers une interface simple et claire, et d'autre part que la gestion de l'affichage des différentes plaques soit automatique.

J'ai implémenté ce système avec un système de *drivers* sur la visibilité des plaques. Lorsqu'on sélectionne le contrôleur de la main, une propriété s'affiche dans l'interface, qui peut prendre une valeur parmi le nombre de variations (dans notre exemple, 0, 1 ou 2). Lorsqu'on change cette valeur, la visibilité des plaques est mise à jour par le driver. Si l'animateur souhaite sélectionner la main fermée, dont l'indice est 1, il met la propriété à 1 et a l'illusion que la main a instantanément changé de forme.

L'intérêt de ce processus est qu'un nombre arbitraire de variations peut être ajouté, et que leur animation est aisée. Pour faire un clignement d'œil, il faut dessiner autant de positions intermédiaires que nécessaire et les parenter en utilisant le système de variations. Au moment de l'animation, il suffira de placer des clefs d'animation entre la première et la dernière position et le rig chargera automatiquement les plaques intermédiaires aux images correspondantes.

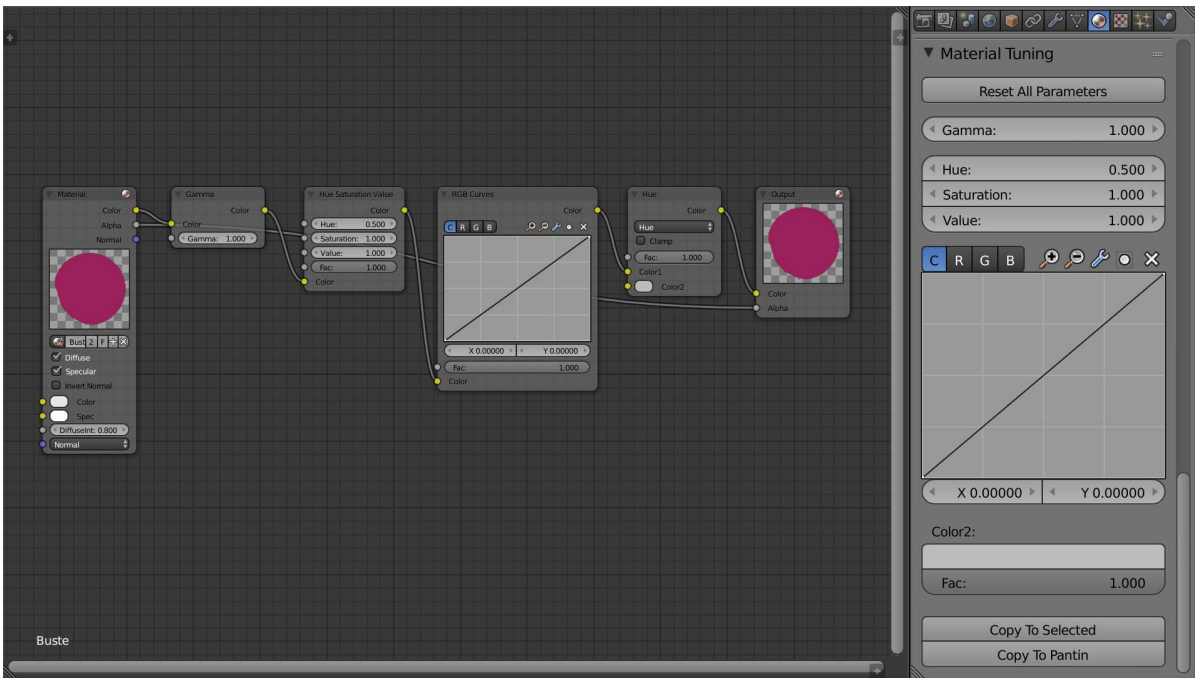


FIGURE III.3.10 – Comparaison de l'interface d'édition des variations colorées. À gauche, l'interface native de Blender, dans laquelle il faut créer pour chaque plaque les nœuds du matériau. À droite, l'interface simplifiée.

Variations colorées

L'autre type de variation avait un but tout différent. Les rues de Paris dans le film devaient fourmiller de vie et montrer des personnages portant des robes de mille couleurs, littéralement. Pour éviter de dessiner autant de personnages différents, certains sont réutilisés plusieurs fois dans le film en changeant la couleur des vêtements. Pour ce faire, j'ai créé une interface rapide à utiliser (figure III.3.10), permettant de changer quelques réglages de couleur sur l'ensemble du pantin ou sur certaines parties seulement. L'interface consiste en un panneau exposant quelques réglages standard (teinte, valeur, saturation ; gamma ; courbe d'ajustement ; recoloration). Ces réglages peuvent être modifiés pendant l'animation. L'approche (shading nodal) proposée par Blender pour réaliser ce type d'opérations offre une grande liberté, mais elle est beaucoup trop complexe et lente pour être mise en œuvre sur un grand nombre d'objets. Cette nouvelle interface, en sacrifiant toute la flexibilité du système de base pour se concentrer sur quelques réglages, s'appuie sur les capacités de Blender, et rend l'opération très simple

et presque instantanée.

III.3.1.5 Publication libre

Des outils tels que ceux décrits ici sont fréquemment développés au cours d'une production précise pour alléger le poids d'opérations spécifiques. Leur développement est toujours complexe, car il faut trouver un équilibre entre un développement dans l'urgence, permettant de répondre aux impératifs du projet, et une élaboration plus réfléchie, permettant une réutilisation pour des projets aux contraintes différentes.

En publiant les outils sous licence logiciel libre, il est également possible de considérer leur utilisation par d'autres studios [10], et donc de les rendre accessibles, documentés, supportés et génériques. À la fin de la production de *Dilili à Paris*, certains outils ont été directement publiés sous licence GPL (voir section II.1.3). Plutôt qu'une solution clef en main, chaque outil est utilisable indépendamment.

En revanche, pour l'outil d'autorig fondé sur Rigify, la publication a aussi eu lieu à l'issue de la production, mais de manière insatisfaisante, car il répondait à des contraintes de production précises et n'avait pas été rendu assez générique et facile à utiliser pour d'autres structures. Un travail ultérieur de publication a donc été entrepris. Il est détaillé dans la section II.3.3.4.

III.3.2 Rendu

Le rendu (cf. section I.1.2.2) est une étape indispensable, puisque c'est à travers elle que sont générées les images qui figureront dans le film, ou du moins un matériau de base, qui sera corrigé, retouché et amélioré au compositing.

Les logiciels de création 3D* livrent en général un ou plusieurs moteurs de rendu intégrés, permettant de paramétrer et de lancer un rendu directement depuis l'interface du programme. Il existe également des moteurs séparés, dont l'exécution est indépendante.

10. De manière générale, une utilisation publique, qu'il s'agisse de studios ou d'artistes indépendants. Voir l'exemple de l'annexe C.

Dans ce cas, une intégration peut être fournie pour, là aussi, lancer le rendu depuis le programme de création, ou pour exporter les données dans un format reconnu par le moteur.

III.3.2.1 Types de moteurs existants

Plusieurs moteurs de rendu sous licences libres existent. On peut les répartir dans plusieurs catégories. La première est dédiée à la recherche sur les algorithmes de rendu. Parmi cette catégorie, on trouve Mitsuba et PBRT, qui est en fait un ouvrage pédagogique sur la conception de ce type de programmes, ainsi que l'implémentation d'un moteur en langage C++. Ces moteurs, quoique sérieux, ne sont pas conçus pour une utilisation en production pour de l'animation, et je n'en parlerai pas plus avant. La deuxième catégorie est celle des moteurs expérimentaux ou jouets, que leurs auteurs publient sous licence libre, mais sans réel support ou développement adaptés à une utilisation en production. Je n'en parlerai donc pas non plus. La troisième et dernière catégorie est au contraire celle des moteurs conçus pour la production, intégrés ou non aux logiciels de création.

III.3.2.2 Moteurs utilisés par les Fées Spéciales

Parmi les nombreux moteurs de cette dernière catégorie, je me concentrerai sur ceux qui ont été utilisés dans les productions de la société où j'ai travaillé. Trois moteurs ont été utilisés, selon les besoins des productions, en particulier selon le style de rendu recherché. Ils sont tous les trois développés par les développeurs de Blender et intégrés à ce logiciel.

Il y a plusieurs raisons pour les avoir choisis. D'abord, c'est le choix par défaut. De la même manière que les utilisateurs de Maya ont longtemps utilisé Mental Ray car celui-ci était livré avec celui-là, les utilisateurs de Blender apprennent généralement les moteurs de rendu livrés avec, en même temps que l'utilisation de Blender lui-même. De plus, le fait que les moteurs soient développés par la même équipe garantit une com-

patibilité immédiate : dans le cas contraire, l'utilisation d'un moteur externe demande le développement et la maintenance d'une passerelle entre le programme de création et le moteur de rendu. Cette passerelle peut être faite par les développeurs du moteur, ou au sein des studios. Par exemple, les développeurs du moteur AppleSeed fournissent des add-ons* pour plusieurs logiciels de création. Certains moteurs comme Guerilla Render (non-libre) choisissent de fonder le pipeline* de rendu sur des formats d'échange standard (Alembic). Cette option ne fonctionne donc que dans la mesure où ces formats sont très bien supportés par le programme de création, ce qui n'était pas le cas de Blender jusqu'à récemment.

Cycles : rendu réaliste

On peut distinguer grossièrement deux types de rendu dans les productions d'animation : photoréaliste et non-photoréaliste (cf. section I.1.2.2). Ces étiquettes sont mal définies, mais elles sont utiles pour classer rapidement les moteurs. Dans ceux que nous avons utilisés, le premier, Cycles, permet un rendu photoréaliste, mais lent. Il a par exemple été utilisé par le studio Tangent Animation pour le long-métrage *Next Gen*, sorti en 2018 et présentant des environnements stylisés, mais un rendu réaliste.

Cycles est un path tracer, une technique permettant d'obtenir un rendu d'un grand réalisme, au prix de longs temps de calcul. La puissance de calcul atteinte par les ordinateurs au cours de la décennie 2010 a permis de rendre viable cette technique en production. Du fait de sa nature parallélisable, elle peut notamment utiliser le rendu sur carte graphique (GPU), qui la rend plus accessible, y compris aux petits studios et aux indépendants.

Cycles est en développement actif par les développeurs de Blender, et a par exemple récemment intégré une option efficaces de débruitage et la possibilité de paramétrer des AOV [11].

11. Les *Arbitrary Output Variables* ou variables de sortie arbitraires permettent de rendre plusieurs shaders définis par l'utilisateur pour chaque image, pour les utiliser au compositing. Elles offrent une bien plus grande souplesse de travail, et de meilleures performances. La plupart des moteurs proposent cette possibilité depuis longtemps, mais je cite cet exemple pour montrer que le moteur évolue aujourd'hui à un rythme rapide.

Les deux autres moteurs qui ont été utilisés sont Blender Internal et Eevee. Ils permettent tous deux, moyennant des efforts de la part des graphistes, d'obtenir des images assez réalistes, mais ils sont aussi mieux adaptés pour du rendu non-photoréaliste : ils sont plus rapides et offrent des possibilités de shading assez avancées.

Eevee

Eevee a été utilisé pour un projet plus récent utilisant le Grease Pencil (cf. section II.3.1), du fait de son intégration avec la dernière version de Blender. Là aussi, l'utilisation était basique, d'une petite partie seulement de ses capacités. Le développement de ce nouveau moteur en 2018-2019 a été beaucoup attendu par la communauté d'utilisateurs, car il utilise les technologies issues du rendu temps réel des jeux vidéo pour créer des images de haute qualité, en un temps bien plus faible que les moteurs réalistes fondés sur le ray tracing (comme Cycles). Ces développements sont abordés dans la section II.3.1.

Eevee ne permet pas d'obtenir un résultat aussi photoréaliste qu'un path tracer, mais peut déjà être utilisé dans beaucoup de cas, y compris pour des projets relativement réalistes. Il peut aussi parfaitement servir pour faire des rendus rapides, mais donnant une idée assez fidèle du rendu définitif, au layout par exemple.

Blender Internal

Blender Internal est un moteur de rendu ancien qui n'est plus supporté aujourd'hui car il a été remplacé dans la dernière version de Blender par Eevee. Il a été utilisé pour prévisualiser les séquences* layout de *Dilili*, et pour les animations du musée de Lodève. Dans ces projets, les besoins de rendu étaient assez simples, et il fallait principalement que les dessins et couleurs des décors et des personnages se retrouvent fidèlement dans le rendu.

Le fait que plusieurs moteurs de rendu de haute qualité soient disponibles et en développement actif ne laisse pas d'inquiétude sur la capacité des studios à les utiliser pour

calculer les images de leurs films. Bien qu'en 2019, certaines fonctionnalités importantes manquent aux moteurs de Blender [12], ils sont déjà entièrement valables et assez stables et rapides pour les besoins modérés des Fées spéciales, et pour les besoins moins modérés d'un film comme *Next Gen*, ou des films de la Blender Foundation comme *Agent 327* et *Spring*.

III.3.3 Compositing

Le compositing (cf. section I.1.2.3) est l'étape qui suit directement le rendu. Elle utilise les images générées par le moteur de rendu, les combine, les assemble, les retouche pour obtenir l'image finale. Les programmes doivent être capables d'une excellente gestion de la couleur, avoir de bonnes performances pour traiter rapidement de nombreuses images avec des effets parfois complexes et avoir un retour visuel aussi immédiat que possible. Ils doivent aussi, peut-être encore plus que les autres programmes de la chaîne de fabrication, être très stables, car il s'agit de la dernière étape de la chaîne de fabrication à proprement parler, et que les compositeurs doivent donc souvent travailler vite et efficacement pour compenser les retards accumulés en cours de production.

III.3.3.1 Logiciels prédominants

Il existe deux approches complémentaires du compositing : le compositing par couches, et le compositing nodal. Le représentant le plus répandu de la première catégorie est Adobe After Effects. Il est dédié à la fois au compositing pour l'animation et les effets spéciaux, et au motion design. Il est utilisé par une majorité des studios de graphisme. Son approche du compositing est unique, et permet une grande efficacité pour certaines tâches. Il n'existe pas aujourd'hui d'équivalent libre à After Effects.

L'autre approche du compositing, le compositing nodal, permet de visualiser le processus dans l'interface par des boîtes (les nœuds), et par des fils qui relient ces boîtes.

12. Par exemple, la gestion des layers et passes et les *overrides* par layer sont des fonctionnalités attendues depuis longtemps.

Les nœuds sont des opérateurs de base (combiner des images, ajouter un filtre, modifier la couleur, etc.), et les images passent d'un nœud à l'autre via les fils. Ce type de représentation des données est très populaire, et la majorité des programmes de compositing l'utilisent en raison de son caractère intuitif, et de la facilité à réutiliser des opérations de base ou des combinaisons complexes d'un plan* à l'autre. Parmi les nombreux programmes de compositing nodal, le plus répandu actuellement est Nuke.

After Effects n'est pas prévu pour les mêmes utilisations que les logiciels de compositing nodal, et malgré ses limitations (voir la note de bas de page de la section II.1.1.3), il rend relativement simples certaines opérations qui seraient complexes avec des logiciels comme Nuke.

III.3.3.2 Natron

Plusieurs logiciels libres permettent de faire du compositing nodal. Initialement, l'équipe des Fées spéciales a décidé, dans la logique d'utilisation et de développement de logiciel libre, d'utiliser le logiciel Natron.

Il a été développé en France entre 2013 et 2017 par Alexandre Gauthier et Frédéric Devernay, avec un financement public de l'INRIA. Il s'agit en fait d'un clone de Nuke. Natron a depuis le départ copié l'interface devenue standard de Nuke, en s'appuyant sur un moteur nouveau et libre. L'idée derrière cette copie de l'interface [13] était de faciliter l'adoption de Natron en fournissant aux utilisateurs une interface déjà connue. S'il manque beaucoup de fonctions avancées à Natron, il offre toutes les fonctionnalités de base d'un logiciel de compositing : gestion et conversion des formats d'image et vidéo ; gestion des espaces colorimétriques ; combinaison de passes, filtres, rotoscopie.

Nous avons donc utilisé Natron sur plusieurs productions, et n'avons pas eu de problème particulier avant de commencer les films sur l'Antarctique (cf. section III.1.2.2), qui étaient en 4K (Ultra HD), une dimension d'image encore très lourde pour les ordina-

13. Il semble qu'une telle copie ne constitue pas une contrefaçon du moment que le but n'est pas de se faire passer pour le programme d'origine, mais d'utiliser son mode de fonctionnement pour être plus intuitif pour les utilisateurs. Dans certains pays dont la France ne fait pas partie, des éléments d'interface peuvent également être protégés par des brevets.

teurs de 2016. Certains plans demandaient un compositing complexe, avec de nombreuses passes. Nous n'avons pas pu savoir précisément pourquoi, mais Natron a montré ses limites dans ces cas, et plantait systématiquement lors des rendus nocturnes des plans composités dans la journée. Il va sans dire qu'il fallait relancer les rendus ratés, ce qui ralentit forcément la production, puisque des humains doivent intervenir pour surveiller les rendus, et que les images qui n'avaient pas été calculées une nuit mobilisaient des ordinateurs dans la journée, ralentissant les performances et donc la productivité. De plus, le programme n'était déjà plus vraiment supporté à l'époque. Le constat de ces problèmes a poussé le chef compositeur à utiliser en urgence un programme qu'il connaissait bien, BlackMagic Fusion. Ce dernier est gratuit, mais non-libre. Il a été capable de refaire le compositing dans ce programme et de rendre les images dans les temps.

Nous avons ensuite utilisé ce même programme sur plusieurs projets, toujours sous la direction du chef compositeur. C'était une bonne décision pour la production, mais le fait de ne pas avoir pu utiliser de programme libre pour des questions de stabilité a entraîné des questionnements. Le développement de Natron, qui s'est arrêté peu après, en 2017, avait stagné depuis un certain temps, et seules étaient publiées des corrections de bugs majeurs. Il n'y a pas eu, dans les derniers temps, d'amélioration de performance et de stabilité, ni d'ajout de fonctionnalités. En effet, la subvention de l'INRIA étant arrivée à terme, les développeurs ont tenté de trouver un modèle économique en vendant du support pour le programme, mais l'adoption n'était pas suffisante et, comme l'expliquent les développeurs dans une interview [60], aucun nouveau développeur n'a été attiré vers le projet, si bien qu'une fois les deux développeurs originaux partis pour d'autres projets, personne n'en a fait la maintenance en 2019. Sans support, son utilisation serait téméraire en production. On peut en conclure que le développement de logiciels libres ne demande pas seulement des compétences solides de développement informatique, mais également de la gestion d'équipe, de la communication, et même du marketing. Les projets qui « réussissent », c'est-à-dire qui au minimum continuent d'être développés dans le temps, sont ceux qui ont su fédérer une communauté, ou trouver un modèle économique pour financer le développement. Cette question est abordée dans le chapitre chapitre II.2.

III.3.3.3 Blender

Un autre programme libre permettant de faire du compositing nodal est, encore une fois, Blender. En effet, il existe depuis 2006 un module de compositing qui a été utilisé et développé pour les films de la Blender Foundation. Il permet lui aussi de réaliser les opérations de compositing de base, mais à la fois son interface, ses performances [14], et son extensibilité sont critiquées.

En effet, beaucoup de logiciels spécialisés comme Nuke et Natron permettent d'utiliser de puissants plug-ins*, libres ou commerciaux, grâce à la bibliothèque* open source OpenFX. Ces plug-ins peuvent être intégrés dans plusieurs logiciels de compositing pour combler certains manques de fonctionnalités, et sont ainsi portables : même en changeant de programme, l'utilisateur peut continuer à utiliser les plug-ins. Ce système n'est pas intégré dans Blender, et la seule manière d'y ajouter des fonctions est de modifier un code source* à l'architecture compliquée. Cela pose des problèmes de maintenance si les changements ne sont pas acceptés par les développeurs dans le programme.

14. Par exemple, il est impossible en 2019 de calculer en une fois la séquence entière, et de la lire depuis un cache. Les images sont donc systématiquement recalculées lors de la lecture, même si elles n'ont pas changé. Le système ne prévoit pas non plus de versions proxy en plus basse résolution pour accélérer le rendu.

Chapitre III.4

Post-production

Introduction

La post-production intervient comme son nom l'indique après la fabrication et comprend plusieurs étapes permettant de finaliser l'œuvre animée. On fait parfois entrer le compositing dans la post-production. Il est indéniablement possible pour un studio en 2019 de faire certaines étapes en utilisant des logiciels libres.

Je me concentrerai donc dans ce chapitre sur l'étude de deux questions rarement considérées : d'abord, les lecteurs de vidéos spécialisés pour l'animation. Ensuite, les formats de fichiers demandés par les clients en vue de leur diffusion.

III.4.1 Validation des images

En cours de production, il est nécessaire de regarder sans cesse les images fabriquées pour repérer les problèmes techniques et artistiques, les faire valider, ou simplement les voir dans le contexte de la séquence*. Pour faire ces examens, souvent appelés *reviews*, il faut un lecteur de séquences d'images et de vidéos.

Besoins spécifiques d'un lecteur vidéo pour l'animation

Un lecteur multimédia grand public comme VLC n'est pas adapté à cette application. En effet, il faut pouvoir charger des séquences d'images dans des formats de production, faire des allers et retours dans la vidéo de manière instantanée, défiler image par image, regarder plusieurs plans* à la suite, comparer des versions, examiner les passes dans des séquences d'images multi-passes comme OpenEXR, ou les couches (rouge, vert, bleu, alpha). Certains programmes permettent également de faire des annotations sur les vidéos, pour donner des retakes* aux graphistes. De plus, il est utile de pouvoir scripter le programme pour lancer automatiquement des séquences avec des paramètres complexes comme les numéros de plans, les variations à afficher, le mode d'affichage, etc. Il faut que le programme soit intégré au pipeline*, pour pouvoir communiquer par exemple avec le production manager, enregistrer les informations d'annotations, etc.

Si des lecteurs libres pour l'animation existent, comme DJV View, et si des palliatifs peuvent être envisagés, par exemple en écrivant un add-on* pour Blender, il existe un manque de ce côté, peut-être en raison du faible nombre d'utilisateurs experts ayant besoin des fonctions décrites, par rapport à la majorité des utilisateurs de logiciels libres.

III.4.1.1 Développement d'une solution maison fondée sur des programmes libres

Dès le début de la production du premier projet de la société, le film *Dilili à Paris*, nous avons eu besoin d'un lecteur vidéo adapté. Nous n'avions pas tous les besoins énoncé plus haut, mais il nous fallait au minimum la capacité de regarder des séquences entières, et de comparer plusieurs versions, ce que n'offrait aucun lecteur libre par défaut. Pendant plusieurs mois, nous avons pour cela utilisé un script* écrit par Flavio Perez. Il s'appuyait sur des logiciels libres, disponibles dans notre distribution de GNU/Linux* [1]. Son utilisation nous permettait, en entrant une ligne de commande de la forme `play N` de jouer la séquence N. Le programme était chargé d'aller chercher dans l'arborescence

1. En particulier, l'outil en ligne de commande 'melt' fondé sur le framework MLT [58].

de fichiers, les séquences vidéo correspondant aux plans*, et à les jouer dans l'ordre, en offrant à l'utilisateur la possibilité de passer d'un plan à l'autre par un raccourci clavier. Ce programme était donc très pauvre en fonctionnalités comparé à d'autres lecteurs : il n'y avait pas d'interface graphique, pas de timeline*, pas d'options de comparaison de version ou d'annotation, etc.

Extensibilité

De plus, il fallait continuer à le développer, pour ainsi dire à le bricoler, à chaque fois qu'un nouveau besoin se faisait sentir. Par exemple, une étape primordiale lors de l'étape d'export vers Maya était de contrôler que le rendu issu de Blender et celui issu de Maya étaient identiques, et qu'aucune erreur de transformation, de couleur ou d'animation ne s'était introduite lors de l'import. De la même manière que pour la lecture consécutive, nous avons écrit un programme qui utilise la bibliothèque* FFmpeg pour générer une visualisation de la différence entre deux séquences vidéo. Une fois habitué à lire l'image résultante, cette méthode était relativement efficace : ce qui n'avait pas changé apparaissait en gris, le reste prenant une autre valeur ou couleur.

Afin d'obtenir ce qu'un lecteur vidéo avancé permet de faire en temps réel en une simple ligne de commande, ou en un clic si le lecteur est intégré au logiciel de suivi de production, il nous a fallu écrire un script qui transcode la vidéo plusieurs fois, la traite, l'enregistre de nouveau, puis ouvrir la séquence résultante manuellement. Malgré la satisfaction d'avoir écrit nous-mêmes ces outils, cela représentait à chaque fois un temps précieux.

En l'absence d'un lecteur libre intégrant toutes ces fonctions, ou des ressources nécessaires pour écrire notre propre lecteur, nous avons encore une fois acheté des licences du lecteur propriétaire RV, qui fait ce qu'on attend d'un lecteur de séquence en production.

III.4.2 Livraison des masters

La dernière étape d'un projet audiovisuel est la livraison du master, les formats et supports qui seront diffusés et conservés par le client. Ces masters sont créés à partir des fichiers de montage du projet, en les exportant dans un format spécifié à l'avance par le client ou le diffuseur. Ce dernier précise généralement les demandes techniques concernant le format de conteneur, les codecs audio et vidéo, et leurs paramètres.

Lors de la livraison du projet Lodève (cf. section III.1.3), qui était un appel d'offre publique, les formats choisis étaient précisés dans le cahier technique des clauses particulières, transmis avant l'appel d'offre et stipulant les détails de mise en œuvre de l'exposition. Le document stipulait que les masters vidéo devraient être livrés soit au format MPEG2, soit non-compressés. Les sons seraient « au standard vidéo […] ou MP3 ». Sachant qu'il n'existe pas qu'un standard en matière de vidéo, le document était plutôt vague. De plus, il restait suffisamment évasif pour que le choix évolue au cours du projet.

Ainsi, rien n'était véritablement demandé. Nous avons donc traité directement avec la société chargée de la prestation vidéo : matériel de diffusion, installation, programmation des serveurs de diffusion, etc. afin de choisir un format adapté au matériel. La question est épineuse, car le matériel en question accepte plusieurs formats, et en recommande même quelques-uns. Il n'y avait donc pas une réponse unique. Le technicien nous recommandait le codec H.264. Ce format est pratiquement le choix par défaut tant il est omniprésent en 2017. Quoique très polyvalent et efficace, il n'est pas clairement libre, pour deux raisons.

Protection par les brevets logiciels

D'abord, sa spécification est en partie établie par un consortium d'entreprises qui ont pour but l'exploitation commerciale de cette technologie, et en contrôlent l'accès grâce à des brevets logiciels. Ces brevets logiciels n'ont pas cours en France, mais il n'en reste pas moins que le codec n'est pas libre dans toutes les juridictions, ce qui peut le rendre impropre à la diffusion dans certains pays.

Les droits d'utilisation de ces brevets concernent en principe tout utilisateur, que ce

soient les fabricants de matériel implémentant la technologie, ou les utilisateurs finaux. Il semble [69] que la redevance ne soit pas réclamée pour les vidéos diffusées gratuitement sur Internet, et il est peu probable qu'un procès ait lieu, simplement parce qu'un prestataire a encodé une vidéo sans payer de droits.

Choix de l'implémentation

De plus, il existe plusieurs implémentations d'encodeurs et de décodeurs, certains propriétaires et d'autres libres. Même si le format n'était pas protégé par des brevets, il faudrait être vigilant sur le programme choisi pour encoder la vidéo. L'encodeur x264 est libre et fournit donc une solution à ce problème précis. Néanmoins, la question est trouble, et nous aurions aimé pouvoir utiliser un format standard et libre, sans ambiguïté.

Cela nous a été impossible. En effet, nous avons obtenu un prêt de matériel afin de tester nos vidéos encodées en conditions réelles. Nous avons essayé plusieurs codecs, plusieurs conteneurs, et de très nombreux réglages afin d'obtenir un résultat visuellement fidèle (compression imperceptible), et une lecture fluide (rapidité de lecture du fichier sur le disque, et vitesse de décodage), mais une combinaison de conteneur et de codec libres telle que ogg + theora n'était simplement pas supportée, ce que confirme la documentation de l'appareil. Nous avons donc utilisé le codec h.264.

