



Université de Montpellier
Faculté des Sciences

MASTER 2 INFORMATIQUE
Parcours Imagine

**Derrière les Coulisses des VFX : Recherche et
Développement chez Benuts**

Rapport de stage

effectué à Benuts
du 29/01/2024 au 28/06/2024
par Adèle IMPARATO

Tuteur en entreprise : Arthur TASQUIN
Tuteur à l'Université : Noura FARAJ

Je tiens à remercier toute l'équipe de Benuts qui m'a accueillie durant ces cinq mois, en particulier mon maître de stage Arthur, mais aussi Brieuc, Simon, Jonathan et Gilles qui ont suivi mes travaux de près et ont consacré de leur temps pour me transmettre leurs connaissances. Je tenais aussi à citer Milan, Pierre et Colin, sans qui cette expérience en entreprise n'aurait pas été la même. Je souhaite également remercier Madame Noura FARAJ, mon encadrante universitaire ainsi que mes parents pour la relecture de mon rapport.

Table des matières

1	Introduction	3
2	Entreprise et environnement technique	4
	2.1 Activités	4
	2.2 Pipeline de production	5
	2.3 Outils	9
	2.4 Organisation	10
	2.5 Ambiance de travail	10
3	Mission	12
	3.1 Rôle de développeur	12
	3.2 Département temps réel	12
4	Travaux effectués	16
	4.1 Recherche sur Unreal Engine	16
	4.1.1 Contexte	16
	4.1.2 Résultats	16
	4.2 Création d'un outil de fusion	18
	4.2.1 But	18
	4.2.2 Description	18
	4.2.3 Implémentation	19
	4.2.4 Résultats	22
	4.3 Création d'un outil de dispersion	24
	4.3.1 But	24
	4.3.2 Description	24
	4.3.3 Implémentation	25
	4.3.4 Résultats	28
	4.4 Création d'un outil de détection faciale	30
	4.4.1 But	30
	4.4.2 Description	31
	4.4.3 Implémentation	32
	4.4.4 Résultats	32
5	Réflexions	36
6	Conclusion	37
	Annexes	40

1 Introduction

Vous avez certainement déjà entendu parler de Georges Méliès, le réalisateur du film muet "Le Voyage dans la Lune", célèbre pour la scène où une fusée est envoyée dans l'oeil de la lune. Ce court-métrage, sorti en 1902, marque les débuts des effets visuels (VFX) [35]. Depuis lors, les VFX n'ont fait qu'évoluer, des premiers trucages jusqu'aux avancées notables dans des films emblématiques tels que "La Charette Fantôme", "Mary Poppins", "Star Wars", "Le Seigneur des Anneaux" ou encore "Avatar" (Figure 1). À travers ce rapport, je vous propose d'entrer dans les coulisses des VFX en suivant l'avancée de mon stage industriel, qui a eu lieu dans une entreprise nommée Benuts, située à La Hulpe, à deux pas de Bruxelles en Belgique, du 29 janvier 2024 au 28 juin 2024. Cette expérience a eu lieu dans la cadre de mon Master en Informatique parcours Imagine et avait pour but de me familiariser avec le monde professionnel, de mettre en pratique mes compétences et d'en acquérir de nouvelles.

Benuts est un studio spécialisé dans les effets visuels, principalement dédié au cinéma, mais également actif dans la réalisation de projets de *motion design* (ou conception graphique animée), ainsi que de rendu temps réel.

Dans ce rapport de stage, je vous présenterai d'une part l'entreprise, qui m'a accueillie durant ces cinq mois, ainsi que son environnement technique : ses activités, sa pipeline de production, ses outils, son organisation sans oublier l'ambiance de travail. D'autre part, je décrirai ma mission en commençant par le rôle de développeur au sein d'une telle pipeline ainsi que le fonctionnement du département temps réel. Ensuite, je rentrerai dans le vif du sujet en décrivant les tâches que j'ai accomplies (i.e. recherches et outils développés) ainsi que les technologies que j'ai été amenée à utiliser. Pour finir, je partagerai quelques réflexions personnelles sur mon expérience de stage, notamment les surprises rencontrées et les perspectives envisagées, avant de conclure ce rapport par un bref bilan de ce que j'ai appris.

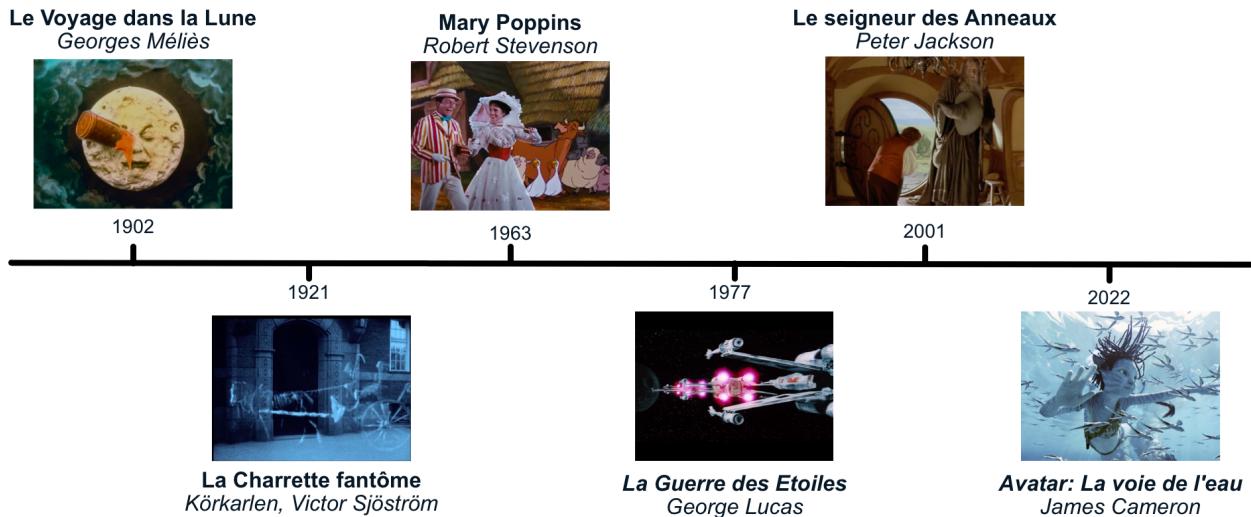


FIGURE 1 – Dates clés dans l'histoire des trucages et effets spéciaux cinématographiques. [37]

2 Entreprise et environnement technique

Benuts est un studio spécialisé dans les effets visuels, c'est-à-dire l'ensemble des techniques utilisées pour manipuler des séquences visuelles numériquement. Une centaine de personnes travaillent chez Benuts, majoritairement des artistes 2D/3D, ayant chacun leur spécialité. L'entreprise est établie dans les trois régions de Belgique : en région flamande à Malines, à Bruxelles-Capitale et en région wallonne à La Hulpe, là où j'ai effectué mon stage. Cette répartition géographique permet entre autres à l'entreprise de bénéficier des avantages et des budgets alloués par chacune des trois régions. Depuis plus de 15 ans, le studio est actif dans de nombreux projets, tant en Belgique qu'à l'international. Si leur nom vous est inconnu, il n'en est certainement pas de même pour leurs créations, avec des projets notables comme des clips musicaux pour Angèle^[5] et Stromae^[34], ou encore les effets visuels de "Astérix et Obélix : L'Empire du Milieu"^[13].

2.1 Activités

De manière générale, les activités du studio se regroupent autour de trois pôles : les VFX, le *motion design* et le rendu temps réel (Figure 2). Les VFX se distinguent des SFX (effets spéciaux) qui sont les effets réalisés sur le plateau de tournage tels que la génération de phénomènes météorologiques, le maquillage des acteurs, les trucages, etc. Contrairement aux VFX, qui sont exclusivement numériques, les SFX sont physiques et se produisent en temps réel devant la caméra.

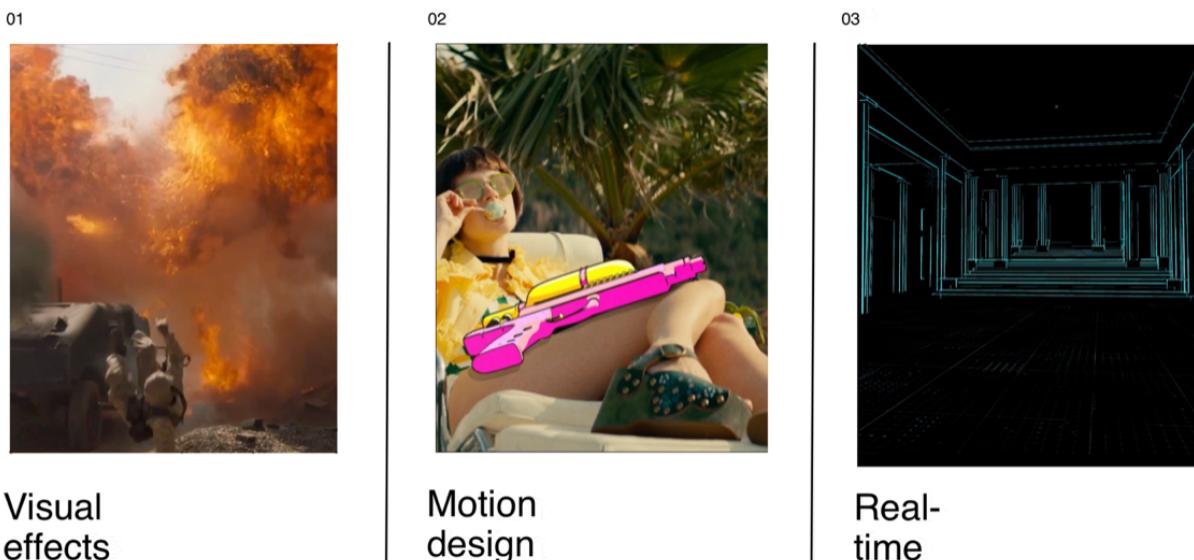


FIGURE 2 – Les trois activités principales de Benuts. Schéma édité du site officiel [11].