Cette concession est faite presque universellement par les studios dans la livraison pour certains médias de diffusion, sans même que la question ne se pose. Il a été intéressant, justement, de la poser, et de constater que d'une part la réponse est inattendue (l'utilisation de certains formats propriétaires est en principe illégale sans paiement d'une redevance), et que d'autre part il est dans certains cas impossible de faire autrement.

Chapitre III.5

Outils transversaux

Introduction

Le pipeline* (cf. section I.1.2), même s'il est difficile à définir précisément, est indéniablement important pour la fabrication au sein des studios d'animation. La troisième partie de cette thèse s'est attachée jusque-là à décrire les possibilités d'utilisation et de développement du libre dans chaque partie de la chaîne de fabrication. Ce chapitre prendra une approche différente, en abordant les éléments du pipeline communs à toutes les étapes. Le pipeline est ici compris dans son sens le plus technique, c'est-à-dire la structure invisible qui permet de fluidifier la fabrication, les échanges de données entre les départements, et une vision plus globale et transversale de la fabrication, plutôt qu'une étape précise.

J'aborderai dans ce chapitre deux domaines qui n'interviennent pas directement dans le graphisme et les opérations réalisées par chaque département, mais qui sont néanmoins indispensables. Le premier est le suivi de production et la gestion d'assets, pour lesquelles il existe des logiciels libres performants, testés avec succès en production dans la société Les Fées spéciales. Le deuxième domaine d'intérêt concerne les échanges de données entre plusieurs programmes, et d'un studio à l'autre, et les développements nécessaires pour pallier les manques des formats d'échange et de leur implémentation dans les logiciels de création. Je me concentrerai en particulier sur l'exemple du film *Dilili à Paris* (cf.

section III.1.1).

III.5.1 Logiciels de gestion de production et d'assets

Comme expliqué dans la section I.1.2.4, il est nécessaire pour une production d'animation de suivre efficacement les éléments en cours de fabrication ou de validation, les différentes versions et variations de ces éléments, les tâches terminées ou restant à faire et les graphistes auxquels elles sont attribuées, le calendrier et le budget de fabrication. Ces suivis sont effectués à travers deux types d'outils complémentaires, les logiciels de suivi de production et de gestion d'assets*.

Dans un article [29] publié par le groupe Le Pipeline [1], le TD* Christophe Archambault développe le rôle de ces outils pour les studios, et en donne une liste succincte mais instructive. Parmi eux, il existe plusieurs solutions libres pour le suivi de production et la gestion d'assets. Je parlerai ici de deux logiciels libres, tous deux utilisés en production par les Fées spéciales, et au développement desquels la société a contribué.

III.5.1.1 Suivi de production : CGWire

Les logiciels de suivi de production permettent de faire l'interface entre les graphistes, qui fabriquent le film, leurs chefs d'équipe, et les chargés de production, qui s'assurent de la bonne avancée du projet. Il faut que le logiciel facilite la communication, en présentant les informations relatives au projet de manière claire et logique, et en permettant une saisie facile de ces informations, par les graphistes et les chargés de production.

Financement du développement

Sur ses premiers projets, la société Les Fées spéciales a développé un prototype d'outil interne, Criquet [2]. Il avait vocation à être développé plus avant, pour en faire une

1. Ce groupe de travail français est composé de professionnels du secteur de l'animation, travaillant dans des studios au poste de TD.

2. Ce logiciel, utilisé pour le suivi de la fabrication de *Dilili à Paris* au sein des Fées spéciales, est détaillé par son auteur, Flavio Perez, dans [20].

solution générique de suivi de fabrication, et diffusée sous licence libre. Une aide financière au développement avait été obtenue auprès du CNC*. En 2017, Les Fées spéciales ont eu connaissance du développement de CGWire, un logiciel aux fonctionnalités équivalentes développé dans la société du même nom par Frank Rousseau. La société a donc décidé d'adopter ce programme pour ses productions, et d'affecter le fonds d'aide au développement à son développement.

Il a été mis en production pour la première fois sur le projet Lodève, alors que le programme était encore jeune, et qu'il y manquait de nombreuses fonctionnalités, développées depuis avec le financement des Fées spéciales et d'autres studios. Le processus de développement du programme est décrit dans la section II.2.3.

Fonctionnalités

Concrètement, CGWire est composé de plusieurs modules interconnectés. Zou, le serveur organisant la base de données et contrôlé par une API* ; Gazu, un client Python pour les développeurs ; et Kitsu, un client Web pour les graphistes et les chargés de production. Il offre les fonctionnalités attendues d'un tel logiciel : les graphistes peuvent voir un tableau récapitulatif des tâches qui leur sont assignées, et saisir pour chaque tâche le statut en cours et le temps qui y a été consacré. Ils peuvent ajouter des commentaires et envoyer la dernière version de la vidéo du plan*. Les chargés de production, de leur côté, suivent l'avancée des graphistes sur l'ensemble de leurs tâches, et leur assignent des tâches au fur et à mesure. Lors des réunions de validation avec les clients, les chargés de production peuvent préparer les plans à montrer, et les annoter des retakes* des clients, pour que les graphistes sachent ce qu'ils auront à changer.

Les fonctionnalités décrites sont donc standard pour un logiciel de suivi. Il est intéressant de noter qu'en raison du mode de financement du développement du programme, ces fonctionnalités sont développées au fur et à mesure que les utilisateurs, au sein des studios, en font la demande auprès de CGWire, et certains studios financent les heures de développement et ont donc la priorité pour la mise en œuvre de fonctions répondant à leurs besoins. Le programme étant open source, tous les utilisateurs bénéficieront de

ces améliorations en installant la version suivante.

Licence et conditions d'accès

CGWire est proposé par son éditeur selon plusieurs modalités différentes, utilisant toutes le même code source* [3]. Le client peut d'abord souscrire à un abonnement, avec un hébergement sur les serveurs de CGWire [4]. Le client n'a pas besoin de s'occuper de l'installation et de la maintenance du programme et de la base de données. En contrepartie, il ne peut pas directement modifier ou étendre le code du programme, ni avoir le contrôle de ses données. Une autre possibilité, puisque le code est open source, est d'installer le programme sur sa propre instance, par exemple dans les locaux de l'entreprise. CGWire peut également facturer une assistance pour installer et faire le support du programme chez le client. Le logiciel est donc destiné aux équipes assez réduites, les petites et moyennes entreprises, voire les écoles d'animation.

Il est intéressant de comparer ces différentes modalités d'accès à celles du logiciel le plus utilisé en 2019 pour le suivi de production, Shotgun, édité par Autodesk. En effet, une production suivie avec Shotgun s'appuie également sur une base de données contenant les informations de suivi. Cette base de données est hébergée sur des serveurs distants appartenant à la société Amazon, et la licence utilisateur standard ne prévoit pas qu'un studio puisse héberger ou sauvegarder lui-même ses propres données. Le studio dépend donc de la disponibilité constante et à long terme de l'outil pour son activité.

III.5.1.2 Gestion d'assets : Kabaret

Les gestionnaires d'assets sont des programmes complexes qui doivent pouvoir s'adapter à l'organisation d'un studio et des différents projets au sein de ce studio. Il existe

3. Certains développeurs de logiciels libres, comme par exemple le logiciel de dépôt de code et de développement collaboratif GitLab créent plusieurs versions, une version libre, et une version propriétaire commerciale, qui offre des fonctionnalités supplémentaires et dont les développements sont généralement reversés dans la version libre quelque temps après la version propriétaire.

4. Ce modèle commercial est celui du *Software as a Service*, où le client loue un service au prestataire, qui héberge le logiciel et les données sur son propre serveur, en offrant une interface web pour s'y connecter.

des solutions commerciales [5] mais la plupart des studios qui travaillent sur des projets d'envergure développent leur propre outil interne.

Approche modulaire pour les TD

Kabaret [43] est un framework* de gestion d'assets écrit par Damien Coureau au sein du studio Supamonks. Plutôt qu'une solution imposant une manière de faire ou une organisation précise, il permet de faciliter pour le TD* la tâche de décrire l'organisation de son studio, à travers l'implémentation de *flows* (cf. section I.1.2) : le workflow et le dataflow. Cela signifie que les TD peuvent utiliser Kabaret pour décrire n'importe quel pipeline*, et ne sont pas obligés de se plier à une façon de faire imposée par le logiciel. En contrepartie, ils *doivent* nécessairement faire du développement supplémentaire, car Kabaret n'est pas une solution clef en main. Cette approche est logique, du fait des spécificités de chaque production, y compris au sein d'un même studio.

Un grand nombre de variables est à prendre en compte lors de la conception du *flow*, et un pipeline devra nécessairement être adapté ce qui proscrit l'usage d'une solution toute faite. Parmi ces variables, on peut citer :

- le type d'œuvre (court ou long métrage, série, œuvre multimédia) ;
- le format (résolution d'image, durée, nombre d'épisodes) ;
- la technique (dessin animé, image de synthèse 3D) ;
- l'importance pour la production de chaque département (animation, effets spéciaux, compositing, etc.) ;
- les programmes utilisés ;
- les données reçues des clients ou des collaborateurs extérieurs.

Intégration dans les logiciels de création 3D

Le pipeline doit être solidement conçu par les TD, et facile à manipuler pour les graphistes. Par exemple, lorsqu'un opérateur crée une version d'un plan* sur lequel il

5. Par exemple, Autodesk édite également Tank, un logiciel de gestion d'asset censé être utilisé avec Shotgun, et intégré avec lui et les logiciels de création 3D*.

travaille, il faut qu'il puisse renseigner facilement les informations de publication. Comme la plupart des logiciels de gestion d'assets, Kabaret est écrit en Python avec Qt pour l'interface, et il est donc relativement facile à intégrer dans beaucoup de logiciels de création 3D [6]. Ainsi, le graphiste a moins d'étapes à réaliser pour publier son travail, et la publication est une étape logique et intégrée à son processus de travail.

Kabaret est publié sous licence LGPL, une licence copyleft faible (cf. section II.1.3.1). Il peut donc être inclus dans une application propriétaire sans que celle-ci ne devienne copyleft. En revanche, toute modification de la bibliothèque elle-même doit être redistribuée sous la même licence LGPL.

III.5.2 Communication avec d'autres studios

Un critère important dans le choix d'un logiciel par un studio est sa capacité à s'intégrer avec d'autres logiciels, et éventuellement avec le pipeline* d'un autre studio, qui peut être complètement différent. Pour cela, il est parfois possible, lorsque les deux studios anticipent suffisamment, de se mettre d'accord en amont pour utiliser les mêmes solutions techniques. Mais cette solution est parfois impossible, notamment si le pipeline d'un des studios repose sur l'utilisation d'un logiciel particulier. Dans ce cas, il faut utiliser des formats d'échange, et développer des passerelles, c'est-à-dire des moyens de faire passer les données et métadonnées d'un studio à l'autre, en respectant leurs nomenclatures et processus de fabrication respectifs. Cette section abordera l'exemple concret du film *Dilili à Paris* (cf. section III.1.1).

Contraintes de projet : un pipeline complexe

Comme nous l'avons vu, la première tâche du studio sur ce film était le layout (cf. section III.2.2). Le pipeline* de fabrication du film était complexe, et faisait intervenir plusieurs studios utilisant des programmes différents. Les Fées spéciales travaillaient sur Blender, tandis que Mac Guff Ligne travaillait sur Maya. Je montrerai dans cette section

6. Blender est une exception majeure, voir section II.1.1.3.

les développements qui ont dû être réalisés pour pouvoir échanger les données avec ce studio, en me concentrant sur le layout. Si le film avait été réalisé en 2019 plutôt que 2016, les solutions choisies auraient été différentes, car les programmes ont évolué entre temps. Néanmoins, les principes exposés gardent leur valeur générale, car il est toujours nécessaire de développer des passerelles dans une production.

Pour pouvoir collaborer avec le réalisateur Michel Ocelot et les équipes de Mac Guff, il a fallu, avant même le début de la production, communiquer avec le directeur technique du projet au sein de ce studio, Malek Touzani, pour établir un cahier des charges des formats d'échanges. Il fallait que nous puissions créer des fichiers contenant les indications nécessaires, et qu'ils soient capables de les traiter pour les intégrer à leur pipeline.

Les éléments à fournir étaient les suivants :

- caméras, dont :
  - transformations,
  - focale,
  - animation ;
- décors ;
- personnages et accessoires animés ;
- groupes contenant et séparant ces différents objets ;
- bande-son ;
- une nomenclature et des réglages particuliers pour les scènes.

Transmission des données à l'autre studio

Afin que chacun de ces éléments soit transmis exactement selon leur forme validée par l'auteur, nous disposions de plusieurs possibilités. Nous aurions pu convenir d'un format d'échange standard, que nous exporterions et que Mac Guff importerait. Cette solution s'est avérée impossible à mettre en œuvre, car aucun format permettant de contenir fidèlement toutes ces données n'était disponible pour Blender. Le format d'échange Alembic, sorti en 2011 et largement adopté par l'industrie, était en cours d'intégration dans Blen-

der, et donc inutilisable. Or, c'est le seul format qui aurait pu éventuellement permettre de transmettre facilement et correctement l'animation. Mais il était trop tôt et nous ne pouvions nous appuyer sur ce format.

Le processus le plus viable pour nous prémunir de pertes de données était donc d'envoyer directement des fichiers au format Maya, utilisé dans le pipeline de Mac Guff. Cela implique évidemment de disposer au minimum d'une licence de ce logiciel non-libre. C'était le cas puisque le directeur technique des Fées spéciales, Flavio Perez, disposait d'une licence personnelle vieille de plusieurs versions mais encore compatible. L'intérêt de ce processus était de contrôler l'échange en s'assurant que toutes les données soient présentes à l'envoi. Cependant, le problème demeurait entier de les passer de Blender à Maya. Pour cela, nous aurions pu concevoir un format propriétaire unique à cette production, ainsi qu'un exporteur Blender et un importeur Maya. Une telle solution est irréaliste et surdimensionnée, sachant qu'il existe déjà des formats plus ou moins bien implémentés, mais offrant de nombreuses capacités. Ainsi, le format propriétaire FBX (cf. section II.3.2.1), maintenu par Autodesk, est vieillissant mais encore reconnu dans le jeu vidéo et l'animation. Il est mal documenté, instable, implémenté différemment dans tous les programmes, mais il a le mérite de convoyer facilement un minimum d'informations, notamment les hiérarchies d'objets, les maillages, les armatures, les animations, les matériaux. Le format a fait l'objet d'une rétro-ingénierie par les développeurs de Blender, qui fournit donc un exporteur.

Les personnages ont été modélisés à cette étape dans une version « RLO » (Rough LayOut), c'est-à-dire layout approximatif. Ce ne sont pas les modèles définitifs qui seront présents dans le film, mais des versions simplifiées, ayant juste assez de détail pour donner une idée juste de leurs taille et corpulence, de leur silhouette. De la même manière, ils sont riggés et animés relativement grossièrement.

III.5.2.1 Format maison

Dès les premiers essais d'export, nous avons constaté que le format FBX était malgré tout insuffisant pour notre usage. En effet, outre certains bugs connus et faciles à corriger

faisant que les transformations étaient corrompues, l'animation squelettique (déformation du maillage) n'était pas supportée, ni certains réglages de caméra, les groupes, les séquences audio, l'animation de visibilité. Les clefs d'animation telles qu'exportées par Blender étaient également corrompues.

Au lieu de chercher à corriger un exporteur complexe vers un format non standard, nous avons estimé préférable de l'utiliser tel quel, mais de pallier ses manques en l'encapsulant dans un export maison. Cela signifie que notre exporteur préparait la scène, puis appelait l'exporteur FBX une fois par groupe d'objets (Layout, Pantins, Décors, Caméra, etc.). Chaque groupe se retrouvait donc dans son propre fichier FBX. Afin de coordonner ces différents fichiers, nous avons également utilisé des fichiers simples fondés sur JSON*, stockant des liens vers les FBX, le fichier audio, et différents réglages supplémentaires. Un exemple d'un tel fichier est présent en annexe A.

Cet arrangement permet non seulement de séparer les objets par groupes logiques, mais encore d'affecter à chacun de ces groupes des propriétés différentes. Par exemple, les décors ne nécessitent pas d'être animés, à l'inverse de la caméra. Les personnages 3D doivent conserver leurs animation et déformation, mais leur couleur est accessoire et donc pas exportée, contrairement aux décors. Ces différents réglages pouvaient être définis dans le format JSON pour chaque groupe. Des réglages supplémentaires étaient également écrits dans ce fichier, comme les images d'entrée et sortie du plan*, la résolution, l'animation de certains paramètres de caméra non supportés par FBX, etc.

III.5.2.2 Export de la déformation

En s'appuyant sur le format FBX, nous connaissions d'emblée une limitation propre à ce format : l'incapacité à stocker les informations de déformation. Le rig des personnages ayant été fait dans Blender, il était aventureux de souhaiter exporter une animation squelettique (fondée sur un système d'armature) d'un programme à l'autre, en retrouvant *exactement* les déformations. Le plus simple dans ce contexte est d'employer un système de cache*. Une fois ce cache* réappliqué, l'animation peut être lue depuis le fichier, au lieu d'être calculée à partir de l'armature. Cette dernière peut donc à la

limite être supprimée. C'est une solution qui est souvent employée, entre deux étapes de production, pour garantir que l'animation ne soit pas corrompue ou modifiée. Par exemple, on emploiera ce système entre l'animation et le lighting, afin que les artistes éclairagistes n'aient pas à se soucier de questions de rig ou d'animation, mais puissent se concentrer sur leur tâche. Ce principe convenait parfaitement à notre situation, puisqu'il nous évitait de lutter pour rendre compatibles deux systèmes de conception très différentes : les rigs de Blender et de Maya.

Blender fournit dans un add-on* la capacité d'exporter un cache au format PC2, que Maya est de son côté capable de lire, en le convertissant dans son propre format MC (Maya Cache). Cette opération de conversion est incluse dans le script* d'import, pour que l'importeur Maya s'en occupe tout seul.

III.5.2.3 Import dans Maya

Une fois les données de layout correctement exportées dans les différents formats, une procédure d'import doit être mise en œuvre dans Maya pour retrouver la scène à l'identique. On pourrait croire que cette étape serait triviale, comme l'import des fichiers FBX, mais en réalité des particularités du système d'import de Maya forcent à intervenir après l'export pour faire des corrections. Par exemple, les noms des objets étaient souvent corrompus. Les chemins des fichiers de textures devaient systématiquement être retouchés pour être reconnus par Maya. Les rotations étaient également tellement corrompues que nous avons décidé de les réappliquer systématiquement en écrasant la hiérarchie d'objets, pour ne plus se soucier des questions de parentage.

Plusieurs autres manipulations avaient lieu au moment de l'import, la plupart permettant de réappliquer les réglages propres aux différents groupes (Layout, Décors, etc.) exposés plus haut. Le fait d'automatiser cette étape permettait d'en profiter pour conformer les fichiers aux desiderata de Mac Guff en matières de nomenclature, de réglages de rendu, de résolution et de *framerate*, de format de rendu, d'unités, etc.

Une fois la scène correctement importée et vérifiée, il ne restait plus qu'à envoyer sur le serveur FTP de Mac Guff les dossiers Maya zippés, et à leur transmettre la liste des

plans* concernés. De leur côté, la production pouvait donc faire le suivi des échanges, et la technique nous donner d'éventuelles retouches à faire sur les scènes.

Chapitre III.6

Exploitation de données ouvertes

Introduction

Une des activités de la société Les Fées spéciales est de concevoir et fabriquer des programmes animés pédagogiques pour des musées et des films. Ces programmes sont généralement conçus pour des clients ayant une exigence élevée de véracité scientifique, ou du moins de fidélité. Même si les programmes sont conçus pour rendre des phénomènes complexes accessibles au grand public, et sont donc forcément des approximations ou des simplifications, il faut que les explications soient fondées sur des modèles scientifiques à jour.

Dans une situation idéale, les données et modèles sont fournis par le client. En réalité, les documents fournis sont souvent de trop faible qualité pour être utilisé dans la production d'animation. Il faut donc soit inventer les données manquantes (la fameuse vue d'artiste), soit s'appuyer sur d'autres sources, parmi lesquelles les données ouvertes (cf. section I.2.4) peuvent être salvatrices.

Ce chapitre explorera plusieurs questions de visualisation scientifique rencontrées lors de la production des projets *Antarctica* (cf. section III.1.2), dans lesquels les programmes audiovisuels avaient pour but la visualisation, particulièrement cartographique. Il analysera donc le rôle des données ouvertes dans la fabrication de ces programmes.

III.6.1 Les trajets des équipes

Conçue pour l'exposition du musée des Confluences (cf. section III.1.2), cette animation figure un planisphère à la surface duquel sont tracés les chemins parcourus par les deux équipes (hommes, et matériel) ayant participé à l'expédition, entre Paris et Dumont d'Urville, la base scientifique française en Terre Adélie (figure III.6.1). Les équipes font plusieurs escales sur leurs trajets, et les courbes décrivant les trajets doivent donc passer par ces points en Grande-Bretagne, aux Émirats arabes unis, en Australie et en Antarctique. Le planisphère doit être reconnaissable par le public, et proche des représentations cartographiques conventionnelles. La déformation doit être limitée et le nord en haut de la carte. La fabrication de ce plan* m'a conduit à explorer à la fois différentes projections de cartes, et la manière de les obtenir avec des programmes open source, en collaboration avec la direction artistique.

FIGURE III.6.1 – Le plan illustrant le trajet qu'ont suivi l'équipe de tournage. En cyan, le trajet des personnes ; en blanc, celui du matériel.

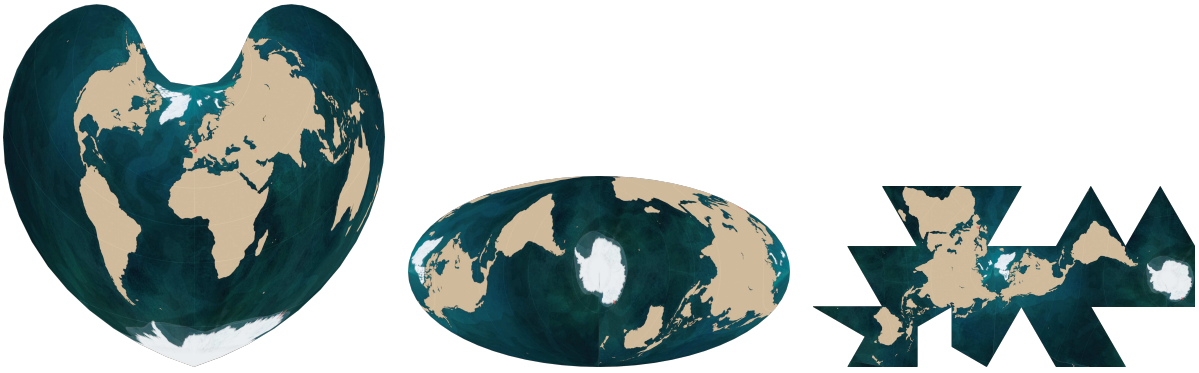
Des bases de données cartographiques sont disponibles sous licences ouvertes [1] (cf. section I.2.4). Pour ce plan, la direction artistique demandait seulement de disposer des contours des continents, aussi j'ai utilisé un simple fond de carte disponible sur Wikimedia Commons [66]. Cette carte a été rendue disponible dans le domaine public par son auteur [2], et on peut donc la réutiliser dans n'importe quel but. Il est indispensable,

1. Il n'existe pas de répertoire global de ces bases de données, et il faut donc, lors de ses recherches, vérifier systématiquement sous quels termes les données sont publiées.

2. Il s'agit d'une version modifiée d'une carte de la CIA, qui publie tout comme la NASA une partie de ses données dans le domaine public. Il est à noter que l'auteur est vraisemblablement états-unien.

quand on utilise des images ou données trouvées sur Internet, de vérifier sous quelles conditions elles sont publiées. Sur les sites de la fondation Wikimedia, cette information est disponible pour tous les médias, et certaines licences ne sont pas adaptées à une utilisation commerciale.

La carte originale utilise une projection equirectangulaire [3]. Pour essayer différentes projections cartographiques, j'ai écrit des scripts utilisant la bibliothèque* Python pyproj [56]. Elle permet la conversion de coordonnées d'une projection à l'autre (par exemple, passer d'une projection latitude-longitude à celle de Mollweide). Cette bibliothèque permet des manipulations complexes de données cartographiques.



(a) Bonne (b) Mollweide (c) Fuller

FIGURE III.6.2 – Différents essais de projections cartographiques utilisant la direction artistique

Pour déterminer la projection qui réponde le mieux à nos critères, j'ai collaboré avec le directeur artistique Eric Serre. Nous avons étudié plusieurs projections (figure III.6.2). Eric avait dessiné sur un fond de carte ses intentions artistiques, et ce design a été à chaque fois reprojeté pour obtenir des cartes aux formes diverses, mais avec la même direction artistique. Nous avons fait des essais d'animation de caméra pour mettre en scène le plan, et avons finalement opté pour la projection transverse de Mercator. C'est

S'il avait été français, il n'aurait pas eu le droit de publier une œuvre dans le domaine public, le droit d'auteur français protégeant obligatoirement les œuvres. Il aurait pu en revanche y associer une licence libre très permissive telle que Creative Commons CC0 1.0 universel (CC0 1.0) [63].

3. Aussi appelée latitude-longitude, c'est une projection très simple, pratiquement la plus basique, puisqu'elle fait simplement correspondre un degré de longitude ou de latitude à une unité de distance sur la carte.

une projection cylindrique, c'est-à-dire que la sphère est projetée sur la surface d'un cylindre qui l'entoure. Dans cette projection, contrairement à la projection de Mercator, très répandue, l'axe du cylindre se situe sur l'équateur et non dans l'axe du globe. En choisissant la rotation du globe le long de son axe, on peut faire tourner les continents autour des pôles pour les placer de manière reconnaissable.

Le plan, une fois terminé, devait recommencer en boucle pour les visiteurs de l'exposition. Le plan étant fixe, la première solution envisagée était simplement de faire disparaître à nouveau les courbes des trajets. Mais un des avantages de cette projection est que le haut et le bas se raccordent, puisque le cylindre se déplie dans une direction nord-sud. Nous avons donc pu faire boucler la séquence* dans un travelling vertical entre l'Europe et l'Antarctique. Cette solution n'était pas anticipée, mais semble rétrospectivement élégante et évidente, et permet de tirer pleinement parti des possibilités offertes par une représentation cartographique non conventionnelle.

D'un point de vue cartographique, il est très déstabilisant d'utiliser une projection inhabituelle parce qu'on ne sait plus comment appeler les directions sur la carte. Par exemple, si la carte est centrée sur le pôle Sud, on a envie d'appeler le bas, le sud ; la gauche, l'ouest, etc. Mais c'est faux, tous les points sont au sud. On se rend compte que la seule manière est d'utiliser des angles (70°S, 90°E), même de manière approximative. On est ainsi obligé de comprendre les latitudes et longitudes.

III.6.2 Le cœur

Ce plan* était une animation expliquant le mouvement de la banquise au cours de l'année : son expansion et sa fonte. Le nom du plan « le cœur » est une métaphore visuelle, puisque la banquise s'étend et se rétracte au cours de l'année. C'est également une métaphore sémantique, car il explique que la banquise fait partie des mécanismes essentiels de régulation du climat sur Terre.

La création de l'animation a consisté à utiliser des données précises de visualisation scientifique pour parvenir à un résultat qui est non seulement plaisant visuellement, mais

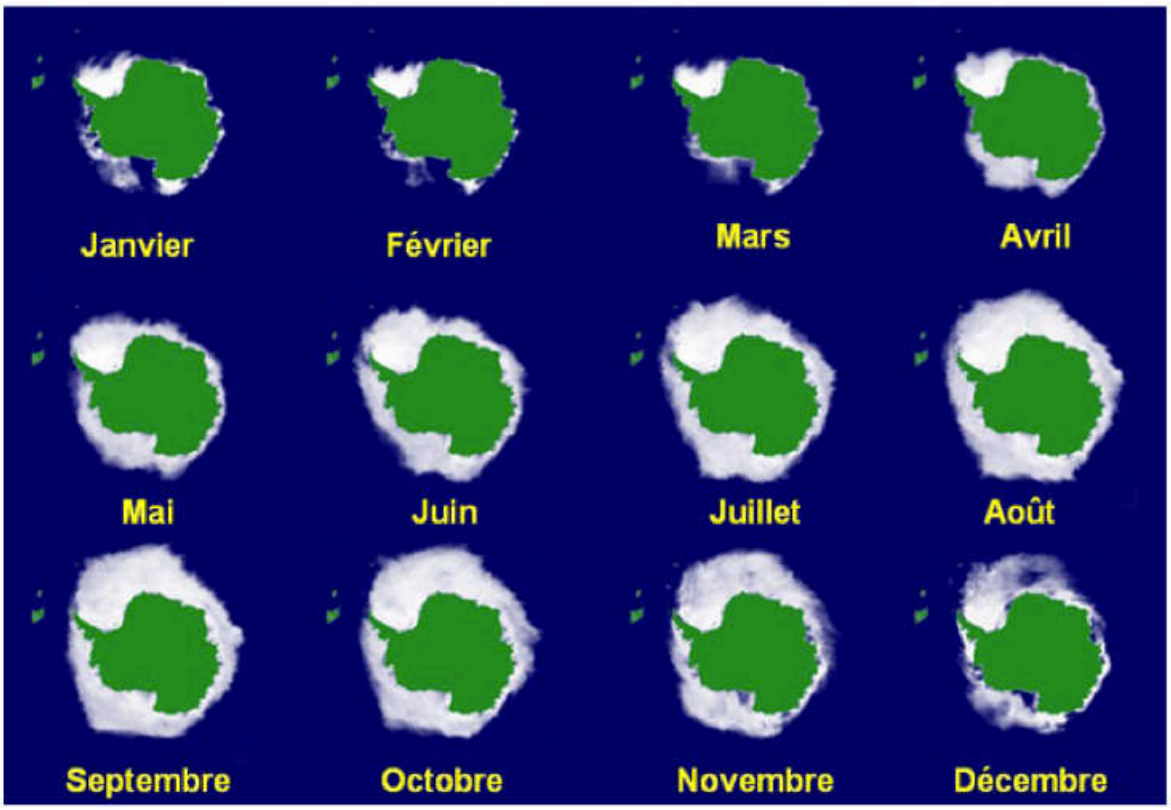


FIGURE III.6.3 – Les données fournies par le client, insuffisantes pour obtenir une séquence juste et détaillée.

beaucoup plus réaliste que si nous avions dû créer l'animation à partir de rien.

En effet, les données qui nous avaient été initialement fournies par le client (figure III.6.3) étaient des vignettes à basse résolution (environ 100 pixels) et non sourcées. Il y avait une vignette par mois de l'année, et l'animation que nous devions réaliser était censée être fluide, vivante et détaillée. Nous aurions pu détourer ces petites vignettes pour obtenir une forme grossière, et l'affiner en inventant des détails, mais cette démarche aurait conduit à des résultats fantaisistes et peu intéressants.

Il était plus logique de chercher une source de données plus détaillée, que nous pourrions réutiliser afin de créer notre version. Il a donc fallu se renseigner sur le sujet pour connaître assez de vocabulaire pour faire des recherches pertinentes. Une vidéo appropriée a été publiée en 2012 par la NASA (figure III.6.4), sous une licence permettant une réutilisation. Plutôt qu'un jeu de données directement extrait de mesures satellites,

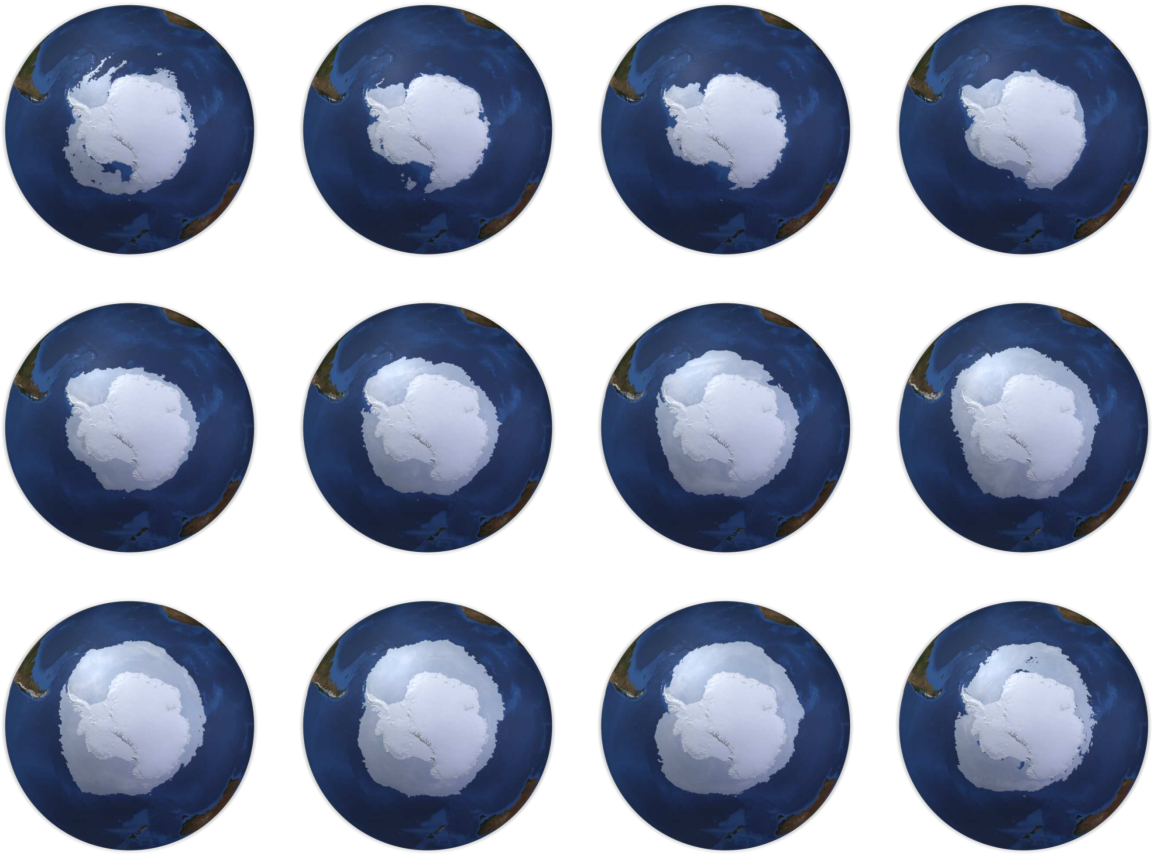


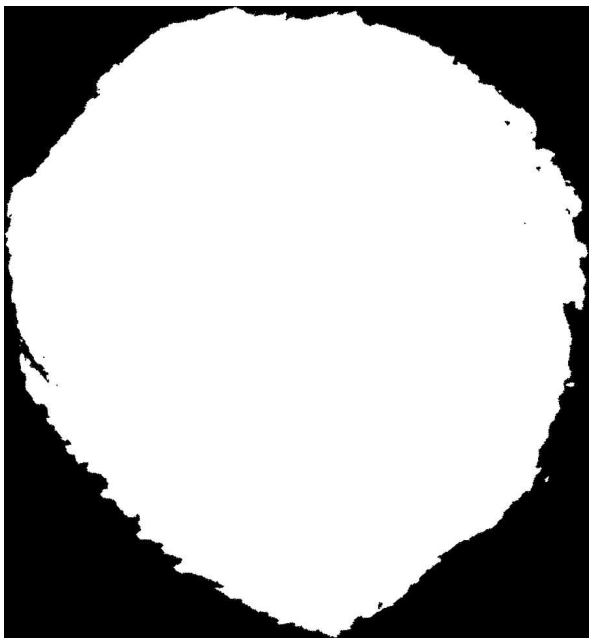
FIGURE III.6.4 – Images extraites de la vidéo de la NASA illustrant l'évolution de la banquise au cours de l'année 2010 [41].

il s'agit ici d'une visualisation scientifique, c'est-à-dire déjà une vidéo de synthèse présentant les données de manière compréhensible pour le public. Il a donc fallu, plutôt que de traiter des données sources, procéder en sens inverse en partant de la vidéo pour en extraire les données, et recréer dans un second temps une autre vidéo dérivée.

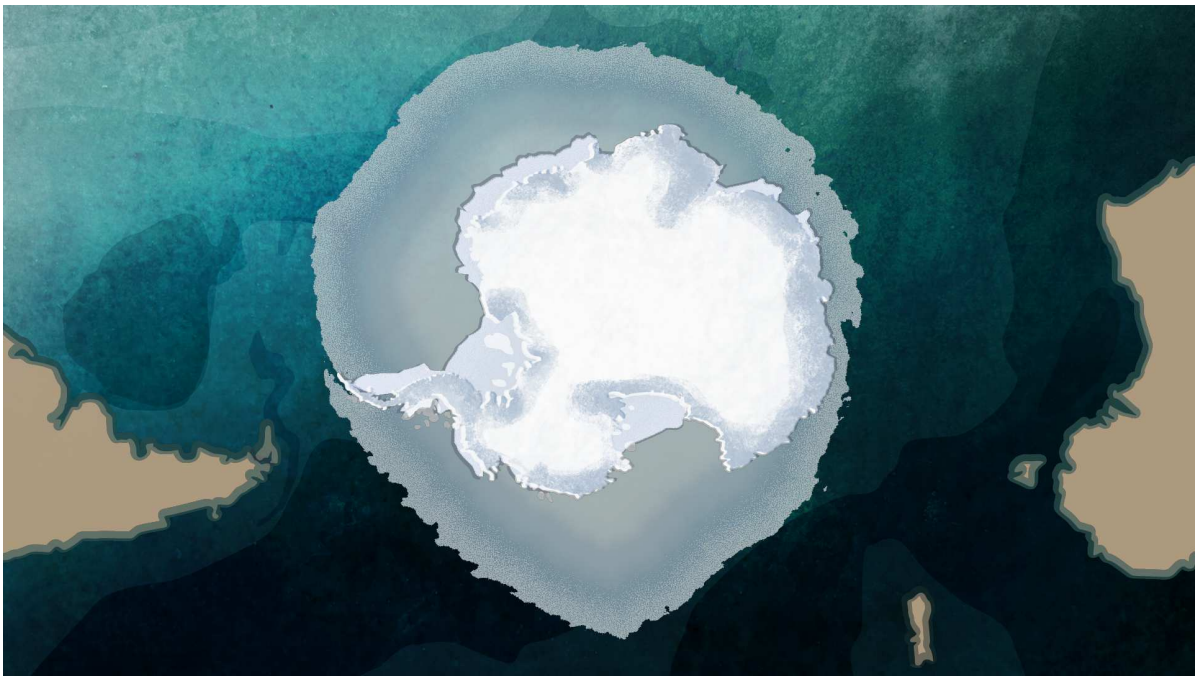
Pour cela, il a été possible d'utiliser une technique de compositing assez classique, à savoir un keying pour séparer la mer et la glace et obtenir un masque noir et blanc. Cette opération a été réalisée dans Natron (cf. section III.3.3), dont les outils de keying étaient très adaptés à ce type de plan.

La majeure partie du travail a ensuite été d'adapter la direction artistique à la séquence de la NASA (figure III.6.5), en retrouvant les couleurs et les effets de matière du fond, de la banquise et du continent, et les textes et repères géographiques. Pour cela

(a) Image originale



(b) Masque résultant du keying



(c) Résultat final

FIGURE III.6.5 – Antarctica : processus graphique pour la séquence de la banquise

aussi, Natron a été utilisé pour reconstruire le plan dans ses détails graphiques et son animation réaliste.

Comme pour la carte décrite à la section précédente, il est évident qu'avant de songer

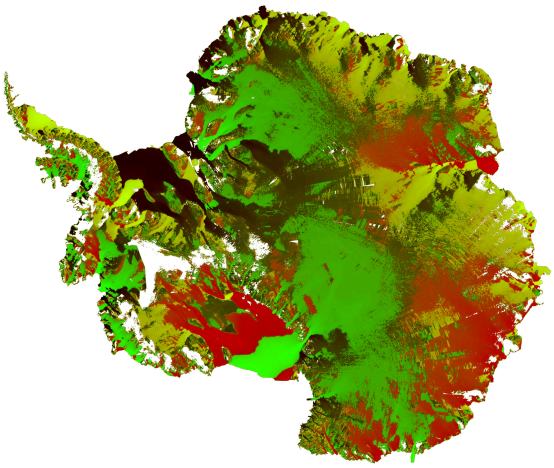
à réutiliser une vidéo existante, il est nécessaire de bien lire les conditions sous lesquels elle est publiée, en particulier si le détournement est autorisé à des fins commerciales ou pédagogiques, et le cas échéant, s'il faut citer des groupes ou individus précis au générique de l'œuvre dérivée.

III.6.3 L'écoulement de la banquise

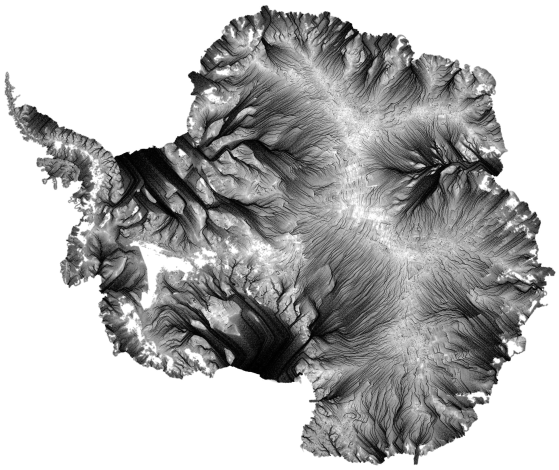
Ce plan* illustre la manière dont les banquises du continent s'écoulent lors de leur fonte annuelle. Le continent est plus ou moins recouvert de glace toute l'année, mais une partie fond l'été, et de l'eau s'écoule depuis les glaciers jusqu'à l'océan. L'animation devait donc visualiser de manière réaliste cet écoulement ressemblant à un système hydrographique avec des rivières et des fleuves sur toute la surface du continent.

L'équipe du film nous a fourni quelques documents, mais ils ne donnaient que très peu d'indications, et le client recommandait encore une fois « d'inventer » un mouvement sans référence précise, le but étant d'illustrer un principe général plutôt que des données exactes. Il fallait donc montrer les mouvements des glaciers sur toute la surface du continent, sans connaître particulièrement ni la topologie, ni l'hydrographie, ni aucune discipline qui aurait pu nous permettre de créer une animation, sinon véridique, du moins réaliste. C'est pourquoi, ici encore, j'ai commencé par me documenter, d'abord sur les phénomènes en jeu, puis rapidement sur les jeux de données disponibles et permettant d'utiliser un modèle scientifique afin d'être au plus près des connaissances actuelles sur le sujet.

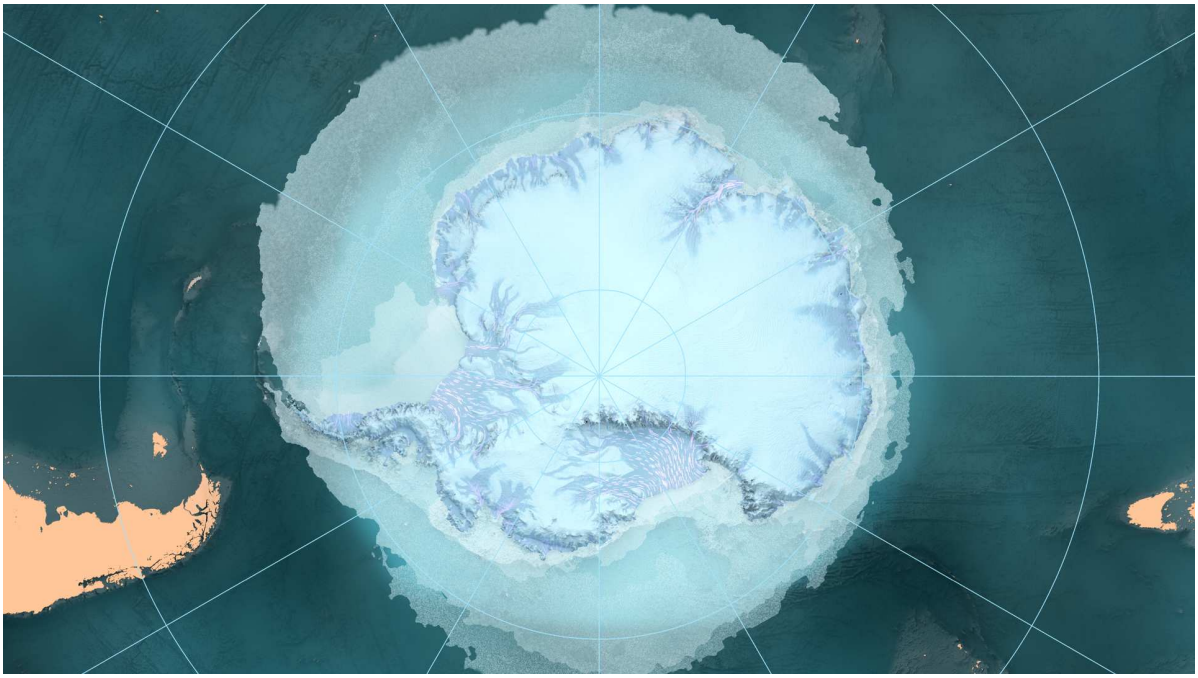
La NASA, au sein du *National Snow and Ice Data Center*, a publié les données d'une expérience mesurant précisément les mouvements des glaces antarctiques. [40] En vertu de la politique de données des agences gouvernementales états-uniennes, l'utilisation de ces données est libre, à la seule condition de citer les auteurs de l'étude [34]. Nous avons donc pu traiter ces données scientifiques à des fins d'illustration, et obtenir des images non seulement justes, mais également intéressantes esthétiquement du fait de l'interprétation que nous en avons faite.



(a) Visualisation des données sources



(b) Passe intermédiaire d'écoulement



(c) Résultat final

FIGURE III.6.6 – Antarctica : processus graphique pour la séquence de l'écoulement

J'ai commencé ce plan en trouvant et téléchargeant les données et en les préparant. Ces données sont des vecteurs de déplacement pour chaque point échantillonné sur le continent. J'ai pour cela examiné le format de données qui m'était inconnu, NetCDF, et écrit un script* pour les extraire vers un format plus facile à utiliser. Cette extraction

n'était pas très complexe, car le format NetCDF est simple à lire. Destiné à stocker des tableaux de données, il est auto-documenté, c'est-à-dire que l'en-tête contient les informations nécessaires : les différents tableaux contenus dans le fichier [4], les dimensions des tableaux, la projection cartographique utilisée, ainsi que des descriptions et des références. J'ai créé à partir de ce fichier une image TIFF non compressée contenant les mêmes données, mais plus facile à charger dans un script pour manipulation future et plus facile à visualiser qu'un long tableau texte (figure III.6.6a).

Une fois ces données extraites, mon collègue Duy Kévin Nguyen a traité le plan en s'occupant de la conception graphique et de la traduction en images. Duy, pour générer les images présentes dans l'exposition, a utilisé l'environnement de développement libre Processing, connu pour sa facilité de prototypage itératif. La méthode employée a été de créer un système de particules émises sur toute la surface du continent, et déplacées (advectées) par les vecteurs présents dans la carte. Puisque ces vecteurs représentent la quantité de déplacement en une année pour chaque point, il lui a fallu amplifier les valeurs afin d'obtenir un mouvement perceptible. Il a procédé en générant de nombreuses séquences d'images (figure III.6.6b) à partir de ces particules, qui devaient ensuite servir de masques colorisés et compositées pour obtenir le plan final (figure III.6.6c), par-dessus d'autres passes générées à partir de cartes d'élévation.

Cette manière de travailler a demandé de constants aller-retours entre une approche d'ingénieur, afin de comprendre, interpréter, analyser les données et leur signification, et une démarche de graphiste pour créer des images lisibles, belles et intéressantes. Du fait de l'utilisation de données ouvertes, elle demande une plus grande rigueur qu'une création arbitraire sans référence, mais offre aussi un résultat impossible à obtenir sans ces données.

4. Ici, il s'agissait des composantes horizontale et verticale des vecteurs de déplacement, exprimées en mètre par an, et d'une estimation de l'erreur pour chaque point. La précision est de 450 mètres, c'est-à-dire qu'un point du tableau représente un carré de 450 mètres de côté. Le tableau représentant le continent entier est un carré de 12445 points de côté, ce qui fait une quantité de données assez importante.

III.6.4 Cartes de l'Antarctique

Au cours des films, plusieurs sites sont montrés sur le continent. Or le public n'est pas familier avec l'Antarctique, et il a donc fallu créer des cartes afin de situer l'action, par exemple sur la base scientifique Dumont D'Urville et la péninsule antarctique. Ces cartes devaient avoir un rendu tirant vers le réalisme, et nous les avons fabriquées en utilisant les données d'élévation de la NASA (cf. section I.2.4.1). Des problèmes sont apparus, qui ont pu être réglé en partie grâce à la nature open source de Blender et à son modèle de distribution.

En effet, nous avons utilisé un système dans Blender permettant de créer un relief en fonction d'une map au moment du rendu, grâce à un déplacement (*displacement*) et à une subdivision adaptative. Cela signifie que la géométrie est plus ou moins subdivisée selon la distance à la caméra. À l'époque, ce système de « micro-déplacement » était en bêta, et donc indisponible dans la version stable de Blender. La fonctionnalité était en phase active de développement et des améliorations sont arrivées en cours de fabrication dans les dernières builds*.

La solution du micro-déplacement donne des images de haute qualité, mais le problème avec ce projet était le format d'image 4K UHD, c'est-à-dire de 3840×2160 pixels. Pour que l'image soit de bonne qualité, il fallait que le maillage soit très subdivisé pour que la taille des facettes soit inférieure ou égale à celle des pixels, sans quoi des artefacts seraient apparus. Il fallait donc calculer un grand nombre de pixels, et chaque pixel était long à calculer en raison de la précision du maillage, à tel point que pour certains plans* particulièrement longs, il est devenu manifeste que nous n'arriverions pas à calculer les images sur nos ordinateurs, même en distribuant les calculs sur toutes les machines, certaines étant même incapables de calculer ne serait-ce qu'une image.

Afin de pallier ce problème, nous avons fait appel à un service de rendu en ligne, Qarnot Computing. Le fonctionnement de ce genre de services est le suivant : le client envoie une scène à rendre à travers une interface dédiée, le calcul est effectué sur les serveurs du prestataire, et le client est notifié quand les images sont prêtes à être téléchargées. Il est intéressant de faire appel à un prestataire quand une entreprise a besoin

de scalabilité*, ou ne souhaite pas investir dans une ferme de rendu, dans le principe du *cloud computing*. Ceci dit, les sociétés qui fournissent ces services limitent généralement le nombre de logiciels et de versions disponibles au client, afin de faciliter la maintenance.

Or, nous avions besoin de calculer nos images avec le micro-déplacement, qui était indisponible dans la version officielle de Blender proposée par Qarnot. Nous avons contacté l'équipe de Qarnot afin de leur demander d'installer les builds récentes, non-officielles, de Blender, dont nous avions besoin. Ils ont consenti à le faire, en les intégrant dans leur système interne de gestion de calculs. Cette expérience était intéressante parce que non seulement nous pouvions utiliser des versions expérimentales du programme [5], mais nous avons pu demander à un prestataire d'en installer un grand nombre sur ses nœuds de calcul. Certes, leur service client était très réactif, mais c'est en partie la disponibilité de Blender qui leur a permis de nous aider à rendre le projet à temps. Les raisons juridiques de cette possibilité sont expliquées en section II.1.1.1.

5. Cette pratique n'est pas nécessairement des plus sages, car un programme bêta peut être instable, corrompre les données, dysfonctionner de diverses manières. Mais le logiciel libre la rend au moins techniquement et légalement possible.

Chapitre III.7

Acquisition et usinage 3D

Introduction

Bien que l'animation 3D utilise principalement des outils numériques, des liens se tissent depuis une dizaine d'années entre les processus de conception numérique et le monde physique, suite à la facilité d'accès aux techniques de numérisation 3D et de prototypage rapide [1]. Ces liens existent dans les deux sens : en passant des objets physiques à leur modèle numérique, et en fabriquant des objets physiques à partir de modèles informatiques.

Les studios ne se limitent plus, comme naguère, à utiliser la numérisation pour la capture de références photographiques pour les textures, mais s'étend à la numérisation complète d'objets volumiques complexes. L'utilisation du LIDAR et de la photogrammétrie pour numériser un décor de cinéma comme référence pour l'ajout d'effets spéciaux est devenue importante pour les grosses productions de cette industrie. Pour le film d'animation et le jeu vidéo, ces techniques sont elles aussi de plus en plus abordables et répandues, pour la création d'assets* notamment.

Dans l'autre sens, celui du numérique vers le physique, la CAO* trouve plusieurs

1. En particulier avec le mouvement *makers* et la création de nombreux fab labs. Des équipements comme les imprimantes 3D et les scanners 3D deviennent accessibles aux petites entreprises et même au grand public, et leur qualité s'est aussi considérablement amélioré au cours de la décennie 2010. L'open source a aussi joué un rôle important, notamment du côté du matériel.

usages pour l'animation. Par exemple, elle est souvent utilisée pour les productions de films en stop motion. Pour animer un visage, plutôt que de modeler chaque expression faciale, certains animateurs préfèrent préparer le visage sur ordinateur, construire les différentes positions, et ensuite les imprimer en 3D. Ces hybridations entre création numérique et monde physique peuvent créer de nouvelles formes d'expression, ou simplement simplifier la création des formes existantes [2].

La nature des projets sur lesquels j'ai travaillé au cours de cette thèse ne m'a pas permis d'explorer ces deux domaines (fabrication d'assets et animation stop-motion). J'ai en revanche participé à un projet composé de différents médias à vocations artistique et pédagogique, dont un des axes principaux était la numérisation et l'usinage de pièces volumiques : le musée de Lodève (cf. section III.1.3). Cette riche expérience a permis d'explorer ces domaines avec plusieurs cas d'étude complémentaires.

Afin de préciser les problématiques de numérisation et d'usinage 3D, j'analyserai donc le travail qui a été entrepris pour les différentes expositions du musée. Je détaillerai les problèmes soulevés par la réalisation de ces pièces, et les difficultés rencontrées en 2018 par le studio pour les résoudre avec des solutions libres.

Pour cela, j'exposerai d'abord certaines contraintes typiques de la création muséographique en volume, à la fois dans la place des œuvres au sein de la muséographie, et dans leur processus de fabrication. J'examinerai ensuite deux techniques ayant permis de modifier numériquement les objets fabriqués pour le musée, afin d'obtenir des résultats cohérents avec les souhaits de mise en scène. Je finirai par évoquer certaines des solutions libres existant en 2019 pour la photogrammétrie, c'est-à-dire la capture photographique des volumes.

2. Par exemple, le studio d'animation états-unien Laika est reconnu depuis *Coraline* (2009) pour ses films utilisant les techniques de prototypage rapide pour l'animation de ses personnages. Le studio a reçu en 2016 une récompense aux Oscars scientifiques (*Scientific and Engineering Awards*) pour ses innovations dans ce domaine.

III.7.1 Contraintes techniques pour la muséographie

Les contraintes d'élaboration d'un modèle 3D destiné à sa fabrication physique sont différentes de celles des assets* de films d'animation, particulièrement en matière de topologie (cf. section I.1.2.2). En effet, tandis que la topologie d'un personnage animé doit permettre de déformer précisément et de manière prévisible le volume, ici la pièce fabriquée sera entièrement statique. Il n'est donc pas besoin, par exemple, de subdiviser une surface plane, puisqu'une seule facette suffit à la décrire. En revanche, d'autres contraintes existent. Pour que le programme chargé de transformer le modèle 3D en instructions pour la machine qui fabrique la pièce (que ce soit une imprimante 3D ou une fraiseuse à commande numérique), il faut que le modèle soit « étanche », c'est-à-dire qu'aucune erreur de topologie ne soit présente dans le modèle. Il ne peut y avoir deux sommets au même point, ou deux arêtes ou deux faces se chevauchant, ou encore deux sommets reliés par deux arêtes différentes. Il ne peut pas non plus y avoir de trou dans la surface. Si toutes ces situations peuvent poser problème aussi pour le film, en particulier pour le rendu et les effets spéciaux, on peut souvent s'en accommoder, voir ne jamais les détecter. Ce n'est pas vrai de la CAO, où il faut s'assurer de les éliminer.