Les VFX regroupent les tâches telles que la création d'éléments 3D (objets, décor, personnages), la simulation de phénomènes physiques (explosions, fluides, fumée, etc.), l'animation de personnages et d'objets ainsi que le *compositing* de différentes couches d'images qui constitue généralement l'étape finale de l'édition des séquences. Le *motion design*, quant à lui, va plutôt regrouper des tâches telles que la création des génériques de début ou de fin de film, des titres, des transitions entre les scènes, des animations d'informations et d'autres éléments graphiques animés. Les missions de rendu temps-réel, quant à elles, concernent plutôt les tâches de *virtual production* (production virtuelle), comme des prévisualisations dites "préviz", du *virtual scouting* (repérage virtuel) ou encore du *in-camera VFX* (effets visuels intégrés à la caméra) [16], mais incluent aussi les expériences en VR (réalité virtuelle) ou sur écran tactile (plus de détails en section 3.2).

Il arrive aussi que Benuts participe à l'élaboration de *storyboard*, de *moodboard* ou encore de dessins/vidéos de concepts, qui vont servir à visualiser des idées pendant l'étape de pré-production du film, avant le tournage. De plus, des superviseurs FX viennent systématiquement sur le plateau de tournage pour s'assurer que les séquences filmées pourront être aisément éditées en post-production. Cela passe par noter des informations sur la caméra utilisée, placer des points de suivi (*tracking points*) sur les décors ou encore s'assurer qu'il n'y a pas trop de reflets sur l'écran vert pour simplifier son remplacement.

2.2 Pipeline de production

Pour un projet dit classique, c'est-à-dire pour la réalisation des effets visuels d'un film, Benuts opère à trois moments : en pré-production, pendant la production et en post-production. Les compétences du studio sont donc réparties entre ces trois phases distinctes, comme le montre la Figure 3 :



FIGURE 3 – Récapitulatif des compétences de Benuts en pré-production, production et post-production.

Les étapes de création d'effets visuels proprement dits se retrouvent durant la phase de post-production et peuvent être résumées comme ceci :

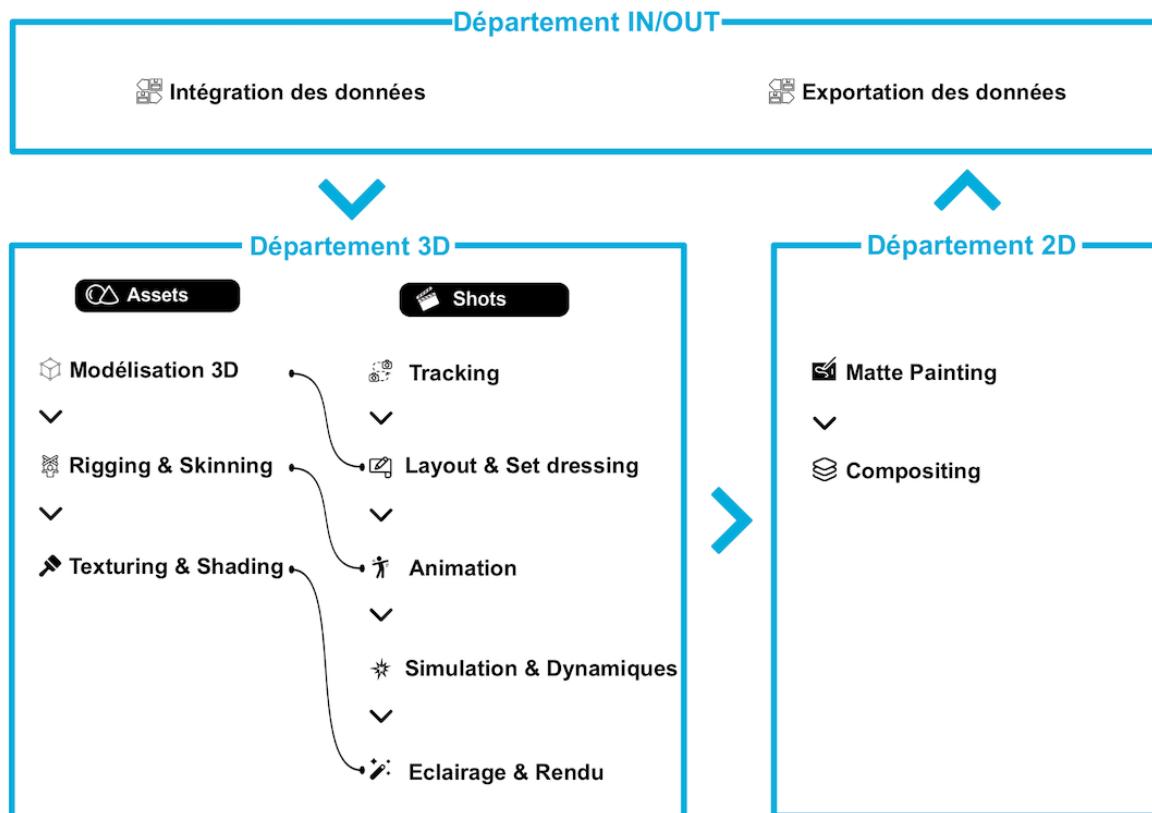


FIGURE 4 – Étapes de post-production divisées en trois départements.

- Intégration des données brutes** : L'intégration (*ingest*) des données brutes consiste à transférer les vidéos et les images envoyées par le client aux départements concernés, et ce, dans le bon format et en veillant à transmettre les données complémentaires ou métadonnées telles que le *frame range* (plage d'images), la résolution, les informations sur les caméras, etc.
- Tracking** : Le *tracking* consiste à suivre le mouvement d'objets dans une séquence vidéo. Cela peut être utilisé pour intégrer des éléments virtuels dans une scène filmée en assurant qu'ils suivent les mouvements de la caméra ou des objets réels.
- Modélisation 3D** : La modélisation est le processus de création de géométries 3D pour représenter des objets ou des personnages dans un environnement virtuel. Ces modèles peuvent être créés à partir de zéro ou bien récupérés en ligne via des banques de modèles 3D. Parfois, ils sont également obtenus en scannant directement des objets du monde réel, c'est ce qu'on appelle de la photogrammétrie.

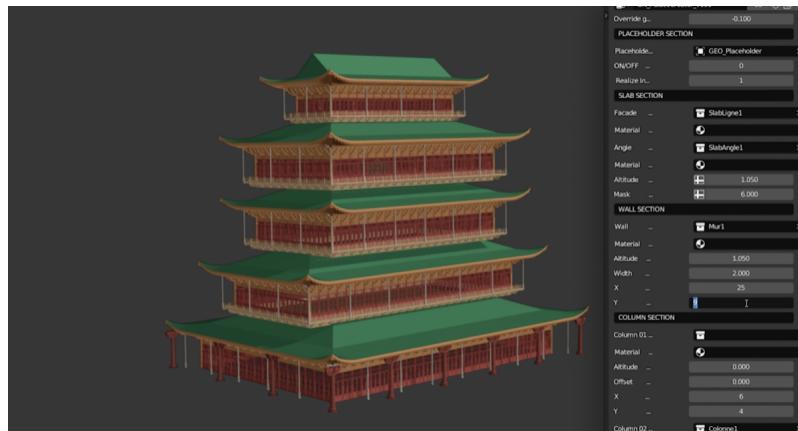


FIGURE 5 – Exemple de modélisation 3D d'un élément de décor. Image extraite du *making of* de "Astérix et Obélix, l'Empire du Milieu" [9].

- Rigging & Skinning** : Le *rigging* consiste à créer un squelette numérique (ou *rig*) pour un personnage ou un objet animé. Ce squelette est utilisé pour contrôler les mouvements et déformations de l'objet lors de l'animation. Le *skinning*, quant à lui, permet de contrôler les déformations de la peau pour les personnages ou objets animés en 3D.



FIGURE 6 – Exemple de création de flammes virtuelles ajoutées à l'environnement réel. Image extraite du *making of* de "Astérix et Obélix, l'Empire du Milieu".

- Texturing & Shading** : Le *texturing* est le processus de création de textures, c'est-à-dire d'images ou de motifs appliqués sur la surface d'un modèle 3D pour lui donner un aspect visuel détaillé. Le *shading* est la création de matériaux en assemblant les textures de manière cohérente afin que le

modèle réagisse de différentes manières à la lumière, simulant ainsi des propriétés comme la réflexion, la réfraction, et la diffusion.