D'autres contraintes existent, d'ordre géométrique. Il faut modéliser selon les cotes prévues, et avec la précision demandée. Si pour un film, on allonge une partie d'un accessoire de quelques millimètres, rien de grave ne se produit. Si en revanche, on modélise une maquette quelques millimètres trop large, elle risque de ne pas entrer dans la vitrine où elle doit être installée.

Je vais à présent décrire différents types de travaux réalisés.

III.7.1.1 Réalisation de maquette schématique

La première pièce volumique réalisée pour le musée est la maquette du causse du Larzac, une formation géologique proche de Lodève. Elle présente une vue en coupe d'un plateau, avec en contrebas une rivière reliée à une grotte souterraine. Dans cette grotte, des habitants du néolithique entreposaient des poteries pour y recueillir de l'eau, d'où

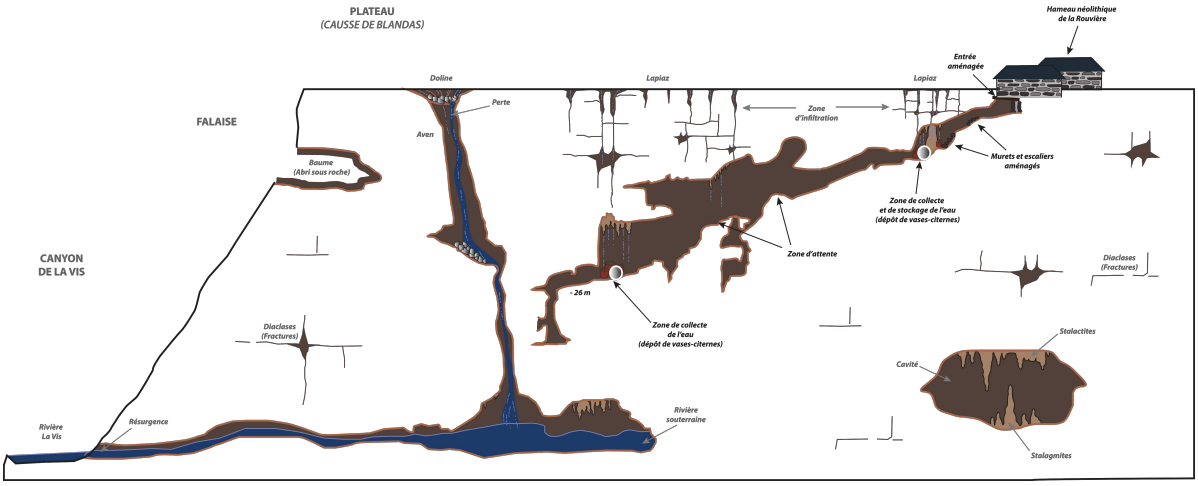


FIGURE III.7.1 – Schéma de la maquette dessiné par l'archéologue Noisette Bec.

son nom, la grotte citerne. Dessinée d'après un schéma (figure III.7.1) de l'archéologue chargée de l'exposition, Noisette Bec, la maquette a été sculptée en 3D dans Blender (figure III.7.2). Ce logiciel est parfaitement adapté pour une telle tâche, dans la mesure où il permet de modéliser des maillages avec un grand nombre de polygones, et où il offre des outils de sculpture performants, ainsi que des outils permettant de repérer les problèmes de topologie empêchant l'usinage.

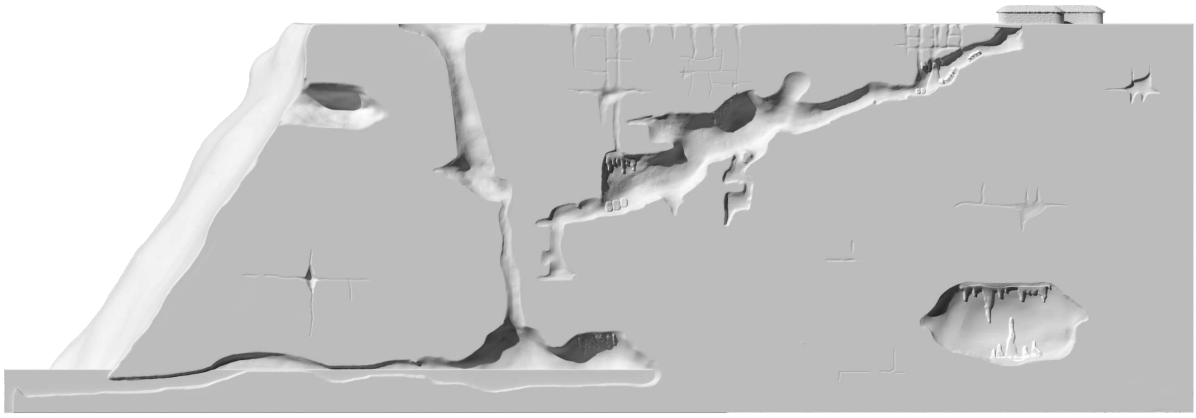


FIGURE III.7.2 – Rendu en élévation du modèle 3D de la maquette.

Le schéma était dessiné à la ligne, montrant la coupe de la grotte. La maquette est certes en volume, mais elle a été usinée avec une fraiseuse à commande numérique, et il était compliqué de creuser sur une grande profondeur avec cet équipement, et surtout d'avoir un volume en saillie, se détachant de la surface. Par exemple, on pourrait imaginer

que les cavités soient représentées creusées, et que les stalactites tombant du plafond se détachent de la paroi, mais l'entreprise chargée de la fabrication nous a précisé que ce serait trop complexe pour ce projet. La grotte a donc été modélisée en bas-relief, et les volumes sculptés strictement à partir du schéma, laissant peu de place à l'improvisation. À chaque étape de conception, la maquette était validée par l'archéologue, et elle l'a été également plusieurs fois par l'entreprise chargée de l'usinage, afin de s'assurer que ce dernier serait possible.

Figure III.7.3 – Maquette installée dans l'exposition.

III.7.1.2 Réalisation d'œuvres hybrides physiques-virtuelles

L'intérêt que peut avoir un studio d'animation à concevoir des pièces volumiques pour l'exposition d'un musée est qu'il peut mettre en œuvre ses compétences d'animation, et augmenter l'œuvre par ce médium. Cela peut se faire en projetant une animation sur l'objet, à l'aide d'écrans juxtaposés, ou même en créant une installation interactive. Ces

deux volets de l'œuvre peuvent être élaborés à partir des mêmes données volumiques, et créer donc un tout harmonieux.

Les Fées spéciales sont intervenues sur deux installations de ce type pour le musée de Lodève, la première utilisant des écrans, et la deuxième des projections directement à la surface de l'œuvre.

Animation : dalle des empreintes

FIGURE III.7.4 – Dalle fabriquée et installée dans l'exposition.

La première installation mettant en œuvre une pièce physique enrichie par de l'animation se situe dans le parcours géologique du musée. Une salle entière est consacrée à une dalle posée au sol, à la surface de laquelle on peut voir des traces de pas d'animaux préhistoriques (figure III.7.4). L'attrait de cette installation est la présentation simultanée de médias diffusés dans la salle et montrant également la dalle.

De chaque côté de la salle, des écrans jouent une animation des deux animaux préhis-

FIGURE III.7.5 – Image extraite du film de la dalle des empreintes. Les traces sur le sol sont les mêmes que celles visibles sur la dalle, et le point de vue de l'écran sur la dalle correspond à l'animation.

toriques qui ont laissé leurs empreintes [3]. Les points de vue des écrans correspondent à leur placement par rapport à la dalle (figure III.7.5). Dans cette installation multimédia, les empreintes au sol devaient donc correspondre à une séquence fictionnelle imaginée par les scientifiques, mais fidèle aux connaissances actuelles sur la locomotion de ces créatures. La recréation numérique de la dalle prenait ainsi tout son sens : les films et l'objet physique étaient conçus en même temps à partir des mêmes données 3D. Le processus de fabrication de la dalle sera étudié dans la section III.7.2.1.

Utilisation de données ouvertes : cartes projetées

La deuxième installation consiste en des cartes animées de la région que les Fées spéciales ont conçues afin d'illustrer des phénomènes géologiques et archéologiques. Elles prennent la forme de cartes en relief placées sur des tables dans l'exposition. Sur ces

3. Ces animaux étaient un dimétrodon, reptile préhistorique à la démarche plutôt lourde, et un éryops, qui nageait à la surface de l'eau à une époque où le sol était immergé, et dont les pattes griffues pouvaient l'atteindre.

FIGURE III.7.6 – Carte projetée d'archéologie dans l'exposition.

supports sont vidéoprojetés des programmes animés depuis un projecteur caché dans le sous-plafond. Le public peut ainsi se pencher pour observer les animations et lire les explications comme sur un écran, mais en relief. La société a été chargée de concevoir les cartes, c'est-à-dire les modéliser et produire les animations projetées. Tout comme pour la dalle, le même modèle 3D a été utilisé pour fabriquer l'objet physique et pour créer les animations projetées dessus. Ainsi, l'œuvre exposée forme un tout cohérent.

Comme les programmes cartographiques du musée des Confluences (cf. section III.6.4), la modélisation utilise des données ouvertes, publiées par l'agence spatiale états-unienne NASA (cf. section I.2.4.1) et par le projet OpenStreetMap. Contrairement à cette première exposition où on voyait le continent antarctique dans son ensemble, il fallait ici des données de beaucoup plus haute précision, car la zone géographique montrée sur les cartes est centrée sur le Lodévois, et mesure quelques dizaines de kilomètres de côté.

Ici encore, les données d'élévation ont été utilisées directement dans Blender pour

déformer la surface d'un plan et obtenir le volume de la table, en réglant seulement l'épaisseur de la table et l'intensité du déplacement, qui est exagérée pour que le relief soit bien perceptible. Cette présentation d'animations en relief demande une assez bonne justesse dans le volume, mais offre une grande liberté artistique, puisqu'on touche au *projection mapping*[4], et qu'indépendamment de la carte relief, assez belle en soi, tout l'intérêt de la pièce est la vidéo qui est projetée à sa surface, et les phénomènes qui y sont illustrés.

L'image projetée sur la carte a aussi été en partie générée à partir de données ouvertes. Ainsi, le fond de carte constitué des villes, des cours d'eau et de la mer, a été extrait de la base de données ouverte OpenStreetMap (cf. section I.2.4.2), tandis que des courbes de niveaux ont également été générées à l'aide de la carte d'élévation de la NASA.

L'intérêt majeur d'utiliser ces données a été de respecter une correspondance stricte entre la surface imprimée et les images générées. De plus, il a été possible d'automatiser la création d'une partie des cartes est, plutôt que de les dessiner manuellement. Pour donner une idée de l'intérêt que cette automatisation représente, on peut la comparer avec la méthode choisie par la directrice artistique qui a conçu l'identité visuelle des cartes, Marie Saby. Dans les dessins de concept qu'elle a créés, elle est partie d'un fond de carte, dont elle a décalqué les courbes de niveau, et colorisé chaque pallier, ce qui représente un travail de titan. En utilisant les données ouvertes, j'ai pu obtenir un résultat de qualité équivalente en quelques minutes, une fois que les données d'altimétrie étaient en place.

III.7.2 Transformations numériques d'objets réels

Comme dans beaucoup de musées, la majorité des collections n'est pas exposée mais dans les réserves, étudiées par les scientifiques et rarement ou jamais montrée au public. Il peut y avoir plusieurs raisons à cela : fragilité des pièces, impossibilité de concevoir

4. Le *projection mapping* est une technique consistant à utiliser des objets physiques comme support de projection pour des animations. Plutôt qu'un simple écran plat, il utilise les reliefs de la surface, les met en valeur ou en retrait par l'image portée. L'œuvre qui en résulte dépend donc autant de son support physique que de l'image projetée.

des vitrines adaptées, ou simplement un nombre trop grand de pièces pour pouvoir tout exposer en même temps.

Ces différents problèmes peuvent être réglés par la reproduction des pièces en réserve. Les techniques de numérisation et d'usinage 3D sont parfaitement adaptées à cette application. De plus, elles permettent de modifier numériquement les pièces avant usinage, pour pouvoir recomposer les pièces manquantes, combler les trous, voire même modifier complètement la forme de la pièce tout en conservant son esprit. Plusieurs pièces que nous avons fabriquées étaient soit des adaptations de pièces physiques dans la collection du musée, soit des reproductions.

J'exposerai dans cette section les processus d'acquisition et de traitement de deux objets conservés dans les collections du musée de Lodève.

III.7.2.1 Recomposition en 2D d'un relief 3D

FIGURE III.7.7 – Moulages de la dalle prêtés par le musée pour la numérisation, dans l'atelier des Fées spéciales

La dalle des empreintes (cf. section III.7.1.2) du musée de Lodève est une pièce de grande dimension installée au sol dans une des salles du musée. On peut y voir des empreintes de pas fossiles d'animaux préhistoriques. Ces traces de pas ont été retrouvées

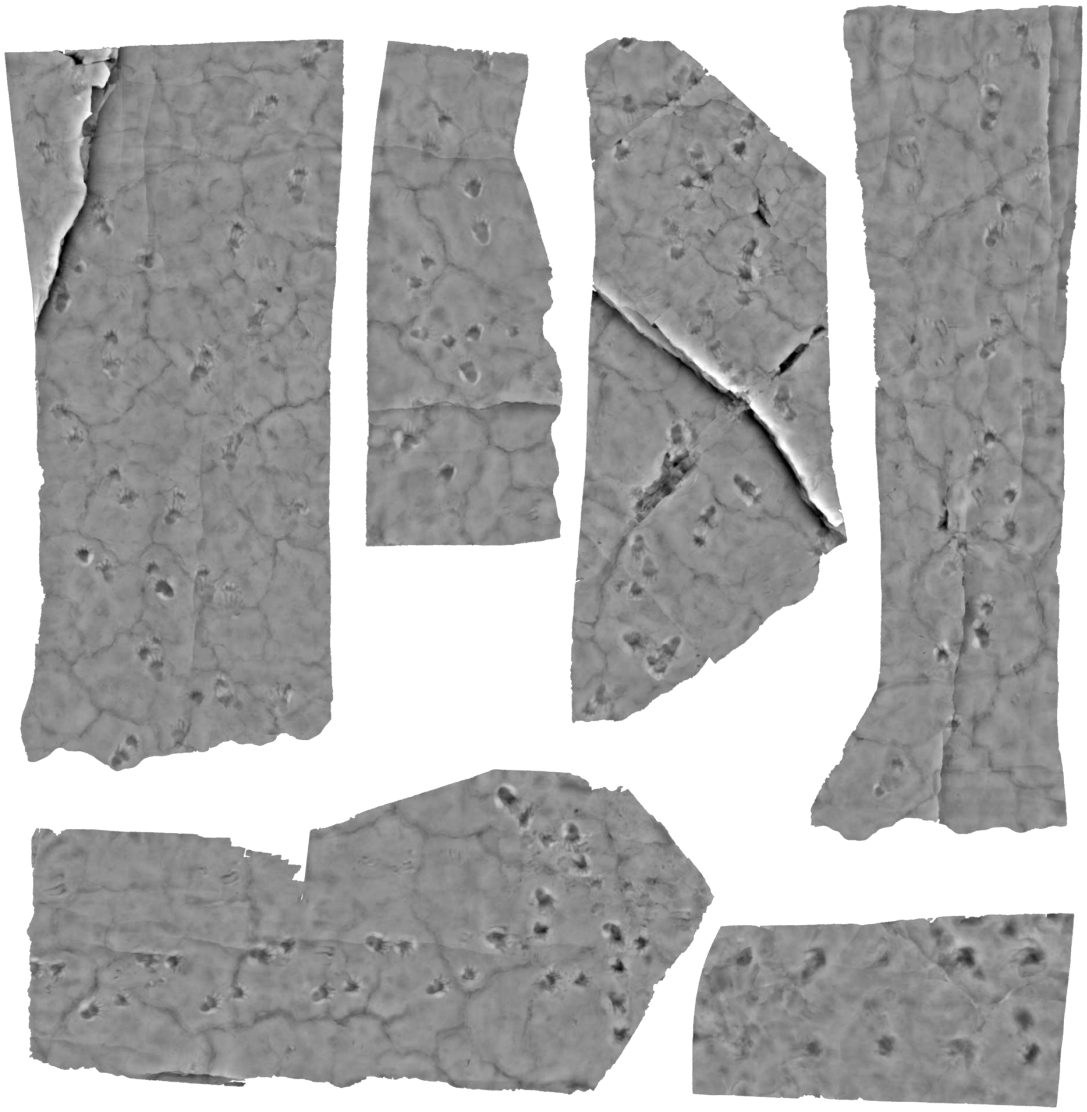


FIGURE III.7.8 – Scans des moulages sous la forme de cartes de hauteur.

lors de fouilles sur un site industriel, et des moulages ont été faits sur place et conservés dans les collections du musée (figure III.7.7). Le site ayant depuis été recouvert, les moulages sont les seuls éléments accessibles à l'équipe du musée. Mais ces moulages sont fragiles et incomplets : seules certaines portions de la dalle ont été levées. Or, le musée souhaitait reconstituer une dalle de 40 m^2 pour l'exposition. Les Fées spéciales ont donc été chargées de concevoir la dalle complète à partir des fragments disponibles. Pour cela,

nous avons numérisé les fragments existants et recomposé l'objet les réordonnant et en comblant les parties manquantes.

Afin d'obtenir une numérisation précise des morceaux de la dalle, nous avons essayé plusieurs techniques. La plus simple à mettre en œuvre est celle de la photogrammétrie, puisque le seul équipement à posséder est un appareil photo, un ordinateur puissant, et un programme spécialisé. Nous souhaitions utiliser pour cela des logiciels libres, et avons donc essayé plusieurs programmes en amont de la production. Les résultats étaient acceptables, mais le processus complexe à l'époque, du fait que tous les programmes existants devaient nécessairement s'exécuter en ligne de commande, sans disposer d'interface graphique. Ces programmes sont abordés dans la section III.7.3.

Au final, les contraintes de production nous ont fait revoir notre méthode. Étant débutants en photogrammétrie, et n'ayant pas eu assez de préparation pour être certains d'avoir un bon résultat dans un temps très court, nous avons utilisé un scanner 3D à main. Celui-ci a très bien fonctionné, mais le programme fourni par le constructeur est propriétaire. Il existe bien des logiciels open source, et des scanners dont le matériel même est open source, mais ils n'étaient pas adaptés à la numérisation d'objets aussi grands. Même si nous avions trouvé un tel matériel, le temps de préparation manquait pour l'acquérir ou le fabriquer, et pour apprendre à s'en servir correctement.

Une fois les morceaux de dalle scannés, j'ai abordé leur recomposition en vue d'obtenir un objet fidèle aux demandes narratives des films projetés, à la muséographie et à la fidélité scientifique. Pour cela, il aurait été possible de procéder en 3D, avec des techniques de sculpture, mais il était beaucoup plus efficace de créer une carte d'élévation à partir de chacune des plaques (figure III.7.8), de les recomposer dans un logiciel de peinture, et finalement de les réappliquer sur le maillage 3D à l'aide d'un displacement. En effet, l'objet fabriqué est en relief, mais il n'y a pas de volumes superposés : tout le relief peut être décrit en plan.

Pour obtenir ces cartes d'élévation, j'ai fait un rendu des scans avec un matériau permettant d'obtenir la hauteur de chaque point en niveaux de gris (figure III.7.8). Ce rendu devait être impérativement fait dans un format d'image permettant de stocker

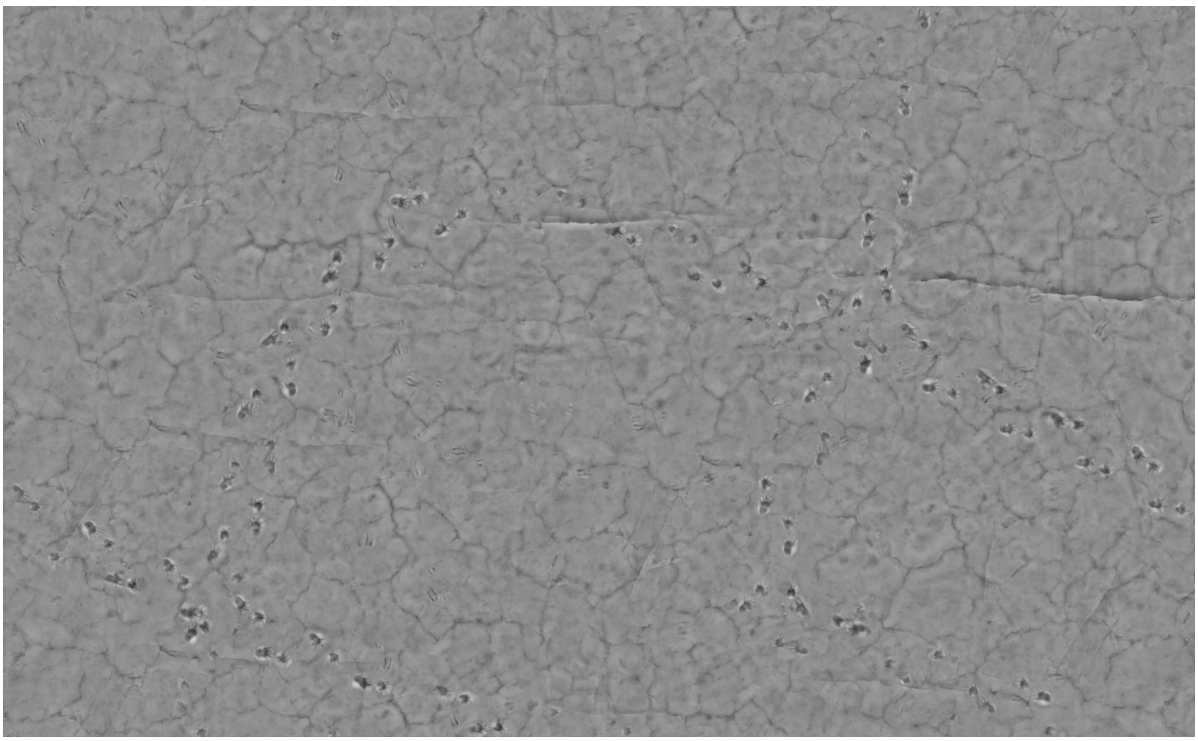


FIGURE III.7.9 – Dalle recomposée numériquement à partir des morceaux.

plus de 8 bits par canal, car une précision standard est insuffisante pour représenter le niveau de détail nécessaire, et un PNG 8 bits, par exemple, aurait fait apparaître des artefacts dûs à la perte de précision. Il fallait également que l'espace colorimétrique n'utilise pas de fonction de transfert, pas de gamma, mais soit simplement linéaire. J'ai donc utilisé le format openEXR, avec des nombres flottants de 16 bits, dont la précision est suffisante pour cette application.

J'ai d'abord essayé d'utiliser le logiciel libre de peinture Krita pour recomposer la dalle. Le logiciel offrait déjà toutes les fonctions nécessaires (lecture et écriture d'EXR 16 bits, tampon de clonage pour combler les trous, calques, filtres, etc.), mais ses performances étaient insuffisantes. En effet, nous avions déterminé la résolution du fichier d'après celle des fraiseuses qui allaient fabriquer la pièce, soit 1 500 pixels par mètre. Il fallait pouvoir traiter la dalle dans son intégralité, et le fichier faisait donc environ 85 mégapixels. De plus, le fichier comprenait un grand nombre de calques, et les opérations de peinture étaient trop lentes dans ces conditions sur Krita. Le studio disposant

de quelques licences de la suite de création d'Adobe, j'ai testé avec Photoshop. Les performances en peinture étaient assez faibles sur un fichier de cette envergure, mais plus acceptables que sur Krita.

Une fois le dessin de la dalle validé figure III.7.9, à ce stade le document de travail n'était encore qu'une image plane représentant le relief, et l'équipe de fabrication avait besoin de fichiers 3D dans un format standard et aux bonnes dimensions. Afin de générer ces fichiers, j'ai appliqué l'image comme un déplacement dans Blender de façon à générer un relief, puis découpé de nouveau l'objet en douze parties, tel que le prévoyait le plan de fabrication.

III.7.2.2 Sculpture numérique

De la même manière que pour la dalle précédemment évoquée, les collections du musée contiennent de nombreux os d'animaux préhistoriques, dont un spécimen assez complet de caséidé. Le squelette fossile de cet animal était très fragile et il était impossible d'exposer l'original aux visiteurs. De plus, il manquait des morceaux à beaucoup d'entre elles. Il a donc été décidé de les numériser, de les réparer virtuellement, de les fabriquer, et de les peindre d'après les couleurs et matières des os originaux, afin de donner l'illusion qu'il s'agissait effectivement des vrais os, tout en les gardant protégés dans la réserve.

Pour ces pièces, dont la précision devait être bien plus grande que pour la dalle en raison de leur petite taille, l'équipe a fait appel à une entreprise spécialisée dans la numérisation 3D. Celle-ci a utilisé un scanner laser et a livré les fichiers 3D. Puisque le travail était sous-traité à un prestataire, il n'y avait pas de choix possible des logiciels et méthodes de numérisation.

L'utilisation de logiciel libre pour l'acquisition est dans ces cas limitée aux seuls formats d'échange négociés avec le sous-traitant. En l'occurrence, le choix par défaut est le format STL [5], qui est assez simple et ancien, et très utilisé dans le domaine de la CAO* pour transmettre des objets 3D. Ses capacités sont assez limitées par rapport aux formats d'échange pour le film ou le jeu vidéo, mais il peut être lu et écrit dans la

5. Abréviation de *stereolithography*, du nom d'une technique d'impression 3D.

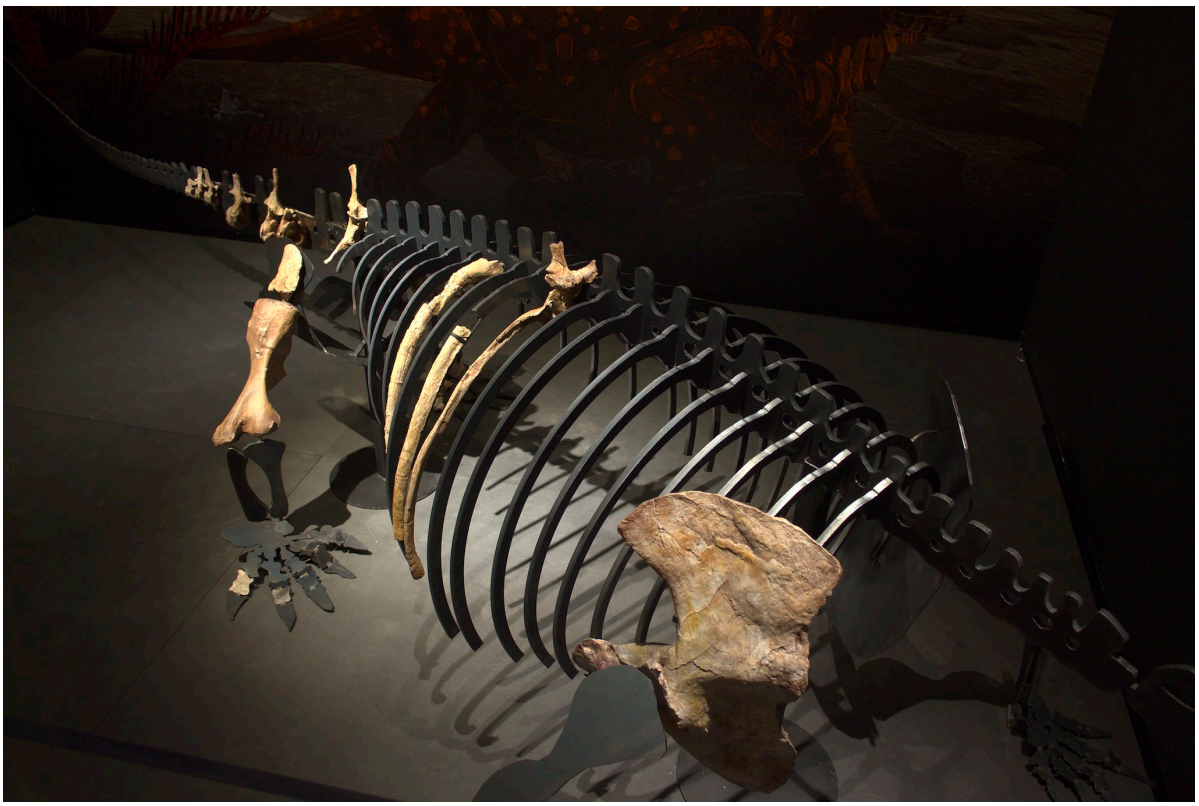


FIGURE III.7.10 – Squelette d'animal préhistorique dans l'exposition.

plupart des programmes, sa spécification est ouverte (cf. section II.3.2), et il est donc nécessaire de l'utiliser pour communiquer avec des équipes spécialisées dans la CAO.

Une fois ces scans 3D obtenus, leur traitement a pu commencer. Duy Kévin Nguyen, qui travaillait également au sein de l'équipe technique des Fées spéciales, les a importés dans Blender afin de reconstituer les parties manquantes et de nettoyer les modèles numérisés (figure III.7.11). Les techniques de sculpture numérique utilisées pour cela sont très proches de celles qui ont cours dans les studios d'animation pour la création d'assets. Le but étant de retrouver les parties manquantes tout en restant réaliste, Duy a appliqué une symétrie sur des os existants lorsqu'ils existaient, et le reste du temps sculpté les modèles d'après des références photographiques.

Malgré les possibilités existantes, la démarche d'utiliser des logiciels libres est parfois compromise par des problèmes de performance ou de stabilité. Ces problèmes peuvent

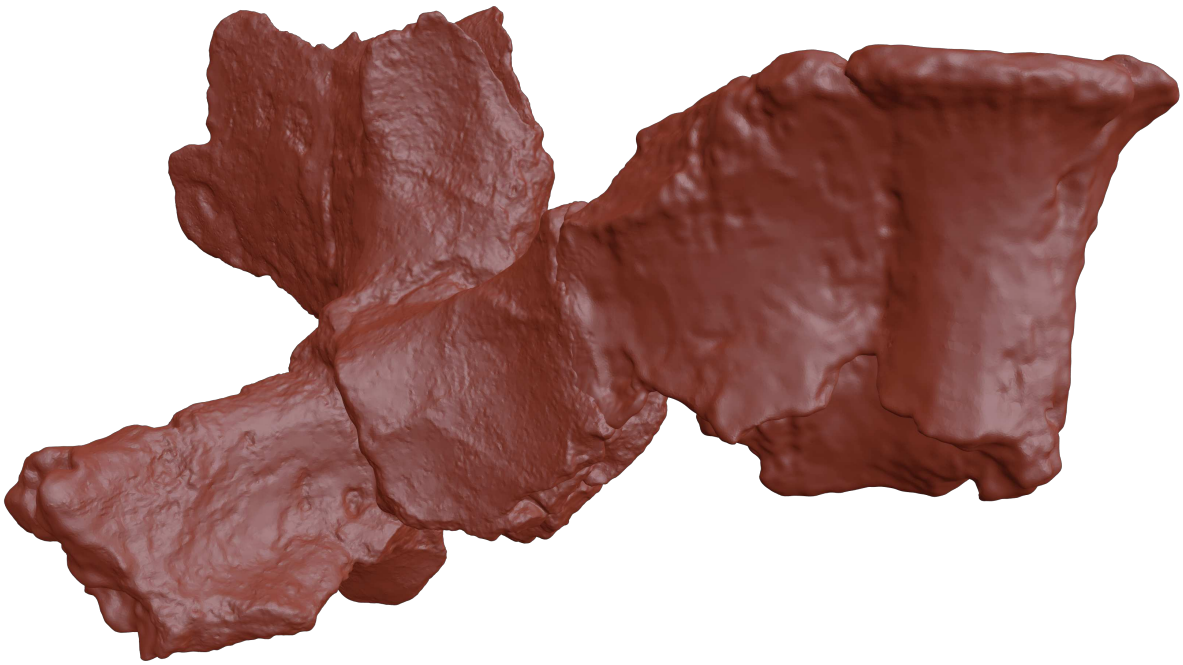


FIGURE III.7.11 – Un os sculpté dans Blender à partir du scan. On voit les parties reconstituées, plus lisses.

être résolus en découpant le travail de plus de manière efficace, ou en contactant les développeurs et en leur faisant part des problèmes rencontrés, mais ces deux solutions prennent plus de temps et leur issue est plus incertaine que celle d'utiliser un programme déjà connu, et pour lequel on a déjà des licences.

Dans tous les cas, un travail préparatoire de veille et de test est indispensable pour être prêt dès le début du projet. Ce travail prend un temps qui n'est pas toujours compatible avec les contraintes de projet auxquels sont soumis les studios d'animation. Ce fait demeure un des obstacles à l'adoption des logiciels libres dans les plus petites structures, dont les équipes techniques disposent d'un temps encore plus limité.

III.7.3 Photogrammétrie libre

La photogrammétrie est un ensemble de principes, d'algorithmes et de techniques permettant de faire une numérisation d'un objet physique à partir de plusieurs photo-

graphies de cet objet. Les principes ont d'abord été développés pour la cartographie, et fondés sur des prises de vue aériennes, bien avant l'invention de l'informatique. Elle a trouvé de nouvelles applications depuis le début du millénaire, et s'est rapidement répandue dans l'animation et les effets spéciaux, avec la facilité de prise de vue des appareils photo numériques et l'augmentation de la puissance de calcul des ordinateurs personnels. Aujourd'hui, il est très facile de s'équiper pour faire de la photogrammétrie.

La photogrammétrie est techniquement constituée de plusieurs étapes, exécutées séquentiellement : établir des points de correspondance entre les images, retrouver les emplacements des points de vue dans l'espace, estimer la distance des points depuis la caméra pour chaque image, projeter ces points dans un espace 3D, interpoler ce nuage de point pour obtenir un nuage de points dense, éliminer les erreurs de mesure, créer un maillage à partir du nuage de point, et projeter la couleur des photogrammes sur ce maillage. Il existe des variations sur ce processus, et plusieurs algorithmes existent pour chaque étape.

De nombreux programmes propriétaires performants existent pour faire les calculs permettant d'obtenir un maillage 3D à partir d'une série d'images. Le but de cette section n'est ni de décrire en détail chaque étape, ni chaque programme, mais d'exposer quelques exemples de solutions libres de photogrammétrie, la logique qui les sous-tend, et l'utilisation que peuvent en faire les studios.

III.7.3.1 MicMac

MicMac [16] est un programme de photogrammétrie composé d'un ensemble de commandes exécutables avec ou sans interface graphique, dont chacune correspond à une étape du processus exposé plus haut. Principalement destiné aux scientifiques, ce programme est difficile d'utilisation, et je n'ai pas réussi à en obtenir de résultats satisfaisants lors de la phase de veille préalable au projet. Il est néanmoins intéressant à mentionner.

En effet, j'ai pris connaissance de ce programme en 2017 lors d'une conférence de

Maxime Seguin, qui était à l'époque archéologue auprès de l'INRAP [6] de Montpellier, sur l'application de la photogrammétrie à l'archéologie [24]. Dans cette conférence, il présentait MicMac en exposant sa méthodologie de numérisation lors des fouilles d'un site archéologique proche de Montpellier, le Mas Rouge.

D'après M. Seguin, ce programme destiné aux scientifiques n'est pas facile d'utilisation, mais il donne des résultats d'une grande précision s'il est bien utilisé. L'utilisation d'un logiciel libre par les scientifiques a le grand avantage qu'on peut en lire le code source pour connaître ses algorithmes, contrairement au logiciel propriétaire Agisoft Photoscan que la plupart des archéologues utilisent et qui a popularisé la photogrammétrie parmi eux, mais dont il est impossible d'évaluer l'erreur.

Le conférencier a également mentionné l'intérêt du logiciel libre dans la conservation de données à moyen et long terme. Selon lui, l'utilisation d'un format de données propriétaire et fermé ne donne pas la certitude de pouvoir le conserver, le rouvrir et l'exploiter à long terme. Cette problématique n'est pas propre aux domaines scientifiques. Elle est fondamentale pour n'importe quelle organisation, et elle est explorée plus en détail dans la section II.1.2.

Ces conclusions de scientifique ne sont pas entièrement pertinentes pour un studio d'animation. En effet, un levé architectural ou archéologique doit être le plus exact possible pour pouvoir être utilisé dans des recherches scientifiques, mais dans la création numérique il suffit que les résultats soient visuellement crédibles, ce qui signifie que de petites erreurs de mesure ont peu de conséquences. En revanche, la facilité d'utilisation et la performance d'un programme sont beaucoup plus importantes, dans la mesure où la photogrammétrie n'est pas l'outil principal des graphistes, et qu'ils ne peuvent pas passer longtemps à paramétrer un programme pour obtenir des résultats très fins. Or, MicMac est précisément à l'opposé. Son interface, en ligne de commande, est très difficile à prendre en main, et il faut très bien connaître le programme pour obtenir des résultats corrects. Il n'est donc pas adapté à une utilisation occasionnelle de la photogrammétrie pour l'animation.

6. Institut National de Recherches Archéologiques Préventives

III.7.3.2 Colmap, openMVG, openMVS, MVE, Meshlab, etc.

Comme je l'ai expliqué en introduction de ce chapitre, le processus de photogrammétrie est composé de multiples étapes, et chaque étape peut être traitée par plusieurs méthodes et algorithmes. Aussi, il existe tout un écosystème de logiciels libres de photogrammétrie, chacun étant conçu pour une étape précise. Mais pour avoir un processus complet, il faut que les différentes étapes soient compatibles, c'est-à-dire que les données sortant d'un programme soient interopérables* avec le suivant. C'est le cas de plusieurs programmes de photogrammétrie libres. Ces programmes peuvent être utilisés ensemble, et certaines parties du processus peuvent être traitées par l'un ou l'autre programme. Certains programmes, comme Colmap et Meshlab, ont une interface graphique, tandis que d'autres ne s'exécutent qu'en ligne de commande. Cette approche est conceptuellement intéressante, parce que sa modularité permet de choisir le programme le plus adapté pour chaque étape. Si un algorithme ou une implémentation plus performante est publiée, il suffit de remplacer une partie du pipeline* photogrammétrique, sans devoir jeter tout le reste.

Cependant, bien que ce processus soit plus facile à mettre en œuvre que MicMac, il est nécessaire de scripter un minimum pour obtenir un pipeline facile d'utilisation, ce qui ne les met pas à la portée de tous les utilisateurs. En revanche, un pipeline entièrement scriptable peut être extrêmement utile à des utilisateurs avancés, en particulier pour traiter de grandes quantités de données.

Colmap est dans cet écosystème le seul programme entièrement dédié à la photogrammétrie à fournir une interface graphique optionnelle, et incluant toutes les étapes [7]. Il est donc assez accessible, avec des réglages corrects par défaut et un processus simple, tout en proposant une grande souplesse et de nombreux réglages.

7. L'autre programme disposant d'une interface graphique est MeshLab, qui est beaucoup plus polyvalent et permet de très nombreuses opérations sur des maillages et nuages de points.

III.7.3.3 Meshroom

Meshroom est un logiciel libre de photogrammétrie développé depuis 2010. Il a suscité beaucoup d'intérêt dans la communauté de l'animation lors de sa sortie publique en 2018, car c'est le premier à offrir des algorithmes performants derrière une interface graphique facile à prendre en main par des non-spécialistes. C'est en fait un regroupement de différents programmes développés au sein de plusieurs laboratoires publics français, tchèque et norvégiens, et du studio Mikros Image.

Comparé aux autres programmes sus-mentionnés, il a pour particularité d'avoir été conçu spécifiquement pour les besoins industriels des studios d'animation, en particulier concernant l'interface graphique, qui suit des conventions habituelles pour les logiciels de création 3D : présence d'une vue 3D, pipeline photogrammétrique paramétrable représenté par une interface nodale, export de caméras animées, de nuages de points colorés et de maillages texturés au format standard Alembic, paramètres par défaut équilibrés.

Meshroom est en fait une interface graphique pour un framework* de photogrammétrie, Alicevision, et les deux projets sont développés en même temps. Il est donc possible, tout comme les autres programmes, de scripter un processus différent de celui prévu par défaut dans l'interface graphique. De plus, il est à noter qu'Alicevision se fonde en partie sur plusieurs des programmes libres en ligne de commande mentionnés, en particulier sur openMVG.

Nous avons fait un bref tour d'horizon des solutions libres de photogrammétrie qui pouvaient être utilisées par les studios d'animation. Au cours de la décennie 2010, on a vu rapidement se développer des programmes dédiés efficaces et accessibles. L'arrivée de Meshroom, co-développé par un studio d'animation et d'effets spéciaux, témoigne de l'intérêt croissant des studios pour ces techniques, qui les utilisent sur de nombreux projets de films.

III.7.4 Conclusion

Ce chapitre consacré à la numérisation et à l'usinage d'objets physiques a mis en évidence la possibilité de faire reposer un pipeline sur le logiciel libre, en dépit de la difficulté d'utilisation de certaines solutions. Comme dans d'autres domaines, un pipeline reposant sur des logiciels libres doit être activement développé par les studios, ce qui demande généralement des ressources initialement plus élevées qu'avec un équivalent propriétaire clef en main.

Il y a dans tous les cas un fort potentiel pour les studios d'animation souhaitant s'intéresser à ces techniques, dans la mesure où d'une part, la convergence est en train de s'établir entre les techniques, et où d'autre part ils sont les mieux à même de proposer des œuvres hybrides mêlant l'image animée à l'installation physique (muséographie, création volumique, vidéoprojection).

Conclusion de la troisième et dernière partie

Cette dernière partie a permis d'explorer la conception et la constitution d'un pipeline libre dans le studio d'animation Les Fées spéciales, à travers certains des projets qui ont été le support de cette réflexion. Nous avons vu que s'il existait des applications performantes permettant de répondre aux besoins de certains départements, d'autres étaient encore lacunaires.

Cependant, plus important que cet état des lieux des logiciels à un instant précis, il a été intéressant d'examiner la manière dont les studios peuvent être moteurs dans le développement de leurs propres outils. Ces outils peuvent être des solutions complètes à un problème donné, comme l'exemple de Storymatic (cf. section III.2.1.4), ou des éléments pouvant être réutilisés dans d'autres contextes, des bibliothèques ou frameworks (cf. section III.5.1.2) ou des add-ons (cf. section III.2.2). Ils sont souvent en premier lieu uniquement destinés aux productions internes d'un studio, mais peuvent également à terme être mises à disposition de tous, comme l'a été l'outil de création de pantins (cf. section III.3.1).

Nous avons également étudié les pratiques d'utilisation de données ouvertes, et comment elles peuvent être utiles aux studios dans la création de certains types de programme, en particulier à visée scientifique.

Enfin, nous avons fait une incursion dans les techniques d'acquisition et d'usinage 3D, permettant de passer du réel au virtuel, et inversement. Cette hybridation est de plus en plus accessible en utilisant des logiciels libres de capture, de modélisation 3D et

de CAO*, et les studios d'animation peuvent donc être amenés à diversifier leur activité, sans pour autant bouleverser leur pipeline.

Conclusion générale

Cette thèse menée en entreprise a permis d'examiner l'utilisation et le développement du logiciel libre ou open source dans l'animation 3D. En étudiant les programmes et les manières de les utiliser, j'ai pu dégager des conclusions selon deux axes importants. Le premier axe, et le plus fondamental, s'est intéressé aux changements que peuvent apporter l'utilisation et le développement de logiciel libre dans le secteur de l'animation, et au regard que peuvent porter les studios sur les moyens de production que constituent les logiciels. Le deuxième axe est celui de la possibilité de mise en œuvre d'un pipeline libre, c'est-à-dire les programmes que les studios peuvent utiliser aujourd'hui dans le cadre de leurs productions, et les éléments qu'ils doivent considérer pour diffuser leurs propres solutions internes en open source.

Contexte de recherche

Une première étape de cette thèse a été d'exposer le contexte de recherche, qui est la rencontre entre deux mondes : le cinéma d'animation et le logiciel libre.

L'animation

Le cinéma d'animation présente un environnement de production particulier, car il mêle des processus de haute technicité à une grande expertise artistique. Le contexte professionnel de ces productions est généralement une entreprise appelée un studio d'animation. Cette thèse s'intéressant à l'utilisation professionnelle des logiciels, j'ai expliqué les particularités de ce type de structure.

En particulier, j'ai évoqué le studio dans lequel j'ai effectué la majeure partie de ma thèse, la SCOP Les Fées spéciales. Ce jeune studio a choisi de mettre un accent particulier sur la R&D*, c'est-à-dire sur le développement de solutions techniques destinées à la fabrication des images. J'ai montré comment mon rôle au sein du département technique de l'entreprise m'a permis d'explorer l'ensemble de la chaîne de fabrication d'un studio, structurée sous la forme de son *pipeline**.

Pour bien comprendre les enjeux que peut représenter l'utilisation de logiciels libres pour la production, j'ai décrit un pipeline* typique d'un petit studio, c'est-à-dire les différentes étapes permettant de passer du concept à l'œuvre finie, et découpées en départements.

Le logiciel libre

La deuxième facette de cette thèse est l'étude des logiciels libres, une catégorie de programme définie par des concepts juridiques complexes, une histoire riche, et des valeurs idéologiques qu'il est important d'appréhender pour comprendre la démarche des créateurs de logiciels libres, et celle que peuvent aussi adopter les utilisateurs.

J'ai donc évoqué la création des logiciels libres en réaction à la fermeture et à la marchandisation progressives des programmes informatiques dans les années 1970 et 1980, en particulier l'œuvre de Richard M. Stallman, un développeur états-unien qui conçut les premières licences de logiciels dites *libres*, ainsi que des programmes diffusés selon les termes de ces licences (cf. section I.2.1).

J'ai analysé les licences elles-mêmes dans une approche de propriété intellectuelle (cf. section I.2.2), la partie des lois gouvernant la protection des logiciels. J'ai évoqué comment la licence logicielle, un contrat entre l'auteur et l'utilisateur, spécifiait les droits accordés à ce dernier, et comment les licences libres détournent ce principe pour accorder beaucoup plus de droits que ne l'envisageait une lecture plus classique de la propriété intellectuelle. En particulier, j'ai évoqué les quatre libertés qui constituent la définition même du logiciel libre et de l'open source : utiliser, étudier, modifier et distribuer le code source informatique.

Enjeux

Le cœur de cette recherche se situe dans la confrontation de ces deux sphères d'activité, pour dégager ce qui caractérise l'utilisation du libre par les studios d'animation. Les interactions entre le logiciel libre et la production d'animation sont complexes, c'est pourquoi j'ai décomposé mon analyse selon trois critères complémentaires : juridiques, économiques, et techniques (cf. partie II).

Juridique

En s'appuyant sur le droit d'auteur, ou copyright dans ses équivalents anglo-saxons, les développeurs de logiciels libres donnent à leurs utilisateurs quatre libertés, qui entrent dans la définition du logiciel libre comme du logiciel open source. Pour un studio, ces quatre libertés (cf. section II.1.1) présentent en théorie de nombreux avantages.

Le studio peut utiliser le programme à sa guise, sans être restreint par un contrat qui le priverait de certains types d'utilisations, sur autant de machines qu'il le désire, dans tous les pays, et sans limitation de durée.

Il peut étudier le code source du programme et vérifier qu'il se comporte comme prévu. Il peut apprendre comment il fonctionne pour réutiliser ses algorithmes dans d'autres programmes. Dans certains cas, il peut même réutiliser directement du code.

Si les équipes rencontrent un dysfonctionnement, elles peuvent le résoudre par elles-mêmes ou engager quelqu'un pour le faire. Elles peuvent modifier le programme pour mettre en œuvre des fonctionnalités manquantes nécessaires aux projets en cours.

Même dans le cas où le studio ne disposerait pas d'une équipe R&D dédiée, les TD* ou les graphistes eux-mêmes peuvent être encouragés à faire aux développeurs des rapports de bugs*, qui sont souvent traités rapidement – selon les capacités de développement du projet.

Du fait que les licences libres donnent un droit d'utilisation illimité, la disponibilité du programme n'est pas soumise aux décisions mercantiles de l'éditeur. Contrairement aux programmes propriétaires, un logiciel libre qui ne serait plus développé par ses développeurs originaux peut continuer d'être utilisé et maintenu par une communauté

d'utilisateurs et de développeurs (cf. section II.1.2).

Si l'utilisation de logiciels libres ne requiert pas de contrepartie, les utilisateurs sont encouragés à y contribuer à leur tour, à la mesure de leurs ressources et de leurs compétences. Cette contribution peut être financière ou technique. Les studios peuvent ainsi, en accord avec les communautés de développeurs et d'utilisateurs, s'impliquer directement dans le développement de leurs logiciels, en écrivant du code, en rapportant les bugs et problèmes de conception, en traduisant les interfaces, en complétant la documentation, ou en publiant leurs outils internes.

Économique

Le logiciel libre n'est pas nécessairement gratuit. En fait, les notions de liberté et de gratuité sont orthogonales. Ceci dit, les logicels libres sont aussi gratuits la plupart du temps, en vertu du fait que chacun puisse librement redistribuer les programmes, d'après la définition même des licences libres, et qu'il soit donc difficile aux éditeurs de se garantir un monopole commercial (cf. section II.2.1).

En première approche, il peut donc être financièrement intéressant pour les studios d'utiliser les logiciels libres, qui sont généralement gratuits. Une licence peut être redistribuée, ce qui signifie qu'un logiciel libre peut être installé sur un nombre arbitraire de machines, sans devoir négocier l'achat ou la location de licences lorsque la taille de l'équipe fluctue, offrant donc la plus grande souplesse aux entreprises.

Cependant, j'ai montré que le prix des logiciels n'est pas seulement celui qui est attaché aux licences : les studios peuvent toujours payer pour du support technique, du développement prioritaire, ou simplement faire des dons aux communautés de développement pour financer le développement des programmes.

Dans un contexte d'entreprise, la gratuité apparente s'avère vite fausse, en particulier quand une équipe R&D doit être engagée pour faire fonctionner le logiciel et l'adapter aux processus de fabrication (cf. section II.2.2).

Même dans le cas où le studio paie le développement des logiciels libres, soit par ses salariés, soit par des sous-traitants, ces développements peuvent profiter à tous les

autres utilisateurs, puisque le logiciel libre constitue un bien commun. Ainsi, les studios peuvent mettre en commun leurs ressources pour développer des solutions libres plutôt que d'essayer de conserver un secret industriel, en recommençant de zéro ce qui a souvent déjà été fait (cf. section II.2.3).

Usages

En plus de ses particularités économiques et juridiques, le logiciel libre peut créer de nouveaux usages au sein des studios d'animation.

J'ai ainsi montré que certains logiciels libres étaient non seulement très performant, mais créaient des fonctionnalités novatrices sans équivalent chez leurs équivalents propriétaires. Il faut garder à l'esprit que cette observation n'est pas généralisable, ni nécessairement imputable au fait que qu'il s'agisse de logiciels libres, mais elle est néanmoins utile pour montrer que le modèle du logiciel libre permet de développer des programmes performants.

J'ai évoqué deux exemples de fonctionnalités intégrées au cours des dernières années au logiciel d'animation Blender : le Grease Pencil, un outil de dessin animé dans un espace 3D, qui permet une hybridation des techniques d'une manière jamais vue dans un logiciel public ; et Eevee, un moteur de rendu utilisant les techniques de rendu « temps réel » des jeux vidéo, pour offrir un rendu interactif de haute qualité (cf. section II.3.1).

Un autre aspect primordial pour les studios est celui de l'interopérabilité et de la standardisation (cf. section II.3.2). Il désigne la capacité de différents logiciels à être utilisés ensemble, en particulier par le biais de formats de fichiers ouverts, dont les spécifications sont accessibles à tous, et qui peuvent être lus et écrits depuis différents programmes. Pour les studios d'animation, cela signifie que plusieurs logiciels peuvent être utilisés dans une même chaîne de fabrication, et les données passer facilement de l'un à l'autre. Dans l'animation, il existe plusieurs formats d'échange importants, comme Alembic et OpenEXR, développés au sein de studios et rendus open source pour en généraliser l'usage. Ces formats sont ouverts et leurs spécifications publiques, mais ils comportent aussi des bibliothèques* logicielles leur permettant d'être intégrés dans n'importe quel

logiciel de création, qu'il soit libre ou propriétaire.

Enfin, j'ai abordé la participation d'un studio au développement de logiciels libres (cf. section II.3.3), qu'ils soient initiés par le studio et publiés ensuite, ou déjà existants. Les logiciels libres tirent leur force de leur communauté, et les studios disposent souvent des compétences leur permettant de participer utilement au développement, puisqu'ils écrivent souvent des outils destinés à leurs productions internes. Cette collaboration communautaire n'est pas nécessairement facile, et j'ai aussi montré les situations qui peuvent freiner les studios, en particulier quand les besoins ou les méthodes de développement de la communauté ne s'aligne pas avec ceux du studio.

Un pipeline libre : possibilités et limites

Parmi les logiciels libres existants dans le domaine de l'animation numérique, plusieurs ont démontré aujourd'hui être des alternatives viables à des programmes plus courants. Entre le système d'exploitation GNU/Linux*, les logiciels de création Blender et Krita, et jusqu'au lecteur vidéo DJV View (cf. section III.4.1) ou au logiciel de storyboard Storyboarder (cf. section III.2.1.3), des alternatives existent en 2019 pour de nombreuses tâches. Dans une démarche de développement collectif, un nombre croissant de studios adopte également des outils de gestion d'assets et de suivi de production libres, comme Kabaret et CGWire (cf. section III.5.1). Il faut aussi observer que de nombreux composants libres sont utilisés pour créer d'autres logiciels, y compris propriétaires, tels que les bibliothèques OpenEXR (cf. section II.3.2.1), ou le langage de script Python, et que ces composants sont même devenus indispensables aux studios comme aux éditeurs.

Cependant, certaines étapes de travail importantes n'ont à ce jour pas d'alternative libre crédible, à une échelle industrielle ou même artisanale. Parmi ces tâches, on trouve en particulier le compositing et le motion design, pour lesquels le logiciel Adobe After Effects est pratiquement incontournable (cf. section III.3.3.1).

Il est important toutefois de nuancer ces premières conclusions, car une telle liste de logiciels, utilisables ou pas, équivalents ou non à des solutions établies, n'est valable qu'à un moment précis. Elle peut s'avérer caduque rapidement, en raison de la vitesse à

laquelle les modes changent dans le milieu.

Ainsi, le logiciel libre de montage vidéo Olive [50] existe depuis 2018 et au moment où j'écris ces lignes, début 2020, il commence à faire parler de lui. Si son rythme de développement et d'adoption continue, il pourrait vite devenir référence dans son domaine. Il pourrait aussi cesser d'être développé du jour au lendemain et tomber dans l'oubli.

Un autre exemple, celui de Natron (cf. section III.3.3.2), a montré que des ressources suffisantes pouvaient mener à la création d'un programme performant dans le domaine du compositing.

Ce même exemple a montré, de manière plus intéressante, que si le prix des logiciels libres est souvent nul, son développement a un coût. Les utilisateurs, et en premier lieu les entreprises utilisant du logiciel libre, ont donc intérêt à financer ce développement en accord avec les développeurs, afin d'éviter qu'il ne se voie arrêté après quelques années (cf. section II.2.3).

Le logiciel libre, une approche différente de la production

Une conclusion découlant de cette étude a été que l'utilisation quotidienne de logiciels libres pour la production n'est pas intrinsèquement différente des logiciels propriétaires. Blender est très différent de Maya, mais pas plus que ne l'est Houdini, un autre logiciel propriétaire.

Quoiqu'instructif, faire une liste de logiciels libres utilisables en production a un intérêt limité, car le logiciel libre présente des différences fondamentales par rapport au logiciel propriétaire : le premier voit le logiciel en tant que bien commun, tandis que le deuxième le considère comme un bien de consommation.

Les données ouvertes, un pas vers la culture libre

Le logiciel libre et open source est maintenant établi depuis des décennies dans d'autres secteurs que celui de l'animation, et commence à pénétrer celui-ci. D'autres types d'informations, de contenus, de « propriété intellectuelle » font l'objet de mouvements similaires.