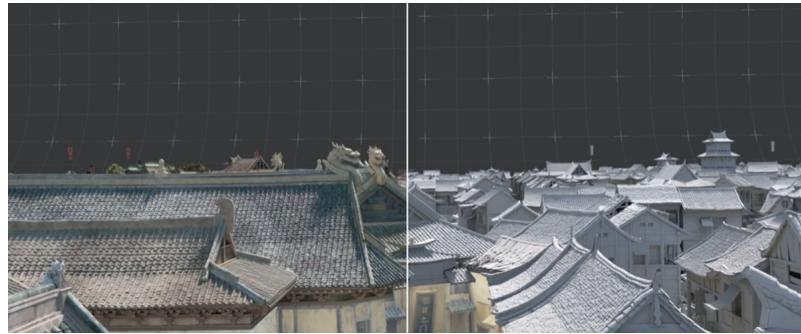


FIGURE 7 – Exemple de textures ajoutées à des modèles 3D de toits. À droite, les modèles sans textures. À gauche, les modèles texturés. Image extraite du *making of* de "Astérix et Obélix, l'Empire du Milieu".

6. **Layout & Set dressing :** Le *layout* implique la planification de l'emplacement de la caméra lors de l'ajout de nouveaux objets à la scène. Cette phase comprend également le positionnement approximatif des nouveaux éléments 3D dans la scène, permettant d'estimer le niveau de détail requis pour chaque objet. C'est ensuite le *set dressing* qui fixe les objets dans la scène, en se basant sur le *layout* préalablement défini.

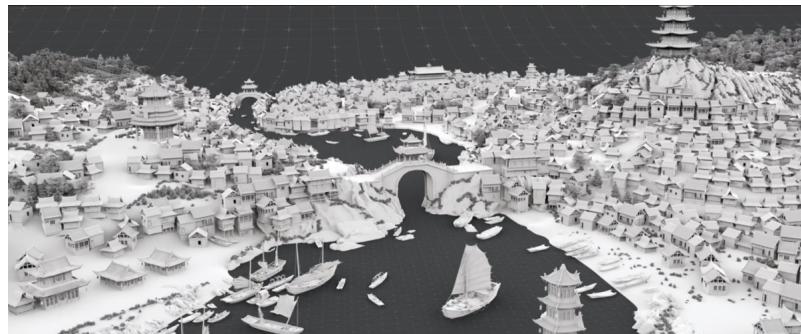


FIGURE 8 – Exemple de placement des objets 3D pour la création d'un environnement virtuel. Image extraite du *making of* de "Astérix et Obélix, l'Empire du Milieu".

7. **Simulation & Dynamiques :** La simulation et les dynamiques sont utilisées pour simuler des phénomènes physiques tels que les fluides, les explosions, la fumée, etc.



FIGURE 9 – Exemple de création de flammes virtuelles ajoutées à l'environnement réel. Image extraite du *making of* de "Astérix et Obélix, l'Empire du Milieu".

8. **Animation** : L'animation permet de donner vie aux objets ou personnages en créant des mouvements et des actions. Les animations peuvent être réalisées manuellement en déplaçant les contrôles du *rig* ou en utilisant des techniques de *motion capture*, qui capturent les mouvements réels d'acteurs pour les appliquer aux personnages numériques.
9. **Éclairage & Rendu** : L'éclairage et le rendu sont les étapes finales où la scène 3D est éclairée virtuellement et rendue en images 2D. Cela comprend la configuration des sources lumineuses, la gestion des ombres et des reflets, ainsi que le rendu de haute qualité.



FIGURE 10 – Exemple d'un environnement 3D éclairé et rendu, prêt à être envoyé au département 2D. Image extraite du *making of* de "Astérix et Obélix, l'Empire du Milieu".

10. **Matte painting** : Le *matte painting* est une technique utilisée pour créer des arrière-plans ou des environnements virtuels peints à la main. Ces éléments sont intégrés aux images en 2D pour étendre ou modifier l'environnement réel ou virtuel.

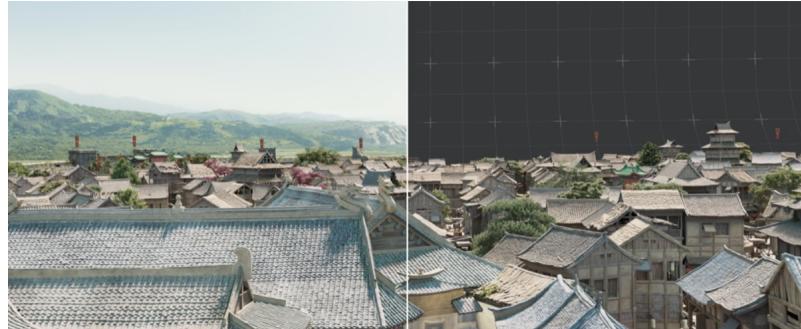


FIGURE 11 – Exemple d'un arrière plan peint numériquement à la main, et ajouté à un décor 3D en partie virtuel et réel. À droite, l'environnement 3D sans arrière-plan. À gauche, avec l'arrière-plan ajouté. Image extraite du *making of* de "Astérix et Obélix, l'Empire du Milieu".

11. **Compositing** : Le *compositing* consiste à combiner différentes couches d'images ou d'éléments visuels pour créer une seule image finale. Cela peut inclure l'ajout d'effets spéciaux, de fonds, de personnages, d'éléments 3D, etc. Cette étape finalise le processus de création visuelle. Elle peut également inclure des techniques telles que la rotoscopie et le *cleaning* (nettoyage), qui permettent d'isoler des parties spécifiques pour un traitement ultérieur ou de supprimer des éléments indésirables de la scène comme les câbles de suspension ou encore des reflets où l'on voit apparaître l'équipe technique.



FIGURE 12 – Exemple de sélection des câbles de suspension pour leur suppression. Image extraite et modifiée du *making of* de "Astérix et Obélix, l'Empire du Milieu".

12. **Exportation des données :** Une fois les images d'une séquence éditées, elles sont renvoyées au client dans le format souhaité et selon la nomenclature demandée.

L'ordre de ces étapes peut varier d'un projet à l'autre, mais dépend surtout des retours du client. En effet, tout au long de la production, le client, souvent le réalisateur du film, peut exprimer ses commentaires et demander des modifications, ce qui peut nécessiter de revenir en arrière sur certaines étapes. Ces ajustements, connus sous le nom de *retakes*, sont quasiment inévitables et influent sur le processus de création des effets visuels.

2.3 Outils



Blender [17] est un logiciel de modélisation 3D polyvalent et open-source utilisé pour créer des modèles, des animations et des effets visuels. **Cycles Render** est son moteur de rendu intégré.



Maya [7] est un logiciel de 3D développé par Autodesk, utilisé principalement pour l'animation, qui permet aussi de faire de la modélisation et du rendu 3D. **Arnold** est son moteur de rendu intégré.



Houdini [33] est un logiciel de création d'effets visuels et d'animation 3D, développé par SideFX, privilégié pour ses capacités dans les simulations physiques et la modélisation procédurale. **Karma** (ou Mantra, selon la version de Houdini) est son moteur de rendu intégré.



3DE [32] est un logiciel de suivi de mouvement (tracking), développé par Science-D-Visions, qui permet le suivi précis de la caméra et des objets.



ZBrush [30] est un logiciel de sculpture numérique et de modélisation 3D développé par Pixologic, privilégié pour la création de modèles 3D organiques (tels que des personnages ou des créatures). ZBrush permet aux artistes de sculpter et de peindre directement sur leurs modèles avec une grande précision et une grande flexibilité.



Substance Painter [2] est un logiciel de *texturing*, développé par Allegorithmic (maintenant une division d'Adobe), qui offre des outils avancés de peinture, de brosses, de calques et de masques pour permettre aux artistes de donner vie à leurs modèles avec des détails et des effets visuels impressionnants.



Unreal Engine [36] est un moteur de jeu vidéo développé par Epic Games utilisé pour la prévisualisation, la création d'environnements virtuels interactifs, l'animation et les rendus en temps réel.



Adobe Photoshop [1] est un logiciel de retouche d'images (en 2D) et de création graphique développé par Adobe. Il offre une large gamme d'outils pour la retouche photo, la peinture numérique, le *compositing*, la création d'effets visuels et permet également de réaliser du *matte painting*.



Nuke [18] est un logiciel de *compositing* nodal développé par Foundry, utilisé pour combiner des éléments visuels provenant de différentes sources, ajouter des effets spéciaux, des corrections de couleur, des titres, et plus encore.



RealityCapture [31] est un logiciel de photogrammétrie qui permet de créer des modèles 3D à partir de photos ou de scans laser, facilitant ainsi la création d'environnements et d'objets numériques réalistes détaillés. Ce logiciel a été développé par Capturing Reality, ensuite racheté par Epic Games.

2.4 Organisation

La communication au sein de Benuts se fait via plusieurs canaux : sur **Discord**[20] pour les informations courtes et informelles (et les gifs rigolos), par email pour les annonces officielles, et bien sûr oralement en interne pour les discussions plus approfondies. Durant mon stage, j'ai été suivie quotidiennement par mon maître de stage avec des entretiens personnels réguliers.

Pour partager des fichiers, Benuts dispose de serveurs internes accessibles depuis tous les ordinateurs du studio. De plus, des machines dédiées à l'exécution des rendus sont mises à disposition, formant ce qu'on appelle la *Render Farm* (Ferme de Rendu).

À cela s'ajoute **Notion**[21], un logiciel qui permet d'organiser toutes sortes de ressources comme des liens, des documentations, des notes, etc. Enfin, **Flow Production Tracking**[6] (anciennement Shotgrid) est utilisé pour le *versioning* et le *time logging*, permettant de publier les versions des séquences d'une étape à l'autre et de faciliter les révisions. De plus, cet outil centralise toutes les informations de production, y compris les tâches, les ressources et les échéances, ce qui améliore la gestion de projet et la coordination entre les équipes.



FIGURE 13 – Discord, Notion et Flow Production Tracking.

2.5 Ambiance de travail

Pour conclure cette section, j'avais envie de parler de l'ambiance de travail chez Benuts. Le studio de La Hulpe, où j'ai effectué mon stage, dispose d'un grand open space où sont rassemblés une cinquantaine de travailleurs. En tant que stagiaire, je n'ai pas eu l'opportunité de faire du télétravail, ce qui signifie

que j'ai passé toutes mes journées de stage dans cet open space, travaillant sur mes tâches et échangeant avec mes collègues.

Dès mon arrivée chez Benuts, j'ai été accueillie par Arthur, mon maître de stage, mais aussi par Brieuc, le *Head of 3D*. Arthur m'a présenté le département temps-réel dont il fait partie, m'expliquant son fonctionnement et me présentant ses membres : Jonathan, Pierre et Milan, avec qui j'ai passé la majeure partie de mon temps. Jonathan et Arthur sont les mentors du département. C'est donc eux qui ont suivi de près mon travail sur Unreal Engine, prenant le temps de m'expliquer les concepts de base et de me donner des retours régulièrement. Une fois un outil terminé, je le faisais également tester par Milan et Pierre pour obtenir des retours supplémentaires.

En parallèle, j'ai eu droit à des sessions journalières avec Brieuc durant les quinze premiers jours de mon stage. Ces sessions avaient pour objectif de me présenter l'ensemble des départements de Benuts et de m'expliquer le fonctionnement des VFX, ainsi que celui du cinéma et de la photographie. J'ai vraiment apprécié avoir ces moments privilégiés qui m'ont permis de comprendre non seulement mes missions, mais aussi le fonctionnement global du studio.

J'ai également beaucoup collaboré avec Simon et Gilles, les deux développeurs du studio qui ont suivi mes travaux de pipeline et m'ont informé sur le fonctionnement de celle-ci. J'ai ainsi été très bien encadrée, et on a pris le temps de me donner des explications à chaque fois que j'en ai eu besoin, ce qui a réellement contribué à la réussite de mon stage.

De manière générale, les personnes que j'ai rencontrées chez Benuts sont très sympathiques. Je me suis facilement intégrée et me suis rapidement sentie à l'aise dans mon environnement de travail. Étant une personne très extravertie, cela a été une motivation supplémentaire pour me lever chaque matin (ou presque) avec enthousiasme. Bien que le secteur des VFX soit exigeant et demande beaucoup d'efforts aux travailleurs, il y avait un esprit de groupe assez convivial. Par exemple, lors des pauses midi en groupe, souvent agrémentées de frites partagées, ou encore lors des cours de néerlandais, donnés les vendredis midi.

3 Mission

Ma mission pendant ce stage comportait deux responsabilités distinctes, toutes deux axées sur la recherche et le développement. D'une part, j'ai collaboré avec les développeurs de l'entreprise pour découvrir la pipeline du studio et développer un outil basé sur l'intelligence artificielle. D'autre part, j'ai travaillé avec le département temps réel pour développer des outils sur Unreal Engine. Mon stage n'impliquait pas un nombre prédéfini de tâches à accomplir dès le début. Au lieu de cela, on m'a proposé une variété de petits projets parmi lesquels choisir, et mes responsabilités ont évolué en fonction de mes progrès. Cette flexibilité m'a permis d'explorer différents types de tâches sans contrainte de délais stricts, ce qui m'a offert une expérience riche et diversifiée.

3.1 Rôle de développeur

Dans un environnement de production complexe tel que celui-ci, où différentes équipes collaborent avec des logiciels sophistiqués en constante évolution, il est impératif de développer des outils facilitant le flux de travail. Ces outils varient, allant de l'intégration de scripts permettant l'importation et l'exportation de certains fichiers standards (comme les fichiers de scène pour Maya) à la création de segments de pipeline complets. Les développeurs cherchent non seulement à optimiser la production en automatisant des tâches répétitives et en réduisant les délais, mais aussi à maintenir et améliorer les outils existants pour s'assurer qu'ils restent efficaces et adaptés aux besoins du studio. Ils travaillent en étroite collaboration avec les artistes pour comprendre leurs besoins spécifiques et ajuster les outils en conséquence.

3.2 Département temps réel

Si les VFX sont aussi puissants aujourd'hui, c'est grâce à de diverses innovations telles que le *compositing* numérique, l'animation 3D, le *ray tracing*, la capture de performances, etc. Actuellement, la *virtual production* représente la dernière itération de cette évolution, sur laquelle le département temps réel se focalise, c'est-à-dire, l'utilisation d'outils dits de temps réel permettant d'obtenir un retour rapide, des effets visuels en direct et *in-camera*. Cette avancée, une fois pleinement intégrée à la pipeline de production, ouvre de nouvelles possibilités pour les équipes de tournage, mais aussi pour les artistes FX, les impliquant dès la pré-production (par exemple, dans la prévisualisation) et même pendant la production. Cette transition réduit la dépendance à la post-production et offre la flexibilité nécessaire pour prendre des décisions cruciales dès le début du processus créatif[35].

Le département temps réel travaille sur une gamme de projets aussi variés qu'intéressants. D'abord en pré-production, où ils créent des prévisualisations ou font du *VR-scouting*. Les prévisualisations consistent en des séquences visuelles rudimentaires qui ont pour but de faciliter la planification de certaines scènes complexes. Généralement, en utilisant des décors simplifiés et des modèles 3D simples. Ces préviz ont pour but de permettre au réalisateur et à son équipe de mieux comprendre la composition des plans, les mouvements de caméra, les séquences d'action et les effets visuels avant de passer à la production réelle. C'est d'ailleurs pour cette raison qu'il n'est pas nécessaire de consacrer du temps au développement visuel détaillé.



(a) Scène de combat complexe, avec en rouge, la trajectoire de la caméra.



(b) Scène de course poursuite.

FIGURE 14 – Exemples de prévisualisations pour la simulation de la trajectoire de la caméra pour le film "Largo Winch : Le Prix de l'Argent" [26]

Une autre tâche de pré-production est le *VR-scouting* qui permet de faire du repérage de décor, en réalité virtuelle. Par exemple, lorsque le décor est inaccessible car il n'est pas encore construit ou qu'il est trop coûteux de déplacer toute l'équipe sur place, le département temps-réel intervient pour modéliser l'environnement ou le scanner en 3D si celui-ci est déjà existant. Cette approche permet de visualiser et d'interagir avec les futurs décors de manière immersive et précise, facilitant ainsi la planification et la préparation du tournage. C'est donc un gain de temps et d'argent.



FIGURE 15 – *VR-scouting* dans un décor virtuel pour le film "8 Rue de l'Humanité" [12].

Dans certains cas, le département temps réel peut aussi participer à la production en faisant ce qu'on appelle du *in-camera VFX*, c'est à dire des effets visuels visibles directement depuis la caméra. Par exemple, en créant des décors qui apparaissent sur *LED wall*¹, permettant aux acteurs de tourner directement devant ces décors virtuels.

1. Un "mur LED" est une grande surface d'affichage composée de panneaux LED, utilisée pour projeter des arrière-plans virtuels en temps réel dans les productions cinématographiques et télévisuelles. Cela permet de créer des décors immersifs sans tournage en extérieur.



(a) Plateau de tournage d'un film avec *LED wall*, sur lequel est diffusé un environnement virtuel dynamique.



(b) Résultat du décor virtuel à travers la caméra.

FIGURE 16 – Exemple de in-camera VFX.

L'ensemble de ces tâches fait partie intégrante de la *virtual production* et c'est l'essence même de la création du département temps réel. Cependant, Unreal Engine n'est pas seulement limité à la virtual production ; il est également capable de gérer des aspects classiques des effets visuels tels que la modélisation, l'animation, le *rigging*, les effets de particules et plus encore, comme le font les logiciels traditionnels d'effets visuels. Sauf que, ce n'est pas courant que ces étapes soient effectuées sur Unreal, et cela s'inscrit donc difficilement dans la pipeline de production du studio à ce jour.

Le département travaille également sur d'autres types de projets, indépendants du cinéma, tels que des expériences interactives en VR ou sur écran tactile, parmi lesquels on peut citer deux projets notables : "Un Centenaire en VR : Bourdelle" et "IBA PTVR".

"Un Centenaire en VR : Bourdelle" est un projet de réalité virtuelle permettant de revivre une exposition consacrée au sculpteur Bourdelle qui a eu lieu en 1928 au Palais des Bozar à Bruxelles[10].