Ainsi, le mouvement de l'open data prend son essor (cf. section I.2.4). De plus en plus d'organisations publiques ou privées voient dans ce modèle un moyen de créer de la richesse partagée, en mettant leurs données à disposition de tous à l'aide d'outils juridiques spécifiques, de façon à créer un bien commun, sur lequel tous puissent s'appuyer pour créer. Dans certains cas, comme OpenStreetMap, ces bases de données sont établies de manière collaborative, et chacun peut les enrichir.

Les studios peuvent s'appuyer sur ces bases de données ouvertes pour créer leurs œuvres, c'est-à-dire créer de la richesse à partir de ce bien commun. Tout comme pour le logiciel libre, les studios peuvent également, et ils ont intérêt à le faire, contribuer dans leurs capacités à ces bases de données, pour que le fruit de leur travail puisse être exploité par d'autres individus qui comme eux, ont utilisé le travail d'une communauté.

Ouvertures

Cette thèse ouvre à plusieurs perspectives de recherche, en particulier les licences de libre diffusion (cf. section II.1.4), un autre type de contrat prévu pour diffuser, non plus les logiciels spécifiquement, mais tout type de contenu protégé par les lois de la propriété intellectuelle, tel que textes, sons, images, polices d'écriture, et vidéos. Ces licences ont permis la création de projets tels que Wikipédia, dans lesquels tous les contenus sont réutilisables par tous pourvu qu'ils respectent les termes de la licence.

Dans le domaine de l'animation, certaines initiatives existent, comme les films de la Blender Foundation et le projet Morevna, mais ces initiatives sont encore marginales. En effet, ce type de publication est très peu connu, et il existe une difficulté à trouver des modèles de financement fiables et pérennes pour la production. Néanmoins, dans une société où le contexte économique, social et culturel est en transformation rapide, ces modèles peuvent représenter des alternatives importantes à la diffusion de masse de quelques gros acteurs, qui demeure aujourd'hui la norme.

Un autre aspect du logiciel libre, d'une importance capitale mais pas toujours pris en considération, voire parfois méprisé, est le rôle social que peut jouer le logiciel libre, en permettant en particulier aux utilisateurs, qu'ils soient salariés ou indépendants, de

s'approprier leurs outils, leurs moyens de production, pour mieux s'émanciper. Cet aspect est très politique, et c'est la raison pour laquelle dans beaucoup de communautés open source, il est ignoré, voire décrié : il devrait s'effacer devant une prétendue neutralité politique, ou devant la simple performance technique des programmes. Bien sûr, l'étude politique n'est pas le sujet de cette thèse, mais le contact que j'ai eu avec le logiciel libre m'a fait prendre conscience que cet aspect était au cœur de la question, et que les principes sous-tendant le logiciel libre pourraient avoir une influence sur le cinéma d'animation.

Par exemple, l'animation française bénéficie de nombreuses subventions publiques, qui lui permettent souvent de survivre. Par ailleurs, un mouvement se développe actuellement, en particulier au sein de la communauté du logiciel libre, dont le but est de demander que tout programme financé par le public soit également disponible pour le public, c'est-à-dire publié sous licence libre. Cette revendication s'étend à la recherche privée financée par des fonds publiques, qui protège souvent ses fruits par des brevets. Cela pourrait également s'étendre à l'animation, c'est-à-dire à la fois aux solutions techniques développées par les studios avec des subventions publiques, mais au-delà, aux contenus audiovisuels. Ces questions de propriété intellectuelle et de biens communs sont aussi bien pertinentes pour la technique que pour la création artistique, et il serait bon que les producteurs de contenu comme leurs consommateurs s'y intéressent.

Glossaire

add-on Un add-on, mot anglais signifiant à peu près « ajout », aussi appelé plug-in, « qui se branche », et parfois greffon en français, est un ajout de fonctionnalité à une application hôtesse principale. C'est donc en soi un programme informatique, mais qui ne peut exister sans être *ajouté* à un autre programme. Un add-on est typiquement écrit en utilisant l'API* du programme hôte, et on le considère souvent de ce fait comme un script*. La distinction est que l'add-on forme un tout cohérent, intégrant des fonctionnalités liées entre elles, et distribuées ensemble. L'add-on peut typiquement permettre d'ajouter de nouvelles fonctions et opérations, de modifier l'interface, de créer des possibilités non prévues par les développeurs du programme hôte. 43, 68, 103, 121, 122, 124, 127, 164, 167, 168, 178, 185, 198, 270

amont En développement logiciel, on parle d'amont et d'aval lorsqu'une version dérivée d'un programme est créée. C'est ce qui arrive notamment en cas de fork*, qui est un procédé normal dans le développement open source. Les sources amont sont celles d'origines, celles d'aval, les versions dérivées. Plusieurs dérivations peuvent se succéder, et une version pourra alors avoir à la fois une source en amont, et plusieurs descendants en aval. Si une version amont adopte les changements de sa version aval, on dit qu'ils sont reversés en amont. En anglais, on parle d'*upstream* et *downstream*. 77, 247

API Une API (Application Programming Interface) est une interface de programmation applicative. Il s'agit initialement de la spécification d'un ensemble de fonctions et de protocoles permettant de simplifier le développement logiciel en faisant abstraction de l'implémentation logicielle ou matérielle. Dans cette thèse je donne à

l'acronyme une définition légèrement différente, celle d'une bibliothèque permettant le développement de scripts au sein d'une application. De la même manière que l'interface graphique expose à l'utilisateur les fonctionnalités d'une application, l'API permet d'utiliser l'application via du code. Concrètement, il s'agit d'une bibliothèque dans un langage de programmation. Cette bibliothèque offre des classes et fonctions permettant manipuler les données dans le programme. On peut s'en servir pour faire des scripts, c'est-à-dire des groupes d'opérations, ou bien des outils, qui étendent les capacités du programme hôte. Par exemple, l'API Python de Blender, `bpy`, permet d'automatiser des tâches en applicant plusieurs opérations à la suite, ou de créer de nouveaux outils dans l'interface graphique, tenant compte des entrées de l'utilisateur. 55, 67, 90, 111, 122, 153, 156, 191, 244, 251

asset Un asset (qu'on peut traduire littéralement en français par « actif », mais qui est plus justement appelé un « élément ») est un ensemble de données informatiques réutilisables pour un projet de film ou de jeu vidéo. Par exemple, un décor, un personnage, un accessoire, sont des assets, et chacun sera constitué de modèles 3D, de textures, de rigs, d'animations, etc. 32, 37, 94, 150, 190, 212, 214

bibliothèque Une bibliothèque est un type de programme informatique. C'est un ensemble de routines réutilisables, permettant de manipuler un type précis de données, ou de répondre à un problème particulier. Contrairement aux applications ou aux scripts, une bibliothèque n'est pas faite pour être utilisée directement, mais pour être appelée depuis un autre programme. L'intérêt est que le développeur d'applications puisse s'appuyer sur du code partagé, souvent bien testé, et de développer donc plus vite, sans recommencer de zéro. Les améliorations apportées à une bibliothèque, quand elles ne cassent pas la compatibilité avec les versions précédentes, bénéficient à tous les développeurs qui l'utilisent. La publication de bibliothèques est essentielle pour les écosystèmes logiciels, car elles permettent de réutiliser du code pour différentes applications et donc de réduire les coûts de développement. C'est une manière d'obtenir une architecture modulaire et non monolithique. 77, 82, 85, 86, 88, 90, 103, 115, 120, 168, 183, 186, 202, 239, 270

build Une build est une version binaire, compilée d'un programme. À partir d'un même code source, un développeur peut générer différentes builds, selon les outils et options de compilation. Lorsqu'on installe la version officielle d'un programme, il s'agit d'une build, mais il est aussi possible, lorsqu'on dispose des sources, de compiler une build. Certains projets proposent également des builds quotidiennes à télécharger pour tester les dernières fonctionnalités ou correctifs, en-dehors de la distribution habituelle. 210

cache Un cache est une manière de stocker des données, indépendamment de la manière dont elles sont calculées. Lorsqu'un programme effectue un calcul complexe, il est intéressant de garder le résultat de ce calcul, et de l'enregistrer dans un cache. Par exemple, une simulation physique complexe pour des effets spéciaux sera généralement enregistrée dans un cache sur le disque pour éviter de le recalculer plusieurs fois. De même, un programme de compositing peut mettre les images d'une séquence en cache, pour pouvoir les lire de manière fluide, plutôt que de les recalculer à chaque lecture.

Ce procédé présente plusieurs intérêts. D'abord, il permet d'accélerer les processus, puisque les calculs ne sont faits qu'une fois. Ensuite, dans le cas où un calcul n'est pas déterministe, comme c'est le cas de certaines simulations physiques, on peut obtenir à chaque fois le même résultat en le mettant en cache. Enfin, un cache permet d'améliorer l'interopérabilité*. En effet, deux programmes qui n'utilisent pas les mêmes algorithmes pour évaluer un calcul peuvent être rendus compatibles s'ils se contentent d'échanger le résultat de l'évaluation. Par exemple, une animation de personnage réalisée dans un logiciel donné peut être exportée en cache et se retrouver à l'identique dans un autre programme utilisant un système de rig complètement différent. 71, 103, 197

CAO La conception assistée par ordinateur (CAO) est la modélisation d'objets et de leurs propriétés physiques, en vue de leur fabrication. Elle permet d'obtenir des plans de fabrication et de montage, et parfois directement de commander les machines de fabrications, telles que les fraiseuses à commande numérique et les im-

primantes 3D, pour transformer des modèles numériques en objets physiques. 29, 212, 225, 234

CNC Le Centre national du cinéma et de l'image animée est une organisation publique française chargée en particulier d'aider à la production et à la diffusion des films, mais aussi d'autres types d'œuvres (séries, jeux vidéo). 107, 191

code source Le code source d'un programme est un ensemble de fichiers informatiques écrits dans un langage de programmation. Il permet de générer le programme, et décrit son comportement, la manière dont il traite les données et fait ses calculs. Il est nécessaire d'avoir accès au code source pour pouvoir modifier un programme, en créer une version différente, ou l'intégrer à un autre programme. 42, 44, 49, 50, 62, 66, 72, 75, 79, 85, 89, 94, 108, 109, 114, 126, 183, 192

fork Un fork est une version alternative d'un logiciel open source. Un développeur crée un fork d'un programme en le modifiant et en le redistribuant. Cela peut faire partie du processus normal de développement si les changements sont ensuite réintégrés dans la version amont*. Mais dès lors qu'un fork s'écarte assez de sa source pour être incompatible avec elle, les deux versions peuvent entrer en concurrence, et il devient difficile de réintégrer les modifications de l'une dans l'autre. 112, 113, 127, 244

format de fichier Un format de fichier est une manière de représenter des données. Un format spécifie selon quelle logique, quel ordre, quelle structure les données sont représentées. Il existe des milliers de formats de fichiers pour représenter tout type de données. La spécification d'un format comprend au minimum sa structure de données, et éventuellement des algorithmes (de compression et décompression, par exemple), et des implémentations. 113

framework Un framework est un ensemble de code informatique facilitant le développement logiciel dans un domaine donné, en fournissant un niveau d'abstraction plus élevé. Il se charge des détails de bas niveau, et offre au développeur une interface simplifiée ou unifiée. 39, 108, 193, 231, 270, 276

GNU/Linux GNU/Linux est le système d'exploitation libre le plus utilisé. Son nom est un sujet délicat. En effet, un système d'exploitation est composé de nombreux sous-systèmes, et « Linux » n'est à proprement parler que le noyau, un composant essentiel mais inutile isolément. Richard Stallman et la Free Software foundation préfèrent parler de GNU/Linux, car GNU était à l'origine le projet de développement d'un OS libre complet, auquel le noyau Linux de Linus Torvalds a été intégré par la suite. Je parle néanmoins dans cette thèse du système Linux, par raccourci. 46, 107, 164, 185, 240

IK La cinématique inverse (*inverse kinematics*, IK) est une technique de rigging et d'animation permettant de faciliter certains mouvements. Plutôt que d'animer chaque articulation, comme c'est le cas avec la cinématique directe (*forward kinematics*, FK), l'IK permet de bouger seulement l'extrémité du membre. Par exemple, pour animer une main, je n'aurai pas besoin de faire tourner l'épaule, le coude et le poignet ; je déplacerai simplement la main, et la position des autres articulations sera déduite automatiquement. 160

interopérabilité L'interopérabilité désigne la capacité qu'ont les programmes et les processus de travail d'être compatibles entre eux. C'est un problème récurrent en informatique, qui existe quand plusieurs solutions existent pour traiter un problème. Par exemple, si j'utilise un programme donné pour éditer un document écrit, je veux pouvoir l'ouvrir et l'éditer sur différentes plate-formes ou systèmes d'exploitation, dans différents programmes. Une solution à ce problème est de concevoir des formats de fichiers normalisés, qui pourront servir à faire passer les données d'un environnement à l'autre. On parle alors de formats d'échange. Mais si le programme que j'utilise ne permet pas de lire et écrire dans un format standard, ou si le standard n'est pas implémenté rigoureusement, c'est impossible. Dans l'exemple des documents textes, le format Open Document permet en théorie de passer d'un logiciel de traitement de texte à l'autre, si tant est qu'ils implémentent correctement le format. 77, 115, 246

interopérable 230, *voir* interopérabilité

JSON JSON est un format de fichier ouvert et standard, conçu pour échanger des données entre processus et entre applications. C'est un format simple, portable car utilisable dans la plupart des langages de programmation, et lisible par les humains. Il est très utilisé par exemple dans le web pour communiquer des données entre le serveur et le client (navigateur), pour mettre à jour une page de manière dynamique. 146, 197

Linux 99, 119, *voir* GNU/Linux

logiciel de création 3D Un logiciel de création 3D, ou suite de création 3D, souvent appelé DCC (*Digital Content Creation*, création de contenu numérique) est un programme permettant de réaliser les nombreuses opérations de l'animation et des effets spéciaux numériques 3D (cf. section I.1.2). Parmi les plus utilisés en 2019, on trouve Maya et 3DS Max d'Autodesk, Houdini de SideFX, Modo de Foundry, Cinema 4D de Maxon et le logiciel libre Blender. 34, 98, 99, 111, 115, 118, 119, 150, 152, 156, 164, 176, 193

model sheet Une model sheet ou planche de modélisation est un document graphique montrant un élément (personnage, décor, accessoire) dessiné avec assez d'indications pour pouvoir être modélisé en 3D. Généralement, y figureront au moins une vue de face, une vue de profil et une vue de dos, l'une à côté de l'autre et correspondant exactement dans la hauteur. Les hauteurs les plus caractéristiques, comme le sommet du crâne ou le bas des mains, pourront être reliées par des lignes pour bien voir la correspondance entre les vues. 25, 28

multi-plans La caméra multi-plans est une version évoluée du banc titre, dans laquelle plusieurs couches de décors et de celluloids sont placés sur des vitres parallèles entre elles. En déplaçant ces couches de décor l'une par rapport à l'autre, on donne à l'image l'illusion de la profondeur grâce à la parallaxe entre ces couches. 151, 152

médium Un médium est le moyen par lequel une œuvre est transmise au public. Il comprend le support physique et les dispositifs de diffusion, mais aussi la technique de création de l'œuvre. 16, 24, 155, 267, 281

ODbL La licence ODbL (*Open Database License*) est un contrat de licence destiné à être attaché à des bases de données ouvertes. Elle repose sur une logique similaire à celle des licences copyleft (cf. section II.1.3.1), mais a été conçue pour les spécificités juridiques des bases de données. 53, 56

patch Un patch est, par analogie avec le mot anglais pour le pansement ou la rustine, un correctif pour un bug informatique. Concrètement, c'est un fichier décrivant les lignes à ajouter, supprimer ou modifier dans le code source pour obtenir une nouvelle version. Les développeurs l'utilisent pour échanger du code, sans devoir s'envoyer des millions de lignes à chaque modification. En ne stockant dans le fichier que le différentiel, ce qui a changé entre les versions, on a un système facile à créer, transmettre et appliquer. 72, 73, 79, 104, 125

pipeline Le pipeline d'un studio ou d'un projet est la manière dont en est structurée la chaîne de fabrication. Il comprend les définitions des départements et étapes logiques de fabrication ; les arborescences et nomenclatures de fichiers et d'éléments au sein de ces fichiers ; les échanges de données d'un département à l'autre, d'un programme à l'autre ou d'un studio à l'autre (voir section I.1.2). 12, 21, 38, 59, 69, 77, 81, 96, 103, 108, 109, 113, 118, 123, 128, 135, 140, 143, 150, 154, 155, 164, 178, 185, 189, 193, 194, 230, 236, 267, 283

plan Le plan est la plus petite unité de découpage d'un film, après l'image. Il est caractérisé par la continuité de l'image, et deux plans sont séparés par une transition ou une coupure. 24, 26, 27, 34, 35, 37, 38, 40, 56, 73, 136, 143, 148, 150, 151, 181, 185, 186, 191, 193, 197, 199, 201, 203, 207, 210, 281

plug-in 163, 183, *voir* add-on

R&D La recherche et développement (R&D) est une pratique industrielle consistant à mener des recherches fondamentales ou appliquées sur un sujet donné, et à développer des produits, services, procédés, algorithmes, programmes. Cette activité est indispensable dans de nombreuses industries, y compris dans l'animation. Une entreprise dotée d'un service de R&D effectue de la veille technologique pour

connaître les pratiques modernes dans l'industrie, et développe ses propres solutions pour répondre au mieux aux problématiques de production. 11, 20, 37, 41, 76, 97, 98, 102, 106, 120, 131, 236, 284

rapport de bug Un rapport de bug, de l'anglais *bug report*, parfois aussi appelé « rapport d'anomalie », est une procédure courante dans le développement logiciel, selon laquelle un utilisateur signale un bug, une anomalie ou un dysfonctionnement aux développeurs du programme qu'il utilise. Un rapport de bug peut être interne au groupe développant le programme, comme c'est le cas lors de tests d'assurance qualité, ou provenir d'utilisateurs réels. C'est un processus indispensable pour améliorer la stabilité des programmes. La correction des bugs demande d'importantes ressources de développement. 73, 75, 237

retake Une retake, mot emprunté au vocabulaire du cinéma et signifiant littéralement reprise, est une modification demandée par la hiérarchie. Elle a lieu lors d'une validation auprès du superviseur ou auprès du client, lorsque des changements sont demandées sur un élément, plan ou séquence. Elle fait partie intégrante de la création d'œuvres animées, et peut intervenir à tout moment de la fabrication. 39, 147, 185, 191, 281

scalabilité La scalabilité d'un système ou d'un processus est sa capacité à être exploité à plus grande échelle en ajoutant des ressources. De l'anglais *scale*. 211, 281

script Un script est un court programme permettant d'appeler des fonctions et de manipuler des données au sein d'une application, via une API*. Il permet d'automatiser des tâches, ou d'ajouter des fonctionnalités au programme de base. Il se distingue donc des bibliothèques, qui sont appelées par les applications, et des applications elles-mêmes, qui peuvent s'exécuter indépendamment. 12, 21, 22, 43, 67, 76, 88, 89, 103, 123, 141, 169, 174, 185, 198, 208, 244, 270, 283

skinning Le skinning est un principe permettant de lier une surface à un rig. La surface est considérée comme une peau (en anglais *skin*) bougeant avec le squelette. Cette liaison permet d'animer des objects complexes dont la surface se déforme, par exemple des personnages. 30, 164, 172

suite de création 3D 76, *voir* logiciel de création 3D

séquence Une séquence est un groupe de plans constituant une unité narrative, souvent une unité de temps et de lieu. C'est grossièrement l'équivalent d'un chapitre en littérature. 24, 27, 35, 38, 40, 73, 156, 179, 184, 203

TD Le TD ou *technical director* — distinct en France du directeur technique — est dans l'animation un poste spécialisé, dont la tâche est d'élaborer le pipeline au sens le plus large : il conçoit l'architecture et l'organisation du projet, choisit les solutions techniques adaptées et crée des outils pour faciliter le travail des graphistes et de la production. Il intervient à toutes les étapes pour fluidifier les échanges et la fabrication. 20–22, 37, 41, 69, 76, 106, 131, 141, 190, 193, 237, 284

timeline La timeline, mot anglais pour « ligne de temps », désigne en infographie une interface de visualisation du temps. Elle est généralement représentée par un large rectangle horizontal, dans lequel la gauche marque le début et la droite, la fin. Une ligne verticale peut symboliser l'instant sélectionné, et on peut aller avancer ou reculer en déplaçant cette ligne. 71, 145, 186

Table des figures

Bibliographie

Livres

[1] Broca, Sébastien, *Utopie du logiciel libre : du bricolage informatique à la réinvention sociale*, Le passager clandestin, 2013, 288 p., isbn : 978-2-916952-95-6, url : `http://lepassagerclandestin.fr/catalogue/essais/utopie-du-logiciel-libre.html`.

[2] Caron, Christophe, *Droit d'auteur et droits voisins*, 5e éd., Paris : LexisNeis, 2017, isbn : 978-2-7110-2845-0, url : `https://www.lgdj.fr/droit-d-auteur-et-droits-voisins-9782711028450.html` (visité le 05/02/2018).

[3] Élie, François, *Économie du logiciel libre*, Paris : Eyrolles, 2009, isbn : 978-2-212-12463-7.

[4] MacLean, Fraser, *Setting the Scene : The Art & Evolution of Animation Layout*, Chronicle Books, 16 nov. 2011, 270 p., isbn : 978-0-8118-6987-4, Google Books : `AoajuAAACAAJ`.

[5] Masson, Terrence, *CG 101 : A Computer Graphics Industry Reference*, Indianapolis, Ind : New Riders, 1999, 500 p., isbn : 978-0-7357-0046-8.

[6] Perline et Noisette, Thierry, *La bataille du logiciel libre : dix clés pour comprendre*, Paris : La Découverte, 2006, isbn : 978-2-7071-4880-3.

[7] Raymond, Eric S., *The Cathedral & the Bazaar : Musings on Linux and Open Source by an Accidental Revolutionary*, 1st ed, Beijing ; Cambridge, Mass : O'Reilly,

1999, 268 p., ISBN : 978-1-56592-724-7, URL : http://www.catb.org/~esr/writings/cathedral-bazaar/.

[8] STALLMAN, Richard M., WILLIAMS, Sam et MASUTTI, Christophe, *Richard Stallman et la révolution du logiciel libre : une biographie autorisée*, Eyrolles, 14 jan. 2010, 345 p., ISBN : 978-2-212-12609-9, Google Books : F1Enl7x8j0YC.

Articles

[9] BROCA, Sébastien et CORIAT, Benjamin, « Le logiciel libre et les communs », in : *Revue internationale de droit economique* t. XXIX.3 (27 nov. 2015), p. 265-284, ISSN : 1010-8831, URL : https://www.cairn.info/revue-internationale-de-droit-economique-2015-3-page-265.htm (visité le 27/08/2019).

[10] CHAPMAN, Owen B. et SAWCHUK, Kim, « Research-Creation : Intervention, Analysis and "Family Resemblances" », in : *Canadian Journal of Communication* 37.1 (13 avr. 2012), ISSN : 1499-6642, DOI : 10.22230/cjc.2012v37n1a2489, URL : https://www.cjc-online.ca/index.php/journal/article/view/2489 (visité le 25/01/2019).

[11] CIANCARINI, Paolo, RUSSO, Daniel, SILLITTI, Alberto et SUCCI, Giancarlo, « Reverse Engineering : A European IPR Perspective », in : *Proceedings of the 31st Annual ACM Symposium on Applied Computing - SAC '16*, The 31st Annual ACM Symposium, Pisa, Italy : ACM Press, 2016, p. 1498-1503, ISBN : 978-1-4503-3739-7, DOI : 10.1145/2851613.2851790, URL : http://dl.acm.org/citation.cfm?doid=2851613.2851790 (visité le 06/08/2019).

[12] GEORGE-MOLLAND, Anne-Laure, PLESSIET, Cédric, PÊCHEUX, Étienne et VIDAL, Dominique, « Le métier de technical director, hybridation entre expertise technique et création artistique au sein des pipelines d'effets spéciaux numériques », in : , La Création Collective au Cinéma, 24 nov. 2017, URL : https://creationcollectiveaucinema.com/revue-n02-2019/.

[13] Jullien, Nicolas et Zimmermann, Jean-Benoît, « Le logiciel libre : une nouvelle approche de la propriété intellectuelle », in : *Revue d'économie industrielle* 99.1 (2002), p. 159-178, DOI : 10.3406/rei.2002.3021, URL : https://www.persee.fr/doc/rei_0154-3229_2002_num_99_1_3021 (visité le 29/01/2019).

[14] Kainz, Florian, Bogart, Rod, Stanczyk, Piotr et Hillman, Peter, « Technical Introduction to OpenEXR », in : (), p. 21.

[15] Luxen, Dennis et Vetter, Christian, « Real-Time Routing with OpenStreetMap Data », in : , ACM Press, 2011, p. 513, ISBN : 978-1-4503-1031-4, DOI : 10.1145/2093973.2094062, URL : http://dl.acm.org/citation.cfm?doid=2093973.2094062 (visité le 27/05/2018).

[16] Rupnik, Ewelina, Daakir, Mehdi et Pierrot Deseilligny, Marc, « MicMac – a Free, Open-Source Solution for Photogrammetry », in : *Open Geospatial Data, Software and Standards* 2.1 (5 juin 2017), p. 14, ISSN : 2363-7501, DOI : 10.1186/s40965-017-0027-2, URL : https://doi.org/10.1186/s40965-017-0027-2 (visité le 05/09/2019).

[17] Tion, Guillaume, « Effets spéciaux : Pierre Buffin, la grande illusion », in : *Libération* (21 avr. 2017), URL : https://next.liberation.fr/images/2017/04/21/effets-speciaux-pierre-buffin-la-grande-illusion_1564388 (visité le 21/06/2019).

Thèses et mémoires

[18] George-Molland, Anne-Laure, « La collaboration au cœur du processus de création des œuvres audiovisuelles numériques : analyse des transformations apportées par le développement des technologies et par l'évolution des savoir-faire. », Paris 8, 12 déc. 2007, 368 p.

[19] Lelièvre, Pierre, *Définition du rôle de TD, Technical Director, au sein des studios de fabrication d'images numériques*, Saint-Denis : ENS Louis Lumière, juin 2012, p. 191, URL : https://www.ens-louis-lumiere.fr/index.php/definition-du-role-de-td-technical-director-au-sein-des-studios-de-fabrication-dimages-numeriques.

[20] Perez, Flavio, *Le pipeline de l'image de synthèse : définitions et enjeux pour les œuvres collaboratives*, Mémoire de master, Saint-Denis : Université Paris 8, 2017.

Conférences

[21] Jon Manning, « Making Night in the Woods Better with Open Source » (Game Developers Conference, San Francisco, Californie, USA), 2017, URL : https://youtu.be/Qsiu-zzDYww (visité le 29/04/2019).

[22] Muller, Mathieu et Le Levier, Bruno, « Actualité du temps réel », Rencontres Animation Développement Innovation (Pôle Image Magelis, Angoulême, France), 18 nov. 2016, URL : https://www.rencontres-animation-formation.org/synthese_2016.

[23] Roosendaal, Ton, « Blender Foundation – Community Meeting », Siggraph (Anaheim, Californie, États-Unis d'Amérique), 2013, URL : https://www.blender.org/bf/sig2013.pdf.

[24] Seguin, Maxime, « La 3D au service de la connaissance », Rencontres d'archéologie de la Narbonnaise (Salle des Synodes, Narbonne), 29 sept. 2017, URL : http://www.prehistoire.org/offres/gestion/actus_all_515_31942-1/la-3d-au-service-de-la-connaissance.html.

Pages web

[25] Adobe Systems Incorporated, *Changes to Creative Cloud Download Availability*, 2018, URL : https://theblog.adobe.com/changes-to-creative-cloud-download-availability (visité le 11/07/2019).

[26] Adobe Systems Incorporated, *Conditions d'utilisation pour les produits et services Adobe*, avr. 2020, URL : https://www.adobe.com/fr/legal/terms.html (visité le 14/07/2020).

[27] Autodesk, *Conditions Générales d'Utilisation*, 18 mai 2018, URL : https://www.autodesk.com/company/terms-of-use/fr/general-terms#software) (visité le 01/09/2019).

[28] Blender Foundation, *FAQ*, URL : https://www.blender.org/support/faq/ (visité le 12/03/2019).