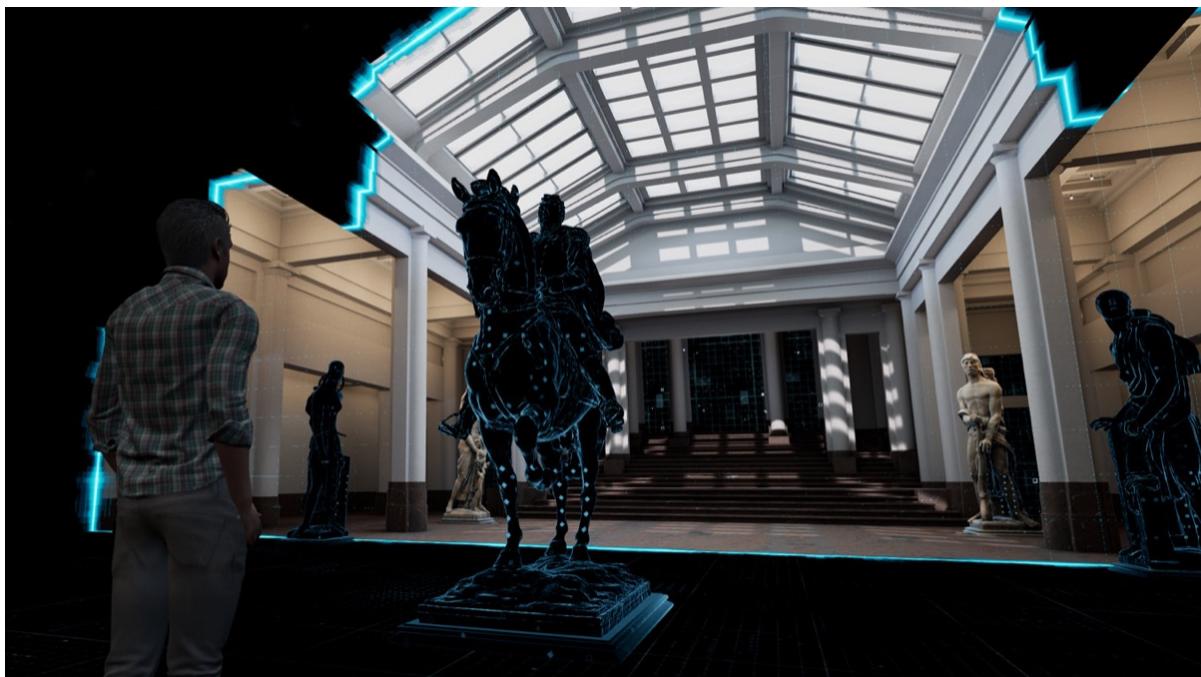


FIGURE 17 – Aperçu de l'expérience VR pour les Bozar : "Un Centenaire en VR : Bourdelle"

"IBA PTVR" est un projet interactif sur écran tactile, permettant de découvrir un centre type et les solutions développées par l'entreprise IBA, spécialisée dans la mise au point de technologies d'accélération

de particules pour des traitements contre le cancer[8].



(a) Plan d'un centre IBA qui peut être visité virtuellement.



(b) Visite de la salle principale du centre avec l'accélérateur de particules.

FIGURE 18 – Projet ”IBA PTVR”.

Ma mission au sein du département temps réel est de développer des solutions d'optimisation qui peuvent être directement appliquées aux projets. Grâce à ma compréhension du langage C++ et du fonctionnement d'un moteur de jeu, je suis capable de créer des outils permettant d'accélérer le processus de création, ou le rendre plus performant.

4 Travaux effectués

4.1 Recherche sur Unreal Engine

4.1.1 Contexte

Au cours de ce stage, le logiciel principal que j'ai été amenée à utiliser n'est autre que Unreal Engine, célèbre moteur de jeu développé et distribué par Epic Games. Ce logiciel permet la création de jeux, d'animations, de simulations ou d'autres expériences interactives.

De manière simplifiée, Unreal propose une interface depuis laquelle il est possible d'éditionner un niveau en y ajoutant une série d'assets, c'est ce qu'on appelle l'éditeur. Un asset fait référence à tout élément numérique utilisé, cela peut inclure des modèles 3D, des textures, des matériaux, des animations, des sons, des effets visuels, des scripts, des niveaux de jeu, etc.

Unreal Engine est un moteur développé en C++. Il est donc possible de coder un projet Unreal exclusivement en C++ ou même de rajouter des fonctionnalités au moteur en allant éditer le code source. De plus, un gros atout d'Unreal est le système de *visual scripting Blueprint* (schéma visuel de script) qui a pour avantage d'être beaucoup plus facile à manipuler pour quelqu'un qui ne maîtrise pas le code, comme les artistes FX. Cependant, il peut arriver que les blueprints soient limités par rapport au code C++. Ma première tâche au sein du département temps réel a donc été d'étudier l'utilisation du C++ et des blueprints au sein d'Unreal afin d'évaluer quelle méthode de script possède quels avantages, et dans quels cas l'utiliser.

4.1.2 Résultats

Après avoir étudié le fonctionnement d'Unreal, et recherché les fonctionnalités proposées par le moteur (ressources en Annexe 6), voici les conclusions que j'ai pu tirer.

Le C++ et les blueprints doivent être combinés pour optimiser la production. De manière simplifiée, on peut dire que le C++ est utilisé pour les tâches de bas niveau, comme la programmation du *gameplay*, tandis que les Blueprints sont utilisés pour des tâches de haut niveau, comme les visuels et les animations. De plus, le C++ permet d'optimiser les opérations coûteuses en ressources. Au-delà de cela, voici les avantages et inconvénients des deux méthodes :

	Blueprints (BP)	C++
Visualisation des assets et effets visuels	Les Blueprints étant des assets, lorsqu'ils référencent un asset, cela crée simplement une dépendance asset-asset (bien gérée par Unreal).	Fonctionne à l'aveugle. Pour référencer un asset, il doit créer une dépendance entre l'asset et le module de jeu compilé. Si l'asset change, le code source doit être mis à jour manuellement et recompilé.
Gestion des comportements scriptés	Gère bien les comportements scriptés. Les <i>Event Graphs</i> sont avantageux pour les <i>timers</i> , <i>timelines</i> (animations dans le temps), séquenceurs, etc. car ils sont orientés programmation par événements.	On peut utiliser des <i>timers</i> et des <i>callbacks</i> , mais c'est plus compliqué à gérer et à itérer.
Test et itération efficaces	L'expérience de création se fait en temps réel sans besoin de processus de <i>build offline</i> . La compilation des Blueprints se fait souvent localement dans l'éditeur, permettant de voir le <i>Event Graph</i> pendant l'exécution pour déboguer.	Les tests nécessitent de recompiler tout le projet à chaque fois. Cela peut être long et il est plus difficile de déboguer car on ne voit pas facilement d'où vient une erreur.
Accessibilité	Accessible aux non-codeurs.	Nécessite des compétences en programmation.
Performance	Passe par plus d'instructions CPU pour le même résultat. Sur-coût insignifiant sauf si on utilise beaucoup d'acteurs qui font de gros calculs à chaque frame.	Performances maximisées à l'exécution. Le code est compilé en instructions machine pour le CPU, contrairement aux BP qui passent par une machine virtuelle de script.
Fusion de code	Grande difficulté pour fusionner les modifications.	La fusion et les différences (<i>diffing</i>) sont plus faciles car le code est du texte. On peut facilement voir les différences entre deux versions.
Documentation	Les listes de variables et fonctions sont affichées dans l'éditeur, réduisant le besoin de documentation supplémentaire.	Le code fondamental doit être en C++. Tout type d'objet fondamental nécessaire doit être codé en C++, bon pour tout ce qui doit être étendu, maintenu ou réutilisé. C++ est un langage fiable, non ambigu, standard.
Fonctionnalités du moteur	–	Permet d'accéder à un maximum de fonctionnalités du moteur.
Bibliothèques tierces	–	En C, C++ mais aussi Python (peut par exemple renommer des assets automatiquement).

À cela s'ajoute le Python, langage non compilé largement répandu. C'est d'ailleurs pour son habileté à être manié qu'Unreal permet l'ajout de plugins en Python, permettant ainsi aux personnes qui ont des bases en codage, d'ajouter des fonctionnalités codées avec un langage plus simple que C++, comme Python. Un des cas d'utilisation fréquent de ce langage est la communication entre Unreal et d'autres logiciels tiers, étant donné sa compatibilité répandue avec de nombreuses applications externes.

4.2 Création d'un outil de fusion

4.2.1 But

Comme mentionné dans la section 3.2, le département temps réel a réalisé plusieurs projets de réalité augmentée et virtuelle. Les casques VR utilisés (à savoir, les casques Meta Quest 2 et 3 [14]) possèdent un processeur limité par rapport à celui d'un PC. Puisqu'ils doivent prendre en compte les performances graphiques pour maintenir des fréquences d'images élevées et une latence minimale, cela a pour effet de limiter le nombre de *drawcalls* possibles. Les *drawcalls* sont des instructions envoyées par le processeur au GPU pour afficher des objets à l'écran. Pour les scènes complexes d'un projet (c'est-à-dire avec de nombreux acteurs²), il est nécessaire d'optimiser les performances. Cela implique d'ajuster la qualité des textures et des modèles 3D ou encore de pratiquer le *frustrum culling*³, mais aussi, de limiter le nombre de *drawcalls*, qui dépend directement du nombre d'acteurs affichés à l'écran (1 acteur = 1 *drawcall*). Pour contourner ce problème, Unreal a mis au point un outil de fusion "Actor Merge tool"^[19] qui permet de combiner plusieurs Static Mesh Actors⁴ (SMA), afin de n'afficher que le SMA résultant (que j'appelle ici *cluster* (regroupement)) et donc limiter le nombre de *drawcalls*. Cependant, certains paramètres doivent être appliqués à la main après le processus de fusion, comme :

- cacher du jeu les acteurs qui forment le cluster,
- les définir comme mobiles,
- désactiver leur ombre,
- désactiver leur système de collision,
- activer l'ombre des clusters,
- désactiver leur système de collision,
- et enfin, les définir comme statiques.

Les étapes ci-dessus permettent, en somme, de mettre de côté les acteurs originaux constituant le cluster, sans les supprimer pour autant. En plus de cela, il arrive qu'il faille modifier un cluster existant ou le supprimer en cas d'erreur, nécessitant de revenir en arrière sur les paramètres modifiés. Ces étapes sont redondantes, lentes, et à la source de beaucoup d'erreurs. Pour régler ce problème pour les projets futurs, ma tâche consiste à créer un outils de fusion avancé (que j'ai appelé *Merge Tool*), en blueprint sur Unreal Engine. Cet outil doit permettre de gérer toutes ces étapes de manière intuitive pour l'utilisateur, tout en couvrant un maximum de cas possibles et en réduisant les risques d'erreurs.

4.2.2 Description

L'outil souhaité consiste en une série de fonctions accessibles à l'utilisateur depuis l'éditeur lui permettant de sélectionner une liste d'acteurs et de les fusionner en cliquant sur un bouton. En les fusionnant, les étapes citées précédemment (qui devaient être faites à la main initialement) sont automatiquement appliquées. En plus de cela, l'outil permet de visualiser les clusters et les acteurs qui les constituent dans le panneau de détail de l'outil, de définir certains paramètres avancés (comme le chemin dans lequel l'asset du cluster va être créé), de modifier un cluster, d'en supprimer un, ou encore de visualiser les clusters directement dans la scène à l'aide d'une vue spécifique qui donne une couleur à chaque cluster.

2. Dans Unreal Engine, un acteur est une unité de base du *gameplay*, représentée par un objet placé dans une scène 3D.

3. Technique d'optimisation graphique qui consiste à ne rendre que les objets visibles dans le champ de vision de la caméra.

4. Dans Unreal Engine, un Static Mesh Actor est un élément de la scène qui représente un objet statique, tel qu'un bâtiment, un rocher, un arbre, etc. Il est utilisé pour afficher des objets qui ne changent pas ou qui ne sont pas animés pendant le jeu.

4.2.3 Implémentation

L'implémentation de cet outil s'est faite de manière itérative : sur base de la description initiale du produit, des améliorations ont été apportées au fur et à mesure. On peut néanmoins découper le processus de création de cet outil en trois phases principales.

Premièrement, une phase d'exploration durant laquelle j'ai commencé par me familiariser avec les blueprints : comment fonctionne le graphe de scripting, comment créer une variable/fonction, quel type de blueprint dois-je utiliser,... Ensuite, j'ai créé un organigramme (*flowchart*) (voir Annexe 6) permettant de schématiser le fonctionnement de mes blueprints et assets et la structure générale de mon outil.

Ensuite vient la phase de scripting pour une première version de l'outil, que j'appelle la version Beta. La version Beta consiste en un premier produit fonctionnel mais qui nécessite encore des améliorations. Cette version de l'outil permettait uniquement de fusionner des Static Mesh Actors, et son interface restait assez limitée et peu intuitive. Une série de tests ont été effectués par les membres du département temps réel, ce qui a permis de lister les améliorations et de passer à la troisième phase : la phase de scripting pour la version finale.

Une des améliorations majeure pour la version finale a été d'inclure la possibilité de fusionner des acteurs qui contiennent des Static Mesh Components (pouvant être ensuite convertis en Static Mesh Actors) mais qui ne sont pas eux-mêmes des Static Mesh Actors. Cela a eu pour effet de modifier considérablement la structure de mon graphe de noeuds. Une autre amélioration était de rendre l'outil plus intuitif pour l'utilisateur, en modifiant notamment la structure des clusters de façon à ce qu'un aperçu puisse être disponible dans le panneau de détails de l'outil (voir Figure 19).

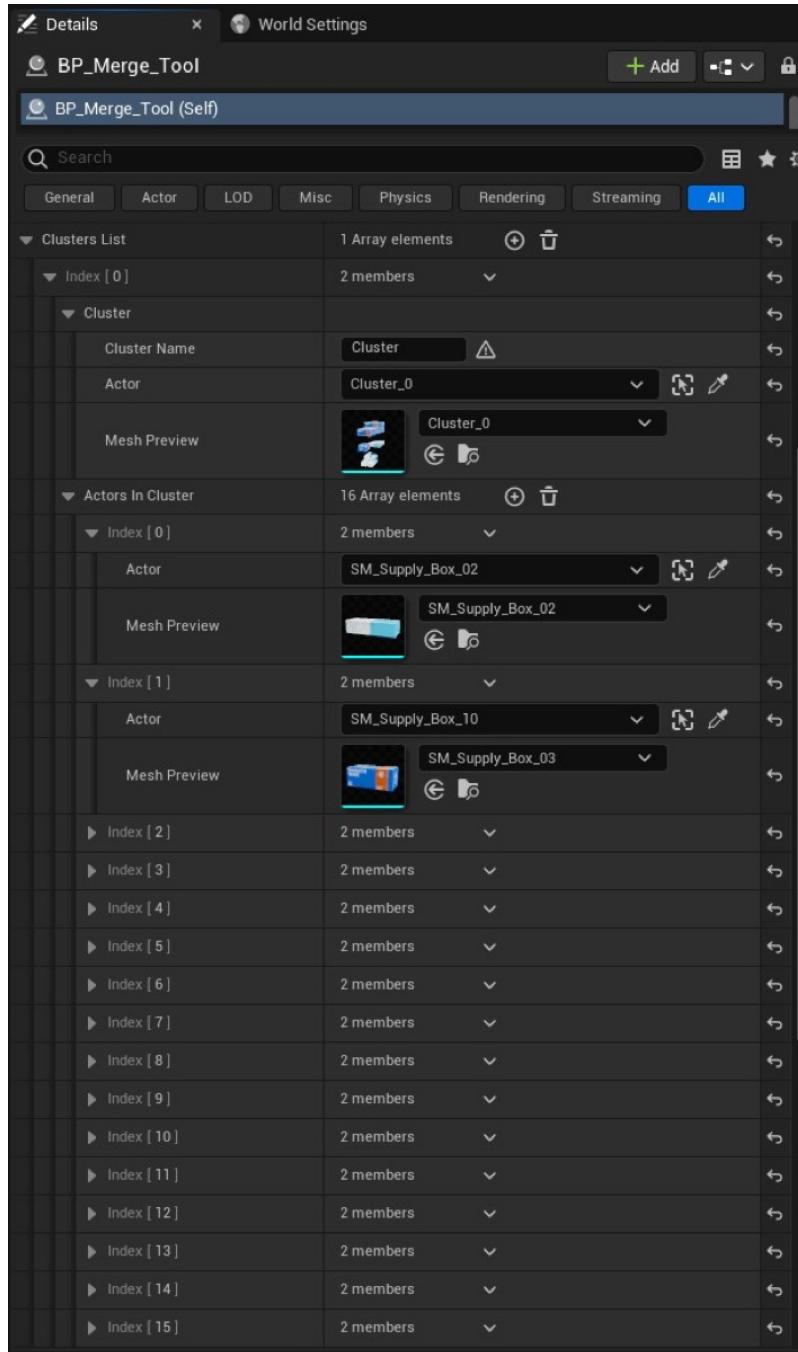


FIGURE 19 – Panneau de détails du *Merge Tool* montrant un aperçu d'un cluster et des 15 éléments qui le constituent.

De manière générale, le défi majeur du développement de cet outil réside dans la communication entre les différents assets et blueprints permettant de faire fonctionner l'outil, schématisée en Figure 20.

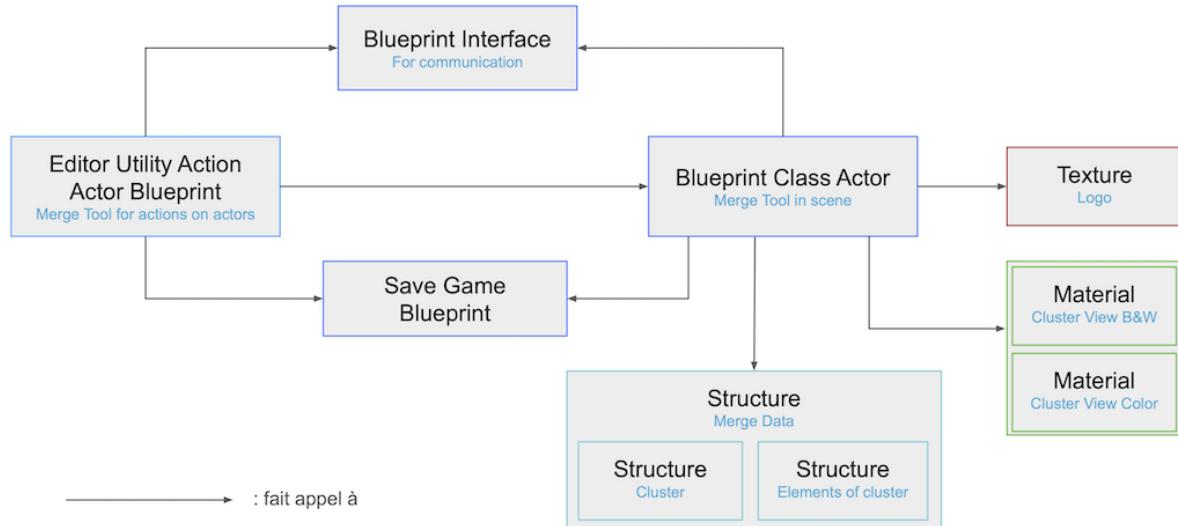
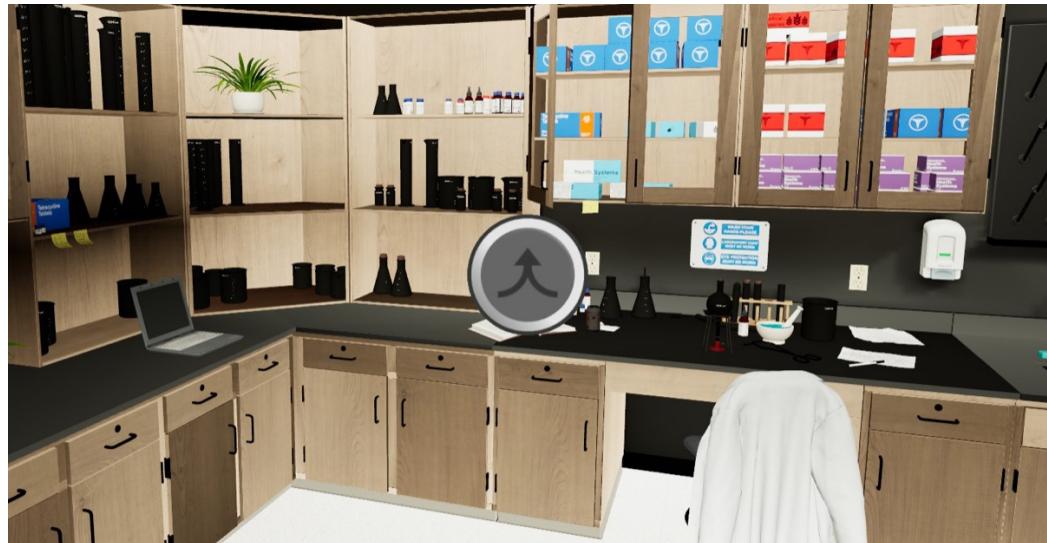