[29] Christophe Archambault, *Asset Managers*, 13 nov. 2018, URL : http://lepipeline.org/asset-managers (visité le 23/09/2019).

[30] Dombrova, Chad, *Why PyMEL ?*, 2009, URL : https://download.autodesk.com/us/maya/2011help/PyMel/why_pymel.html (visité le 15/03/2019).

[31] Electronic Frontier Foundation, *Coders' Rights Project Reverse Engineering FAQ*, 6 août 2008, URL : https://www.eff.org/issues/coders/reverse-engineering-faq (visité le 06/08/2019).

[32] Epic Games, Inc., *EULA*, URL : https://www.unrealengine.com/en-US/eula (visité le 18/07/2019).

[33] Free Software Foundation, Inc., *What Is Free Software and Why Is It so Important for Society ?*, URL : https://www.fsf.org/about/what-is-free-software (visité le 28/01/2019).

[34] National Snow and Ice Data Center, *Use and Copyright*, URL : https://nsidc.org/about/use_copyright.html (visité le 21/05/2019).

[35] Pope, Lucas, *Return of the Obra Dinn*, 14 août 2014, URL : https://forums.tigsource.com/index.php?topic=40832.msg1051550#msg1051550 (visité le 17/07/2019).

[36] Rousseau, Frank, *TNZPV Studio Sponsors January Features for Kitsu*, 7 jan. 2019, URL : https://medium.com/cgwire/tnzpv-studio-sponsors-january-features-for-kitsu-301405ed23e1 (visité le 21/07/2019).

[37] Stallman, Richard, *Why Open Source Misses the Point of Free Software*, 2007-2019, URL : https://www.gnu.org/philosophy/open-source-misses-the-point.html (visité le 09/08/2019).

[38] *Tone.Js*, URL : https://tonejs.github.io/ (visité le 27/05/2018).

[39] Visual Effects Society Technology Committee, *VFX Reference Platform*, 2014-2019, URL : https://vfxplatform.com/ (visité le 27/09/2019).

Données

[40] Rignot, Eric, Mouginot, Jeremie et Scheuchl, Bernd, *MEaSUREs InSAR-Based Antarctica Ice Velocity Map, Version 2*, 2017, DOI : 10.5067/d7gk8f5j8m8r, URL : https://nsidc.org/data/nsidc-0484/versions/2 (visité le 21/05/2019).

Films et vidéos

[41] Kostis, Helen-Nicole, *Pulse of Snow and Sea Ice*, avec la coll. de Starr, Cindy et Markus, Thorsten, 2012, URL : https://svs.gsfc.nasa.gov/3944 (visité le 05/09/2019).

[42] Veselin Efremov, *Adam*, 2016, URL : https://unity3d.com/pages/adam (visité le 01/09/2019).

Logiciels, scripts, plug-ins

[43] Coureau, Damien, *Kabaret Studio*, 2012, URL : https://www.kabaretstudio.com (visité le 20/09/2019).

[44] Esau, Andreas, *Cutout Animation Tools*, version 1.0.4, 2015-2018, URL : https://github.com/ndee85/coa_tools (visité le 17/04/2019).

[45] Forman, Charles, *Storyboarder*, version 1.13.0, Wonder Unit Inc., 2016, URL : https://wonderunit.com/storyboarder/ (visité le 30/08/2019).

[46] Harvey, Geoff, *Hoafaloaf/Seqparse*, 2017, URL : https://github.com/hoafaloaf/seqparse (visité le 26/03/2020).

[47] Holowka, Alexander, *YarnSpinnerTool/YarnSpinner*, Secret Lab, 2015, URL : https://github.com/YarnSpinnerTool/YarnSpinner (visité le 22/10/2019).

[48] Israel, Justin, *Justinfx/Fileseq*, 2012, URL : https://github.com/justinfx/fileseq (visité le 26/03/2020).

[49] MapBox, *TileMill*, 2010-2019, URL : https://tilemill-project.github.io/tilemill/ (visité le 27/05/2018).

[50] Olive Team, *Olive-Editor/Olive*, Olive Team, 2018, URL : https://github.com/olive-editor/olive (visité le 18/03/2020).

[51] *Openexr/Openexr*, OpenEXR, 2003, URL : https://github.com/openexr/openexr (visité le 25/10/2019).

[52] Perez, Flavio, *Storymatic*, Les Fées Spéciales, 2016, URL : https://github.com/LesFeesSpeciales/storymatic (visité le 18/10/2019).

[53] Rousseau, Frank, *CGWire*, 2017, URL : https://www.cg-wire.com/ (visité le 20/09/2019).

[54] *Substance*, Adobe, URL : https://www.substance3d.com/ (visité le 09/03/2020).

[55] Van Nieuwenhuizen, Jasper, *2d Animation Tools*, 2014-2017, URL : https://github.com/jasperges/2d_animation_tools (visité le 04/03/2019).

[56] Whitaker, Jeffrey, *Pyproj4/Pyproj*, pyproj4, 2013, URL : https://github.com/pyproj4/pyproj (visité le 05/09/2019).

[57] Yamaguchi, Kota, *Psd-Tools*, psd-tools, 2012, URL : https://github.com/psd-tools/psd-tools (visité le 20/02/2020).

[58] Yates, Charles et Dennedy, Dan, *MLT*, MLT Framework, 2003, URL : https://github.com/mltframework/mlt (visité le 21/02/2020).

Divers

[59] Adobe Systems Incorporated et Knoll, Thomas, *Adobe Photoshop File Formats Specification*, 1990-2016, URL : https://www.adobe.com/devnet-apps/photoshop/fileformatashtml/ (visité le 04/03/2019).

[60] Alexandre Prokoudine, *The Demise of Natron : How We Got Here and Where We Go Further*, avec la coll. d'Alexandre Gauthier-Foichat et Frédéric Devernay, 25 nov. 2018, URL : http://libregraphicsworld.org/blog/entry/the-demise-of-natron (visité le 23/08/2019).

[61] April, Association, *Livre Blanc Sur Les Modèles Économiques Du Logiciel Libre*, déc. 2007, URL : http://www.april.org/livre-blanc-sur-les-modeles-economiques-du-logiciel-libre (visité le 03/10/2017).

[62] Centre National de la Cinématographie, *Soutien financier aux industries techniques du cinéma, de l'audiovisuelet des autres arts et industries de l'image animée — Présentation générale*, 2019, URL : https://www.cnc.fr/documents/36995/159604/pr%C3%A9sentation+du+Soutien+financier+aux+industries+techniques.pdf/1b3f18fa-a53f-5cdc-866b-230fbed6ebfd.

[63] Creative Commons, *Creative Commons — CC0 1.0 Universel*, 2009, URL : https://creativecommons.org/publicdomain/zero/1.0/deed.fr (visité le 05/09/2019).

[64] Free Software Foundation, Inc., *GNU General Public License, Version 2*, 1991, URL : https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html (visité le 19/02/2019).

[65] Hughes, Kerrie, *Autodesk Answers Your Questions on the Demise of Softimage*, avec la coll. de Patel, Maurice, 20 avr. 2014, URL : https://www.creativebloq.com/3d/autodesk-answers-your-questions-demise-softimage-31411069 (visité le 26/08/2019).

[66] John Harvey, *Blank Map World Equirectangular*, avec la coll. de Canuckguy et Central Intelligence Agency, 2008, URL : https://en.wikipedia.org/wiki/File:BlankMap-World6-Equirectangular.svg (visité le 05/09/2019).

[67] Kyu Lab, *Étude sur l'identification des besoins en competences et en formation des studios*, juin 2018, URL : http://www.cpnef-av.fr/les-etudes/films-d-animation-quels-besoins-en-competences- (visité le 22/07/2019).

[68] *LOI n° 2014-856 du 31 juillet 2014 relative à l'économie sociale et solidaire*, 31 juil. 2014, URL : https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000029313296 (visité le 26/07/2019).

[69] MPEG LA, *MPEG LA's AVC License Will Not Charge Royalties for Internet Video That Is Free to End Users through Life of License*, 26 août 2010, URL : http://www.mpegla.com/Lists/MPEG%20LA%20News%20List/Attachments/74/n-10-08-26.pdf.

[70] Sam Hocevar, *What The Fuck Public License*, URL : http://www.wtfpl.net/about/ (visité le 01/09/2019).

Annexe A

Exemple de fichier JSON pour les exports de *Dilili à Paris*


```
{
    "files": [
        {
            "use_textures": true,
            "group": "PANTIN",
            "fbx": "PANTIN.fbx",
            "visibility": {}
        },
        {
            "use_textures": false,
            "group": "RLO",
            "fbx": "RLO.fbx",
            "cache": "RLO.pc2"
        },
        {
            "use_textures": false,
            "group": "DECOR_REF",
            "fbx": "DECOR_REF.fbx",
            "visibility": {}
        },
```

```
        {
            "use_textures": true,
            "group": "DECOR_RND",
            "fbx": "DECOR_RND.fbx",
            "visibility": {}
        },
        {
            "use_textures": true,
            "group": "CAMERA",
            "fbx": "camera.fbx",
            ".focalLength": {}
        }
    ],
    "sounds": [
        {
            "start_frame": 1,
            "file": "DIL_S12_P0041_son-maquette_v1.wav"
        }
    ],
    "settings": {
        "frame_start": 101,
        "resolution_x": 2048,
        "frame_end": 172,
        "resolution_y": 1080
    }
}
```

Annexe B

Autonomisation des artistes : *Music Road*

Music Road est un projet personnel conçu pour explorer des questions liées au logiciel libre mais, s'éloignant du cinéma d'animation, faire une incursion dans le monde du web. Plus précisément, je cherchais à déterminer dans quelle mesure une œuvre créée par un seul artiste pouvait s'appuyer sur du code et des données open source. Il n'a pas été réalisé dans le cadre de l'entreprise, et ne répond donc pas du tout aux mêmes contraintes industrielles de production. Il n'explore pas les relations entre studios et développeurs de logiciels libres, ni l'élaboration d'un pipeline* complexe, mais plutôt la carte en tant que médium* artistique, un mode de représentation qui m'intéresse par le rapprochement qu'il fait entre l'art et la science.

Principe du projet

Le projet *MusicRoad* (figure B.1) consiste en une application dont le fonctionnement est le suivant : une carte permet de calculer un itinéraire, à la façon des services comme Mappy et Google Maps, en plaçant des points (marqueurs) sur la carte. En plus de calculer et d'afficher l'itinéraire, le programme propose également une mélodie, calculée à partir des mêmes données. On obtient ainsi, pour chaque ensemble de points de départ, d'arrivée, et d'escales, une mélodie unique associée à un trajet.

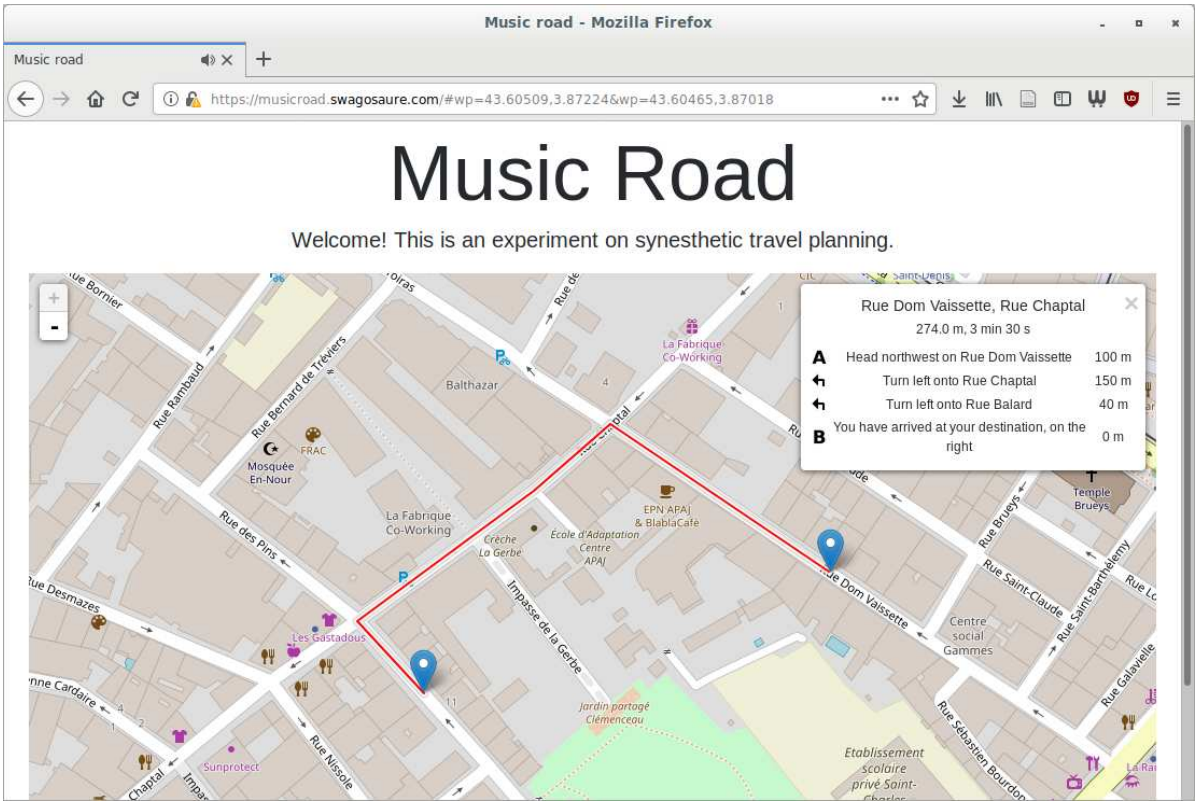


FIGURE B.1 – Accueil du site web, interface de l'application.

Pour générer une mélodie, l'idée initiale était de transposer l'itinéraire de la manière suivante, choisie en partie arbitrairement, mais selon une logique déterministe [1] fondée sur la musique et le sens de lecture occidentaux :

— Chaque instruction (changement de direction) correspond à un changement de note.

— Les virages à gauche signifient une note plus grave.

— Les virages à droite signifient une note plus aiguë.

— Les virages serrés signifient une note plus éloignée de la précédente que les plus faibles inflexions de direction.

— La durée d'une note est fonction de la distance parcourue entre deux instructions.

Cette version de l'algorithme de génération musicale a été complexifiée au cours de l'implémentation, afin d'apporter de la variété, mais en respectant le principe fondamen-

1. En d'autres termes, le hasard n'intervient pas dans la génération. Un seul trajet et une seule mélodie sont calculées d'après les données en entrée.

tal qu'un itinéraire donné doit correspondre à une et une seule mélodie. L'inverse n'est pas vrai : une même mélodie peut être jouée sur plusieurs trajets différents.

Une première itération de ce programme, conçue avant le début de ma thèse, faisait appel à l'API de Google Maps. En envoyant une requête comprenant les coordonnées (latitude, longitude) des différents points constituant l'itinéraire, on obtenait un document contenant les instructions du trajet. Il était facile de faire une analyse syntaxique de ce document (*parsing*) pour générer la mélodie. Des méthodes étaient également fournies pour intégrer une carte Google Maps dans une page web, ainsi que l'itinéraire. Cependant, les indications retournées par la requête étaient insuffisamment détaillées à l'époque (en 2014, je n'ai pas réessayé depuis) pour permettre de calculer une mélodie.

Quelques années plus tard, au cours de ma thèse et de mon activité professionnelle, j'ai été amené à élaborer des cartes animées pour des films et des musées (cf. sections III.1.2 et III.1.3). J'ai donc pris connaissance de certains éléments de l'écosystème d'outils et de bases de données cartographiques, en particulier libres.

La démarche de ce travail est de se servir le plus possible de ressources open source. Un autre critère était initialement de ne s'appuyer sur aucun service en ligne (logiciel en tant que service, *Software As A Service*) [2], pour trois raisons : garantir la disponibilité du programme sans dépendre de celle des services utilisés et, dans l'hypothèse de données confidentielles, l'accès aux données et le contrôle de ces données ; pouvoir offrir le service hors ligne, dans le cadre d'une installation artistique in situ par exemple ; avoir une plus grande liberté dans le rendu des éléments graphiques, en pouvant modifier le code, les images et les données.

L'application finale est disponible sur Internet, sur mon site personnel [3].

2. Les *Software As A Service* sont un type d'infrastructure *cloud* où un logiciel est exécuté sur un serveur distant, et non pas sur la machine de l'utilisateur. C'est le cas de tous les sites proposant des fonctionnalités requérant du calcul ou des données en ligne. C'est le cas également pour ce projet, lorsqu'il est utilisé depuis le site web, et non pas chez l'utilisateur, comme c'était l'intention de départ.

3. Le site est disponible à l'adresse suivante : `https://swagosaure.com/music_road/`

Processus créatif

La conception du projet *Music Road* a consisté à utiliser des bibliothèques*, des composants logiciels publiés sous licences libres, ainsi que des données ouvertes, afin de créer une œuvre originale. Il a été l'occasion d'une recherche sur ce qui caractérise l'utilisation de bibliothèques libres pour la création. Cette section développera le processus de développement de l'application, et examinera les intérêts pour l'artiste, ainsi que les difficultés qui existent à appréhender un vaste écosystème de composants logiciels. Cette application développée pour le web se détache quelque peu des outils utilisés ordinairement dans l'animation, mais le processus de développement est similaire, dans la mesure où des composants logiciels existants peuvent être utilisés pour développer des outils pour l'animation (voir l'exemple de pyproj dans la section III.6.1). Les étapes de conception (cahier des charges, veille, prototypage, implémentation), sont aussi sensiblement les mêmes que celles qui sont décrites ici. L'approche est aussi différente et complémentaire, car si je connais assez bien le développement d'add-ons* et de scripts* pour l'animation 3D, le web était à l'époque un domaine que j'abordais tout juste. Les enseignements que je tire de cette expérience d'apprentissage peuvent être transposés au développement pour l'animation.

B.1 Cahier des charges

Une fois définies l'idée globale et les mécanismes de l'application (cf. annexe B), la première étape de ce projet a été de déterminer de quels composants logiciels j'aurais besoin pour la construire. En voici une liste issue des premières réflexions.

- jeu de données cartographiques,
    - bibliothèque d'affichage de ces données dans le navigateur, générateur de tuiles ;
- programme ou bibliothèque de calcul d'itinéraire,
    - bibliothèque client d'affichage d'itinéraire compatible ;
- framework* JavaScript, environnement de développement ;

— bibliothèque de synthèse sonore (serveur ou client) ;

La première phase de recherche m'a conduit à l'idée fausse qu'il serait facile de trouver des implémentations libres de ces composants, puisqu'il en existait un grand nombre sous licence libre répondant à ces besoins. Pour mener à bien ce projet, j'ai évalué les composants suivants : Node.js, OpenStreetMap, Leaflet, Mapnik, Tilemill, OSRM, Leaflet Routing Machine, Flocking, Tone.js, timbre.js. Chacun de ces composants est plus ou moins bien documenté, et facile à installer, paramétrer, utiliser, modifier et étendre. J'en détaillerai certains plus loin, mais cette énumération suffit à se rendre compte que l'abondance d'applications dans l'écosystème de développement web constitue en soi un obstacle à la conception d'une application par un artiste développeur. C'est bien sûr vrai de tout travail de création, mais ici la complexité est d'autant plus accablante qu'elle part en tous sens, et que chaque nouveau problème a de nombreuses solutions existantes possibles à explorer et valider [4].

Le composant essentiel, qui permet à ce projet d'exister, est la base de données OpenStreetMap (cf. section I.2.4.2). L'intérêt de cette base de données pour une application comme celle-ci est que les données peuvent être téléchargées, utilisées comme matériau de base de création, mélangées avec des données originales, et remixées. Le téléchargement de la base de données complète est simple, mais une contrainte à considérer est son poids, d'environ 800 Go [5] en 2017. Ne disposant pas de quoi stocker et traiter cette quantité de données, j'ai choisi de me concentrer sur un extrait de l'ancienne région où je réside, le Languedoc-Roussillon, qui est facilement téléchargeable [6].

4. Cette problématique est directement applicable au développement d'outils pour l'animation. Par exemple, l'équipe technique des Fées spéciales a eu besoin à un moment de traiter des séquence d'images, et donc d'analyser les images présentes dans un dossier, trouver le nombre de chiffres de la séquence, et repérer les images manquantes. C'est un problème récurrent dans l'animation. Nous avons cherché des bibliothèques Python permettant de simplifier ce problème, et bien qu'elles soient assez mal référencées, nous en avons trouvé plusieurs [46, 48], aux fonctionnalités similaires mais aux interfaces différentes. Suite à une phase de test, le choix entre ces bibliothèques a été plus ou moins arbitraire.

5. Ce chiffre est donné pour un extrait mondial de la base de données, c'est-à-dire toutes les données à un instant donné. Le poids de la base entière, comprenant tout l'historique du projet, n'est pas directement disponible, et je ne peux pas la récupérer pour la peser, mais il est certain que c'est plus du double. La page `https://wiki.openstreetmap.org/w/index.php?title=Planet.osm` donne régulièrement le poids à jour de la base.

6. La page `https://wiki.openstreetmap.org/wiki/Planet.osm#Country_and_area_extracts` explique où télécharger des extraits de zones internationales, nationales, ou régionales.

B.2 Rendu graphique de la carte

Les sites proposant des cartes en ligne reposent sur de lourdes bases de données géographiques, comprenant des données dans l'espace (points représentant des coordonnées terrestres, lignes reliant ces points, polygones délimitant des zones), et des métadonnées associées à ces primitives géométriques, définissant les caractéristiques des objets décrits. Par exemple, les routes sont constituées de lignes, auxquelles on associe des informations telles que le type (de l'impasse piétonne à l'autoroute), le nom, les sens uniques, les limitations de vitesse, etc. De la même manière, on peut décrire géographiquement les infrastructures, les bâtiments, les étendues d'eau, les paysages, etc. Ces bases de données SIG (Système d'Information Géographique) forment la partie des données qui est invisible pour l'utilisateur. Car avant d'être servies sur la page web, ces données sont rendues en images appelées tuiles (*tiles*). Ces images sont ainsi nommées car la surface de la carte est découpée en petites parties, qui une fois juxtaposées, permettent d'afficher la carte. L'avantage est que l'utilisateur n'a pas besoin de charger l'intégralité de la carte, qui pèserait bien trop lourd pour être transférée et affichée en une fois.

Mon intention initiale était d'utiliser les données téléchargées (la région Languedoc-Roussillon) dans deux buts différents : rendre les tuiles de la carte, et calculer l'itinéraire. Pour le premier problème des tuiles, j'ai commencé par installer les composants logiciels recommandés sur le wiki d'OpenStreetMap [7] pour un serveur complet, à savoir un serveur web (Apache), un moteur de base de données (PostgreSQL), dans lequel la base téléchargée précédemment est importée. Une extension du serveur web Apache (mod_tile) permet la gestion des tuiles : lors d'une requête de tuile, un cache est consulté pour savoir si la tuile existe et est suffisamment récente. Dans le cas contraire, elle est rendue, stockée, et servie à l'utilisateur. Le rendu se fait d'après un système complexe de feuilles de style. Je ne décrirai pas ce système en détail ici, car la création de ces feuilles de style est un sujet à part entière. En effet, de nombreux systèmes existent, pas nécessairement compatibles. Cependant, je suis parvenu à installer et utiliser un autre programme de

7. Des explications sont disponibles ici : `https://wiki.openstreetmap.org/wiki/Deploying_your_own_Slippy_Map`

création de styles de carte, Tilemill [49], qui permet de faire des sélections dans un langage proche du CSS [8].

Si passionnants que soient les métiers de designer et de cartographe, j'ai finalement renoncé à la création d'un design nouveau pour ce site, en raison du temps limité dont je dispose, et de la difficulté de maîtriser les outils, tant pour le façonnage que pour le rendu. Sachant, d'une part, qu'après plusieurs jours infructueux, le serveur de rendu de tuiles ne marchait pas, et que le style auquel j'étais parvenu était au mieux un rudimentaire essai d'amélioration par rapport aux styles d'exemple, et d'autre part que les tuiles d'OpenStreetMap.org peuvent être utilisées par d'autres sites, j'ai décidé de faire cette concession, et de faire pointer mon serveur web directement sur celui des tuiles d'OSM. Cela constitue une entorse aux contraintes initiales, puisque cela implique que des données venant d'un serveur externe seront utilisées par le programme. En cas d'indisponibilité du serveur, mon site le sera également. La même chose se produit si un changement de format ou d'API rend incompatibles les solutions mises en œuvre. On peut également considérer le cas où OpenStreetMap déciderait de changer le style ou le rendu de ses cartes : MusicRoad changerait en conséquence sans que j'en aie fait le choix, peut-être même sans que je le sache. Ce ne serait pas dramatique et n'affecterait pas nécessairement l'expérience de l'utilisateur, mais dans la mesure où je n'aurais pas de contrôle dessus, pourrait-on encore dire que j'en serais l'auteur, même en tenant compte du fait que mon œuvre est dérivée d'une autre ? Cela étant, cela illustre bien la difficulté à l'heure actuelle de faire du développement de services web en se passant des SaaS (*Software as a Service*), malgré l'existence du logiciel libre. C'est aussi une contrainte pour l'artiste, qui perd en contrôle ce qu'il gagne en facilité de mise en œuvre.

B.3 Itinéraire

8. Langage de mise en forme pour le web. Il permet de définir les caractéristiques de forme de chaque élément : textes, tableaux, images, divisions de la page, etc., indépendamment du contenu. Pour les cartes, les éléments sont les points, lignes, surfaces, noms et attributs, et ils peuvent être coloriés ou hachurés, représentés par des idéogrammes, les textes placés et mis en forme, etc.

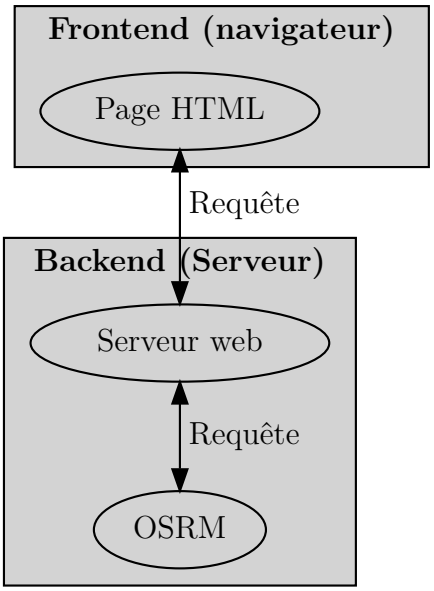


FIGURE B.2 – La partie backend est complète, mais la carte ne s'affiche pas encore.

L'étape suivante fut la recherche des programmes et bibliothèques permettant le calcul de l'itinéraire et l'affichage de la page web. En ce qui concerne le premier point, plusieurs programmes capables de traiter les données OpenStreetMap pour générer un itinéraire existent. J'ai choisi OSRM (Open Source Routing Machine) [15]. La raison initiale en était la page de démonstration très convaincante, et le fait que ce programme soit utilisé sur le site officiel d'OpenStreetMap, ce qui laissait à présager de sa robustesse. De plus, un exemple de *frontend* (page web connectée au serveur de calcul d'itinéraire) était donné dans les dépôts. Notons que ce choix est encore une fois assez arbitraire, car le wiki d'OSM liste une dizaine de programmes écrits dans six langages différents. Comment choisir, lorsque le temps et les connaissances manquent, et qu'aucune autorité ne peut donner de conseils ? La veille est bien sûr une part essentielle du travail de recherche, mais dans un cas comme celui-ci, où l'investissement est lourd par rapport à la finalité (à savoir, mélanger des technologies existantes pour créer un programme finalement simple), il faut parfois se résoudre à s'en tenir à la première impression, ou à faire confiance à ce que disent des inconnus sur des forums ou des wikis. Le serveur ne fut pas difficile à compiler en suivant les instructions. OSRM traite ensuite les données géographiques extraites de la base de données OSM, pour créer un fichier dédié spécifiquement au calcul d'itinéraire.

J'ai dit qu'OSRM était un serveur. Cela signifie que pour obtenir les informations d'itinéraire, il faut faire une requête HTTP sur un port spécifique, dans un format donné. C'est ce qu'il se passe lorsqu'on consulte n'importe quel site web. Il faut par ailleurs un *autre* serveur HTTP, le serveur web communiquant avec l'extérieur, qui sera chargé d'envoyer les pages web lorsque l'utilisateur se connectera via son navigateur. Il faut

donc que le serveur web puisse recevoir une requête d'itinéraire depuis la page web, la transmette localement au serveur OSRM, puis transmette la réponse dans l'autre sens. On le voit dans la figure B.2, où le serveur Apache envoie la page au *frontend*, chez le client.