FIGURE 20 – Schéma de la communication entre les différents assets pour la création de l'outil de fusion.

Puisque l'outil doit pouvoir agir sur les acteurs de la scène (i.e. en les fusionnant), j'ai conclu qu'il me fallait utiliser un "Editor Utility Action Actor Blueprint" (que je vais appeler Editor Blueprint pour simplifier), qui est un type de blueprint qui permet de créer des fonctions prévues pour être appelées depuis l'éditeur (et jamais pendant l'exécution) grâce aux "Scripted Actor Actions" disponibles depuis un clic droit dans la scène. En plus de cela, comme on veut pouvoir modifier les clusters, il est nécessaire de visualiser les acteurs qui les composent depuis l'éditeur. Cela est rendu possible par l'utilisation d'un "Blueprint Class Actor" (que je vais appeler Actor Blueprint pour simplifier) qui va pouvoir être placé dans la scène et récolter toutes les informations sur les clusters créés.



(a) Logo de l'outil de fusion.



(b) Aperçu du "Blueprint Class Actor" pour l'outil de fusion, placé dans la scène et pouvant être repéré par son logo.

FIGURE 21

Il y a donc deux manières de créer et d'éditer des clusters. D'une part depuis le panneau de détails comme on le voit sur la Figure 22 dans le cadre jaune. D'autre part depuis un clic droit dans la scène comme on peut le voir sur la Figure [fig:mergetool'lab] dans le cadre rose.

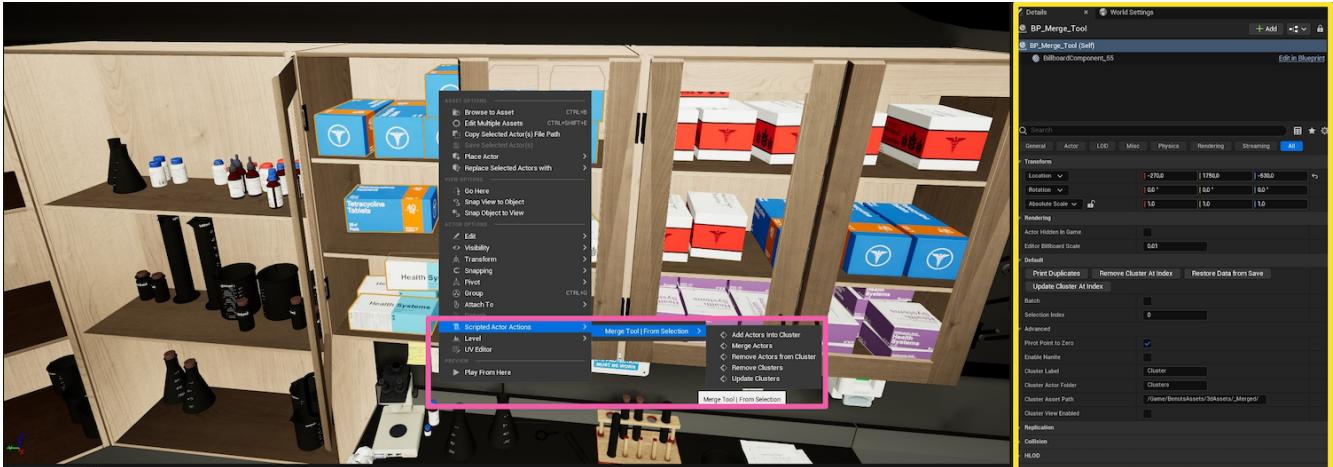


FIGURE 22 – Aperçu de l'utilisation de l'outil de fusion pour une scène contenant un grand nombre d'acteurs. Dans le cadre rose : les fonctions disponibles de l'outil s'appliquant aux acteurs sélectionnés. Dans le cadre jaune : le panneau de détails de l'outil.

Sur la figure ci-dessus, on peut voir un aperçu d'une scène 3D d'un laboratoire contenant beaucoup d'acteurs. Nombreux d'entre eux peuvent être fusionnés en clusters puisqu'ils sont proches dans l'environnement et seront donc, en principe, toujours visibles à l'écran en même temps. Prenons l'exemple des boîtes de médicaments dans l'armoire. L'utilisateur peut les fusionner en les sélectionnant dans la scène puis en faisant un clic droit et en choisissant la nouvelle fonction "Merge Actors". On continue ensuite pour tous les autres groupes qui peuvent être fusionnés. C'est à l'utilisateur de définir quels acteurs sont fusionnés, en fonction de leur proximité.

Comme expliqué précédemment, la communication entre l'Editor Blueprint et le Blueprint Actor est une partie essentielle dans l'implémentation de cet outil. Sur la Figure ??, on comprend que cette communication est directe de l'Editor Blueprint au Blueprint Actor, alors que dans l'autre sens, elle est rendue possible grâce à l'utilisation d'une interface. Ce schéma reprend aussi les autres assets utilisés pour le déploiement de cet outil tels que : trois Structures permettant d'encoder les données des clusters de manière organisée, un blueprint de sauvegarde, la texture du logo et enfin deux matériaux pour visualiser les clusters en couleur et le reste de la scène en noir et blanc (voir Figure 23).

4.2.4 Résultats

Cet outil atteint son objectif en permettant de réduire le nombre de *drawcalls* pour une scène 3D interactive complexe. Par exemple, pour la vue depuis la caméra en Figure ??, le nombre de *drawcalls* est de 239. Alors qu'avant la fusion des groupes d'acteurs proches, ce nombre s'élevait à 499. Le nombre de *drawcalls* a donc diminué de plus d'une moitié.

De plus, il est possible de mettre en évidence les clusters créés à l'aide du booléen "Cluster View Enabled" visible sur le panneau de détails du *Merge Tool* (voir Figure 22). Cette vue ajoute un matériel de couleur aux clusters (basé sur leur indice unique) et rend le reste de la scène en noir et blanc. En se basant sur la sélection de la Figure 22, voici le résultat obtenu :



FIGURE 23 – Mise en évidence d'un cluster (en violet) à l'aide du "Cluster View".

Une fois l'outil terminé, j'ai rédigé une documentation décrivant son fonctionnement afin que n'importe quel utilisateur Unreal puisse en faire usage. Un aperçu de cette documentation peut être trouvé en Annexe 6.

4.3 Création d'un outil de dispersion

4.3.1 But

Dans le processus de création d'une scène 3D, il est fréquent de vouloir disposer des objets similaires (ou presque similaires) de manière répétitive et rapprochée sur une surface. Cette tâche peut souvent s'avérer longue et redondante, surtout si la surface est grande. Par exemple, pour le décor du film *Night in Paradise*[27], le département temps réel a été amené à modéliser un long tunnel dans lequel des lampes devaient être disposées tout le long, et ce, avec des paramètres ajustés (comme la couleur ou l'intensité de la lumière) pour un rendu plus réaliste (voir Figure 24). Pour remédier à ce problème, on souhaite développer en blueprint un outil de dispersion (que j'ai appelé *Scatter Tool*) permettant de distribuer des objets de manière régulière.