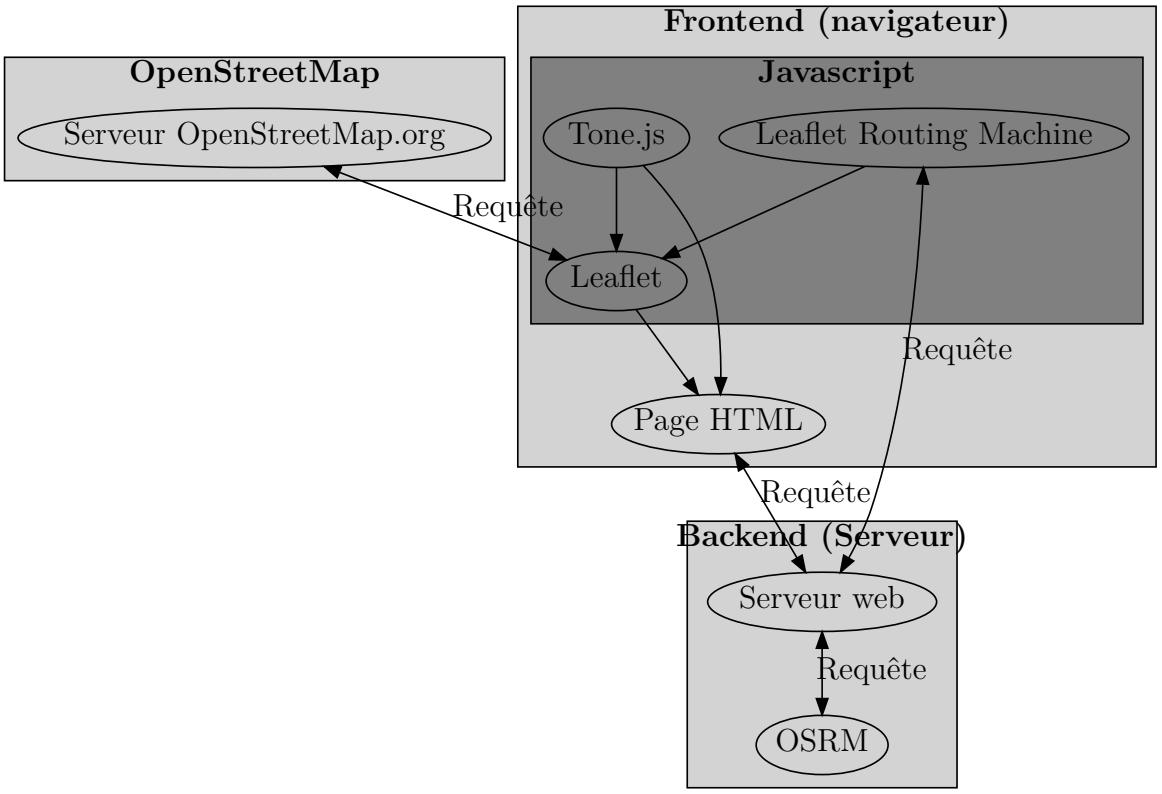


FIGURE B.3 – Architecture finale de l'application, comprenant le frontend, le backend, et le service externe.

La partie *backend* de cette application est simple : elle se contente de servir une page statique comprenant un formatage HTML simple, le code client JavaScript, et la gestion de la requête d'itinéraire expliquée plus haut. Pour cela, lorsqu'une requête HTTP comprenant les options propres à OSRM lui parvient, le serveur fait une nouvelle requête locale, et retourne, sans y toucher, le résultat de cette requête. Cela permet que toutes les requêtes (page et itinéraires) passent par le même serveur Apache, qui se charge d'en détecter le type et de faire la bonne action en conséquence : servir directement les fichiers web, ou faire la requête au serveur OSRM. Dans la figure B.2, on voit que le

serveur web est en mesure de communiquer avec la page, et d'envoyer des requêtes au serveur d'itinéraire, mais il n'a nulle part où les envoyer, parce que le composant chargé d'afficher la carte n'est pas encore intégré.

Le *frontend*, partie du code écrite en JavaScript et exécutée dans le navigateur chez le client, utilise quant à lui plusieurs bibliothèques de cartographie web (figure B.3) : *Leaflet*, qui s'occupe du rendu de cartes glissantes à la manière de Google Maps, et *Leaflet Routing Machine*, chargée de l'itinéraire. Cette dernière bibliothèque permet de créer sur la carte des points d'itinéraire, de départ, d'arrivée, et éventuellement d'étapes ; c'est aussi elle qui fait la requête au serveur d'itinéraire via le serveur web, et qui affiche, d'une part, l'itinéraire sous forme d'instructions, et d'autre part la ligne décrivant cet itinéraire, directement sur la carte. C'est enfin elle qui sera chargée d'appeler une autre bibliothèque, qui générera la musique.

B.4 Procédure de génération musicale

Jusqu'ici, j'ai décrit la méthode que j'ai suivie pour intégrer des données cartographiques existantes dans une page web simple, en me servant autant que possible de données et de programmes locaux. Je parlerai maintenant de ce qui fait la particularité de cette application : la génération de musique à partir de données affichées en temps réel. Ici encore, j'ai été confronté au même problème : il existe une multitude de solutions, plus ou moins complexes. La plus évidente, et la plus difficile, est de faire appel directement à l'API audio du web, en cours de standardisation, et présente sur tous les navigateurs modernes. Les appels à cette API permettent la synthèse sonore à partir d'oscillateurs ou de tampons (*buffers*), et de filtres simples, interconnectés à la façon d'un synthétiseur modulaire. Si cette approche est très puissante pour la manipulation audio, elle est de relativement bas niveau et ne comprend aucune structure de données ou procédure pour faciliter la génération musicale. Il existe des frameworks* intégrant ces possibilités. Comme dans les autres cas de figure, cela implique de chercher une solution appropriée parmi de nombreux candidats plus ou moins aboutis, aux fonctionnalités se

recoupant parfois. J'ai choisi Tone.js [38] après avoir lu les présentations de cinq frameworks différents. Ce dernier est conçu pour la création musicale interactive. À l'inverse d'autres frameworks, il n'intègre pas d'éléments d'interface utilisateur, mais permet de prototyper rapidement un instrument virtuel, et d'en jouer les notes, soit lors d'événements, soit à l'aide de motifs déclarés préalablement. Tone.js comprend également un système permettant de synchroniser de l'animation graphique avec le son. C'est cette dernière fonction qui m'a fait choisir cette bibliothèque, sachant que j'en aurais besoin pour animer la carte. L'intégration a ensuite été assez aisée : à chaque fois que l'utilisateur déplace des marqueurs sur la carte, un nouvel itinéraire est demandé. Lorsqu'il arrive, il est affiché immédiatement par Leaflet, et passe en parallèle vers Tone.js. Ce dernier, d'après les règles de composition choisies, synthétise la mélodie en réagissant au mouvement du point le long de la ligne de l'itinéraire.

Une fois validée la possibilité de jouer du son, je me suis attelé à l'élaboration de l'instrument virtuel. Dans Tone.js, un instrument est décrit en déclarant un objet de type `Synth` (synthétiseur) avec un ensemble de paramètres. Par exemple, l'instrument que j'utilise pour *MusicRoad* consiste en deux oscillateurs, un sinusoïdal et un triangulaire ; ce dernier passe dans une enveloppe ADSR (Attack, Delay, Sustain, Release), et dans un filtre passe-bas. Cette combinaison donne une sorte de gentil piano à l'attaque rapide, sous lequel on peut entendre la note à l'octave inférieure pour plus de profondeur. Cet instrument, à vrai dire, ressemble plutôt à un piano-jouet qu'à un piano de concert, mais le critère premier était surtout de ne pas avoir les oreilles détruites pendant le développement de l'application.

Une fois encore, ce processus m'a confronté à un problème : le projet est celui d'un touche-à-tout, qui ne maîtrise pas tout ce qu'il touche. J'ai fait ce rapide exposé de synthèse audio, non afin d'expliciter chaque composant d'un synthétiseur numérique, mais dans le but d'illustrer que, quoique j'aie quelque connaissance dans ce domaine, cette connaissance s'est avérée insuffisante pour concevoir un instrument véritablement plaisant, ou adapté à son application, en un mot, *designé*.

J'envisage deux solutions possibles à ce problème : la première consisterait à consa-

crer le temps nécessaire à tout apprendre correctement. Ce n'est pas faisable en pratique, le champ de la connaissance humaine étant plus vaste qu'une vie pourrait explorer. La deuxième serait la collaboration. Quel modèle envisager pour une telle collaboration ? Avec qui collaborer ? Rappelons que l'ambition initiale pour ce projet était de tout concevoir moi-même, dans ma tour d'ivoire. À ce stade du prototype fonctionnel, je n'ai pas élucidé cette question. En ce qui concerne les modèles collaboratifs, il serait intéressant de faire suffisamment connaître le projet pour créer une petite communauté de développement sur un dépôt de code public. Elle pourrait utiliser à la fois des procédés et des principes du logiciel libre, mais dans un but de création plutôt que de résolution de problème.

B.5 Conclusion

En décrivant en détail le processus de création [9] du site web MusicRoad, j'ai cherché à déterminer le rôle des outils et données libres en tant que facilitateurs de la création artistique sur le web. De fait, ce projet simple (il consiste en seulement 380 lignes de code, sans compter les différents composants externes, qui comptent des dizaines de milliers de lignes de codes) a permis de valider l'hypothèse que l'existence et l'accès à du code et à des données libres facilitent la création. Le projet a également mis en lumière les difficultés que posent ces mêmes technologies. En effet, en raison de l'aspect abondant et nébuleux de l'écosystème, plus de la moitié du temps a été consacrée à la recherche de programmes et de données existantes, compatibles, et à leur intégration. Les étapes de conception, fabrication, test, debug, ajustements, qui sont d'ordinaire les plus importantes en art numérique, sont presque mineures en comparaison.

Cependant, malgré l'investissement non négligeable que représente la recherche de solutions libres, une fois ces technologies connues et maîtrisées, le gain de temps compense largement cet investissement, pour le reste du processus et pour d'éventuels projets à venir. En effet, l'artiste se repose sur du code existant, éventuellement tenu à jour et

9. J'ai toutefois passé sous silences plusieurs étapes de sélection de bibliothèques.

amélioré par une communauté de développeurs, ce qui peut lui offrir une plus grande efficacité, voire simplement lui permettre d'accomplir sa tâche. Sans l'existence de cette richesse de ressources, de code, de données et de documentation, je peux affirmer que je n'aurais pas pu mener à bien ce projet. D'abord, ce petit projet repose indirectement sur l'effort de milliers d'individus, en particulier en ce qui concerne l'élaboration de la base de données cartographiques (OpenStreetMap). Ensuite, contrairement aux nombreux auteurs des outils utilisés pour le projet, le développement web ne compte pas parmi mes compétences habituelles.

Mais aussi, l'investissement a ceci d'intéressant que la connaissance des standards, procédés et outils, permet de stimuler l'imagination, de faire des rapprochements entre des éléments disparates, de faire du neuf avec du vieux, en un mot de créer. Si le code est l'outil de l'artiste, et le concept son matériau, l'environnement lui pourrait servir d'inspiration. La puissance du logiciel libre et de ses communautés est bien illustrée par le fait que des applications à portée artistique ou expérimentale existent sur Internet, servant à leur tour d'inspiration à d'autres œuvres et outils faciles d'accès, réutilisables et détournables.

Le détournement est un aspect essentiel du projet. Les technologies évoquées ici sont dans la majorité des cas utilisées dans des applications purement cartographiques (localisation, guidage), pédagogiques ou commerciales. Dès lors, on peut parler de détournement puisque le but affiché de l'application est esthétique et ludique. L'utilisateur n'est pas dépaysé par les codes de représentation et d'interface, qui sont omniprésents. Il peut commencer immédiatement à jouer avec la carte, chercher des endroits intéressants musicalement, créer et partager ses trouvailles, ou ses compositions s'il adopte lui-même la démarche créative de placer plusieurs points de trajet, d'écouter, de modifier, et de recommencer.

Annexe C

Les pantins en dehors du studio : Loqmane Bahri

Au cours de mon emploi aux Fées spéciales, j'ai rencontré un jeune réalisateur de film d'animation, Loqmane Bahri, qui était à l'époque en stage dans la société. En tant que membre de l'équipe de fabrication des pantins, il faisait de la colorisation, du détourage et du rig de personnages, et utilisait notamment le système d'autorig. Il a par la suite travaillé aux concepts sur d'autres projets. Après la fin de son stage, il a entrepris un nouveau film en utilisant entre autres certaines techniques apprises pendant son stage. Alors que son projet approchait de sa fin, j'ai parlé avec lui de cette expérience pour essayer de déterminer le rôle que pouvaient avoir la publication d'outils libres d'un studio pour de jeunes artistes comme lui. Je m'appuierai ici sur son témoignage.

La première de mes interrogations était la raison du choix de la technique des pantins pour sa pratique personnelle. C'est une technique que Loqmane avait déjà utilisée pour d'autres projets, mais pas avec les mêmes outils. Son choix est donc d'abord esthétique, comme il le précise :

> « Elle permet dans un premier temps de garder une esthétique fidèle aux traits de l'auteur. En effet, une certaine personne m'a dit un jour que mon style de dessin serait impossible à mettre en œuvre en animation… Je suis content d'avoir enfin

trouvé une technique qui s'adapte parfaitement au style de dessin que j'adopte. [1] »

Cette raison est effectivement la même que celle invoquée par Michel Ocelot, et d'autres adeptes de la technique, et aussi une des raisons premières pour les Fées spéciales. Plus encore que dans le cas de Loqmane, pour le studio, l'esthétique peut être très différente d'un projet à l'autre, selon le médium*, le type de projet, la direction artistique, et les choix et contraintes artistiques. L'intérêt est donc évident par rapport à d'autres techniques d'animation 3D, dans lesquelles une traduction doit forcément être faite entre le style visuel des concepts et celui des images animées du film. Mais comme l'explique l'artiste, son choix est également technique :

« De plus, c'est une technique qui permet d'effectuer des corrections rapidement. En effet, si je dois corriger un petit défaut de la tête, ou modifier une main, il me suffit de corriger directement sur le PNG en question ou de redessiner une nouvelle main par exemple. C'est une technique modulable qui permet cela.

« Je la trouve assez rapide à mettre en œuvre une fois la prise en main de celle-ci. De plus elle est tout de suite très efficace lorsque le découpage du pantin est top.

« C'est une technique qui prend moins de temps que la 3D et donc moins onéreuse en coût de production et de temps de vie, ahah. »

Cette analyse de l'artiste est éclairante, car elle montre un autre aspect de la technique : sa souplesse, et par extension sa scalabilité*. Comme il le dit, les corrections sont rapides. Il explique avoir dessiné et animé les personnages seul dans son projet, assisté par des collaborateurs pour le rig. La situation est donc plus simple que pour un long-métrage, réalisé par une grande équipe au sein d'un studio, mais a fortiori dans ce dernier cas, le gain de temps est d'autant plus appréciable qu'il y a plus de personnages, de plans*, et que les réalisateurs sont plus exigeants — et demandent plus de retakes*.

Un autre intérêt que souligne Loqmane est d'ordre pédagogique, car la technique demande un investissement moindre que celui de la 3D :

« Pour l'animation c'est très pratique surtout pour une personne qui « débute » dans l'animation de personnage. La prise en main est rapide !! Je n'ai pas de

1. Entretien par courriel avec l'auteur, 22 mars 2019.

« formation » d'animateur mais je peux tout de même animer les pantins.

« La 3D ne m'attire absolument pas pour le moment. »

Cet aspect du discours m'intéresse tout particulièrement. Comme il le dit, Loqmane a de l'expérience pour avoir réalisé plusieurs films durant ses études, mais il a reçu peu de formation théorique dans l'art et la technique de l'animation. Contrairement à l'animation 3D, qui demande tant de connaissances qu'elles demandent à être divisées en spécialisations, l'animation des pantins, même numériques, peut être assez rapidement apprise, et entièrement maîtrisée par une seule personne sur un projet. Paradoxalement, Blender est un programme d'une grande complexité, mais grâce au processus des pantins, il peut être utilisé par des artistes ne connaissant qu'une petite partie de ses fonctions — quitte à en apprendre plus par la suite.

Sachant que Loqmane avait appris les outils conçus au sein du studio, je l'ai questionné sur l'utilisation qu'il en faisait dans sa pratique personnelle. Mais je sais également qu'il existe d'autres outils que ceux des Fées spéciales, libres ou non, pour m'en être en partie inspiré en premier lieu. J'ai donc demandé quels outils il aurait choisi si ceux-ci n'avaient pas existé, et si même il aurait animé des pantins :

« Oui, j'aurais tout de même animé des pantins avec Blender en m'inspirant des pantins papier type Michel Ocelot. Si je n'avais pas connu Blender je serais allé vers After Effects qui permet la même technique mais plus complexe à mettre en place. […] Pendant un an, 2015-2016, j'ai utilisé After Effects pour mon premier film. Un collègue m'a fait découvrir DUIK, mais au départ dur dur de comprendre comment ça marche… on a galéré comme des fous furieux à comprendre, c'était horrible. Jamais plus je ne l'utiliserai. Vive Blender et les pantins ! […] C'est très efficace, on s'y retrouve facilement quand on anime les pantins. L'ergonomie des controllers est très bien pensée. Cela permet une production de masse de pantins avec un protocole assez clair, je pense.

« De plus, le rig des fées est une excellente base si on veut pousser encore plus loin cette technique. Ce qui est bien c'est qu'on peut l'adapter, le modifier, en fonction du genre des pantins. »

L'affirmation que les outils des Fées étaient plus faciles à utiliser que DUIK m'a beaucoup surpris, car sans l'avoir utilisé beaucoup moi-même, j'en connais les fonctionnalités, et j'en ai surtout entendu des compliments de la part d'animateurs. Peut-être est-il trop complet, ou trop complexe à utiliser pour un non-spécialiste. Un autre point intéressant est la notoriété des outils cités : ils ne sont pas forcément les plus connus dans le milieu par les animateurs, qui apprennent plus fréquemment ToonBoom pour le film, ou Spine pour le jeu vidéo.

Un dernier point soulevé par Loqmane est complémentaire aux fonctionnalités du programme, et concerne son support, et en particulier la facilité d'installation. En effet, au sein du studio, les outils de pantins font partie du pipeline*, et des solutions ont été mises en œuvre pour qu'ils soient automatiquement déployés sur tous les postes de travail, pour chaque graphiste. C'est loin d'être le cas dès que les outils sont publiés et utilisés en dehors de la société qui les a créés. La majorité des programmes open source sont rendus disponibles sur des forges, des services comme GitHub ou GitLab, qui permettent de faciliter le développement collaboratif, le déploiement, et le support technique des programmes, en particulier pour des programmeurs « amateurs », dont l'édition de logiciels n'est pas l'activité principale. C'est le cas pour les scripts* des Fées spéciales, publiés entre autres sur GitHub. Mais le support technique sur une telle plate-forme n'est pas des plus intuitifs pour qui n'a pas un minimum d'expérience de programmation. Loqmane raconte par exemple avoir eu des difficultés à installer les outils :

> « Dur dur au départ quand Kévin [collaborateur au studio] et toi aviez essayé d'installer le tout sur mon ordinateur. Souviens-toi, je suis sur Windows, il a fallu que tu installes je ne sais plus quoi pour pouvoir commencer à installer les outils des fées. Je ne sais plus si vous l'aviez fait mais je pense qu'il serait judicieux de faire un tuto, une page [...] dédiée à l'installation des outils qui explique de A à Z le processus d'installation que l'on soit sur Windows ou autre. La difficulté d'installation peut en rebuter plus d'un. »

Il est indéniable que le fait de concevoir et maintenir un outil qui réponde à la fois aux

besoins internes du studio et à ceux de tous les utilisateurs potentiels demande du temps, que l'on n'a pas nécessairement en production. De fait, beaucoup de programmes libres ont l'habitude d'être fournis « en l'état », comme l'indiquent la plupart des licences libres, c'est-à-dire sans garantir qu'ils fonctionnent pour un usage précis, ou même tout court. Cela ne veut pas dire que les développeurs de logiciels libres ne fassent jamais attention à l'expérience de l'utilisateur, mais que dans un contexte comme le nôtre, elle est secondaire. Et même si Loqmane avait connu le système de support technique intégré à GitHub, il n'est pas certain qu'il l'aurait utilisé pour demander de l'aide, et il est encore moins certain que les créateurs du programme aient eu le temps d'y répondre.

Mais malgré les difficultés d'installation, le simple fait que Loqmane, qui a peu de connaissances en 3D, moins encore en scripting, qui n'a pas de studio de fabrication pour l'appuyer, ni une équipe de TD* ou de R&D* pour l'assister, a pu réaliser avec succès son film en utilisant les outils développés au départ pour un usage interne, est révélateur du rôle potentiel du logiciel libre pour les artistes individuels, dont les seules ressource sont leurs propres argent, savoir-faire et temps.

Annexe D

Méthode pour trouver le nombre de films dont le scénariste est aussi réalisateur

Dans la section I.1.2.1, j'affirme que beaucoup de films sont écrits et réalisés par la même personne. Mon collègue Flavio Perez m'a fait remarquer à juste titre que cette affirmation était douteuse. Pour le vérifier, j'ai téléchargé un extrait de la base de données du film IMDb[1]. J'ai choisi les fichiers `title.basics.tsv`, décrivant les films et `title.crew.tsv`, listant les réalisateurs et scénaristes. J'ai ensuite créé une base de données SQLite3 pour comparer les informations des films et des créateurs, et ne garder que les entrées pertinentes. J'utilise pour cela le script suivant :

```
--   Copyright © 2019 Damien Picard <dam.pic@free.fr>
-- This work is free. You can redistribute it and/or modify it under the
-- terms of the Do What The Fuck You Want To Public License, Version 2,
-- as published by Sam Hocevar. See http://www.wtfpl.net/ for more details.
```

1. Cette base de données non-libre est disponible à l'adresse `https://www.imdb.com/interfaces/`. Il n'existe pas de base de données ouverte (cf. section I.2.4) sur les œuvres cinématographiques, c'est-à-dire pouvant être utilisée à n'importe quelle fin, y compris commerciale, modifiée, agrégée et redistribuée. L'utilisation de ces données demande d'afficher le message suivant : Information courtesy of IMDb (`http://www.imdb.com`). Used with permission.

```
-- Création des tables contenant les films et les équipes
CREATE TABLE movies
  (tconst TEXT, titleType TEXT, primaryTitle TEXT, originalTitle TEXT,
   isAdult BOOL, startYear INT, endYear INT, runtimeMinutes INT, genres TEXT);
CREATE TABLE crew
  (tconst TEXT, directors TEXT, writers TEXT);

-- Passer en mode tabulation pour respecter le format des fichiers
.mode tab
-- Importer les fichiers dans les tables précédemment créées
.import ./title.basics.tsv movies
.import ./title.crew.tsv crew

-- Passer en mode CSV (valeurs séparées par des virgules),
-- et exporter la requête suivante
.mode csv
.output title_crew.csv

-- Requête joignant les deux tables et sélectionnant les entrées pertinentes
SELECT * FROM movies INNER JOIN crew USING (tconst) WHERE (
  -- Sélectionner les films (œuvres de type 'movie')...
  titleType == 'movie'
  -- et dont le genre est 'Animation'...
  AND INSTR(genres , 'Animation')
  -- et dont le réalisateur et le scénariste sont renseignés
  AND directors != '\N'
  AND writers != '\N'
);
```

J'obtiens le fichier `title_crew.csv`. J'utilise ensuite ce script Python pour déterminer quels films ont été écrits et réalisés par au moins une personne :

```
#!/usr/bin/env python3
#
#   Copyright © 2019 Damien Picard <dam.pic@free.fr>
# This work is free. You can redistribute it and/or modify it under the
```

```
# terms of the Do What The Fuck You Want To Public License, Version 2,
# as published by Sam Hocevar. See http://www.wtfpl.net/ for more details.

import csv

# Ouvrir le fichier CSV et en lire les lignes
with open('./title_crew.csv') as f:
    rows = list(csv.reader(f))

# Initialiser deux compteurs à 0
same = 0
diff = 0

# Pour chaque ligne,
for r in rows:
    # Récupérer les deux dernières valeurs,
    # correspondant aux scénaristes et aux réalisateurs
    directors, writers = r[-2:]
    # Il peut y avoir plusieurs scénaristes ou plusieurs réalisateurs
    # On crée donc des ensembles (set()) pour les comparer
    directors = set(directors.split(','))
    writers = set(writers.split(','))
    # Intersection des deux ensembles.
    if directors & writers:
        # Si l'intersection contient au moins un élément, on incrémente same
        same += 1
    else:
        # Sinon, on incrémente diff
        diff += 1

# Afficher le nombre de lignes où un élément est contenu
# dans les deux colonnes, et le nombre où ce n'est pas le cas
print("Identique:", same, "\nDifférent:", diff)
# Afficher le pourcentage
print("{:0.2f} %".format(same / (same + diff) * 100.0))
```

Annexe E

Guide d'entretien semi-directif

Confidentialité

— Puis-je enregistrer cet entretien ?

But de la recherche, positionnement

— Thèse de doctorat sur FLOSS dans l'animation
— Orientation technique, éventuellement artistique, donc questions à des techniciens (TD…)

Utilisation

— Utilises-tu le logiciel libre en production ?
— Depuis quand ?
— Si oui, pourquoi avoir changé ?
— Dans quels domaines ?
  - ☐ OS
  - ☐ création
    - ☐ DCC 3D
    - ☐ texturing / painting
    - ☐ compositing
    - ☐ montage
    - ☐ simulation
    - ☐ autre :
  - ☐ langages et environnements de développement

- ☐ python ou autre langage script
- ☐ VCS
- ☐ IDE
- ☐ autre :

☐ pipe

- ☐ asset management
- ☐ production management
- ☐ autre :

☐ outils divers

- ☐ gestion de ferme de calcul
- ☐ encodage de médias
- ☐ format d'image
- ☐ étalonnage / color pipeline
- ☐ communication
- ☐ autre :

— Quelle utilisation fais-tu du logiciel libre ?
— Quels intérêts techniques vois-tu au logiciel libre ?
  — Innovation ?
  — Accès au code, extensibilité, adaptation au pipeline ?
  — Concernant les outils ? les DCCs ?
  — Pérennité ? (exemple XSI)
— Quels freins vois-tu à l'adoption du logiciel libre ?
  — Investissement R&D, temps de développement ?
  — Limitations techniques ? pipeline ? artistiques ? juridiques ?
  — Amateurisme ?
  — Problèmes de sécurité ?
  — Support technique ?

Contribution

— As-tu contribué à des projets de logiciel libre ?
  — De quelle manière ? (utilisation, design, code, bugfix, rapport d'erreur, suggestions…)
  — Si publication : outil maison ? lib/framework ? application ?

Juridique

— T'es-tu renseigné sur les licences logicielles pour utiliser ou publier du code ?
  — As-tu déjà lu le texte d'une licence libre ?
  — As-tu déjà consulté un avocat concernant le droit d'auteur ?
— Fais-tu une distinction entre Logiciel libre et Open Source ?
  — Cette distinction est-elle importante pour ton travail ?
  — De manière générale ?

Données, contenus libres

— As-tu déjà vu des films sous licences ouvertes ? (ex. CC)
— T'intéresses-tu à ce type de contenus ?
  — Économiquement ? (financement, crowdfunding, etc.)
  — Pour les possibilités créatives ? (Remix, réutilisation)
  — De diffusion ? (copie libre)
  — Éthiquement ? (société du partage)

Talon

— Nombre de salariés sur une prod. moyenne :
— Secteur
  ☐ Animation
  ☐ Post-production / VFX
  ☐ Autre :
— Poste
  ☐ TD
  ☐ Dev
  ☐ Graphiste
  ☐ Production
  ☐ Autre :

Document rédigé en Org mode dans GNU Emacs et composé avec LaTeX.