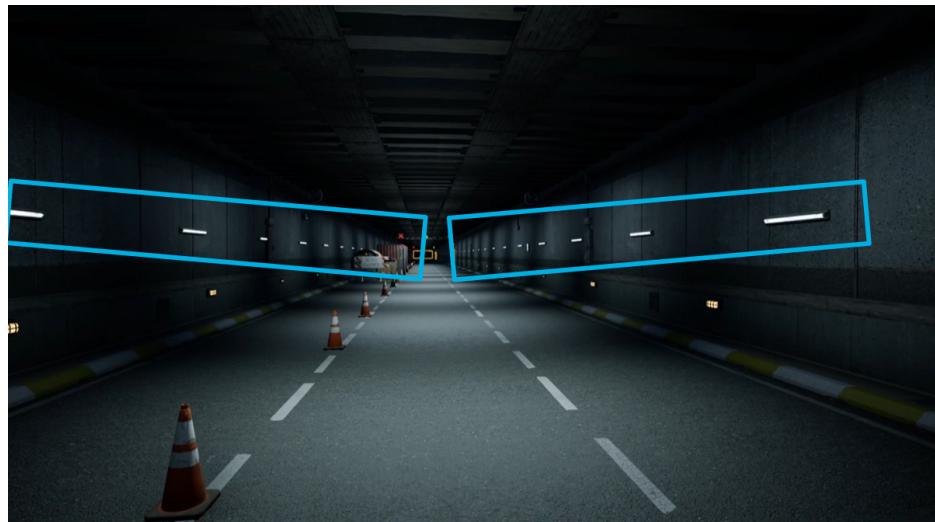


FIGURE 24 – Extrait du film *Night in Paradise*[27] dans lequel le décor a été modélisé sur Unreal et les lampes disposés le long du tunnel à l'aide d'un outil de dispersion.

4.3.2 Description

Le but de cet outil de dispersion, est de disposer des objets dans la scène de manière régulière selon trois cas : le long d'une courbe spline, sur la circonférence d'un cercle ou encore dans une grille 3D. Une courbe spline est une courbe lisse définie par une série de points de contrôle, utilisée pour modéliser des trajectoires complexes.

Dans le cas de la spline, on souhaite disperser les objets selon un paramètre unique :

- l'espacement entre les objets

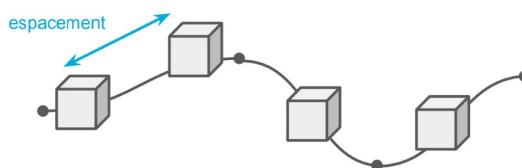


FIGURE 25 – Schéma de la dispersion d'objets le long d'une spline selon l'espacement.

Pour le cercle, voici les trois paramètres de contrôle qui peuvent être édités :

- le rayon du cercle

- le nombre d'objets
- le décalage (ou *offset*)

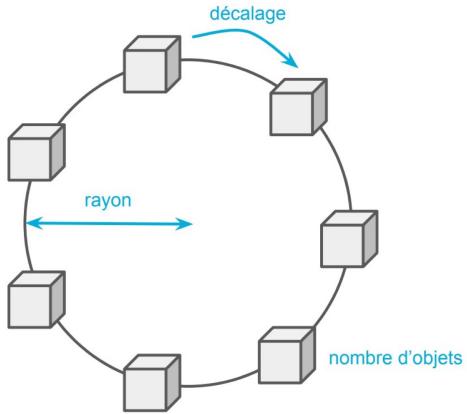


FIGURE 26 – Schéma de la dispersion circulaire d'objets selon le rayon du cercle, le nombre d'objets et le décalage.

Et enfin, pour la grille 3D, les deux paramètres éditables sont les suivants :

- le nombre d'objets (ex x, y et z)
- l'espace entre les objets (en x, y et z)

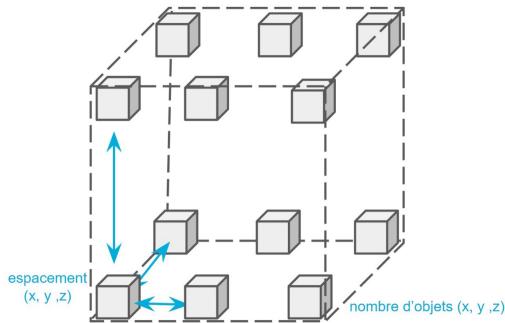


FIGURE 27 – Schéma de la dispersion d'objets en grille 3D selon l'espacement entre les objets (en x, y et z) et le nombre d'objets (en x, y et z).

À cela s'ajoutent les paramètres communs aux trois cas : l'échelle et la rotation des objets (en x, y et z).

En outre, il est possible d'introduire de la variation selon un spectre spécifié par l'utilisateur pour : la position, la rotation et l'échelle. C'est ce que j'appelle la dispersion irrégulière. Aussi, plusieurs objets peuvent être utilisés pour une même dispersion, avec chacun une probabilité d'apparition, ce que j'appelle la dispersion avec variété. Ces fonctionnalités complémentaires sont susceptibles d'offrir un résultat plus naturel selon la nature du projet.

4.3.3 Implémentation

Après une phase de recherche et d'exploration, j'ai pu commencer à développer l'outil. Le blueprint principal utilisé pour cet outil est un "Actor Blueprint", qui peut être placé directement dans la scène, et son fonctionnement lancé depuis le script de construction. Ce script a pour avantage de se lancer dès

qu'une modification est appliquée au blueprint (comme par exemple si un paramètre est modifié), ce qui permet d'agir en fonction.

Le script de construction agit selon six cas différents, définis par la combinaison de deux boîtes combinées (ou *checkbox*) éditables par l'utilisateur : l'une pour le type de dispersion (i.e. spline, cercle ou grille) et l'autre pour le type d'objet. L'objet peut être soit un Static Mesh (SM) soit un Actor Reference. Un SM est l'équivalent d'un Static Mesh Actor sauf qu'il ne se situe pas dans la scène, ce n'est donc pas un acteur. Un Actor Reference est une variable qui stocke la référence à un acteur.

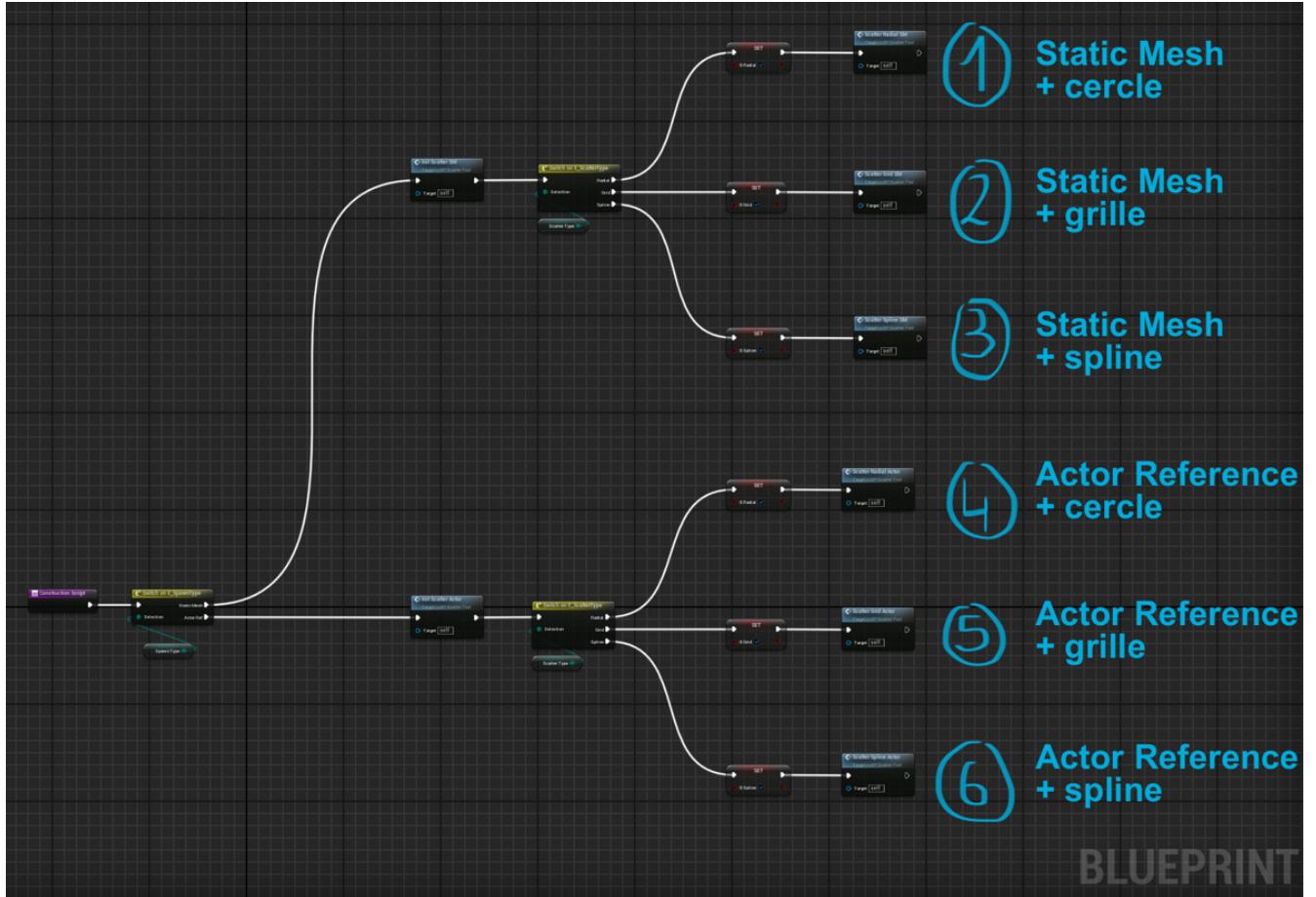


FIGURE 28 – Aperçu du script de construction de l'outil de dispersion, agissant selon six cas.

En fonction du cas sélectionné par l'utilisateur, certains paramètres sont visibles, d'autres cachés. Cette fonctionnalité est rendue possible grâce à l'utilisation d'un plugin : MDMetaDataEditor[15].

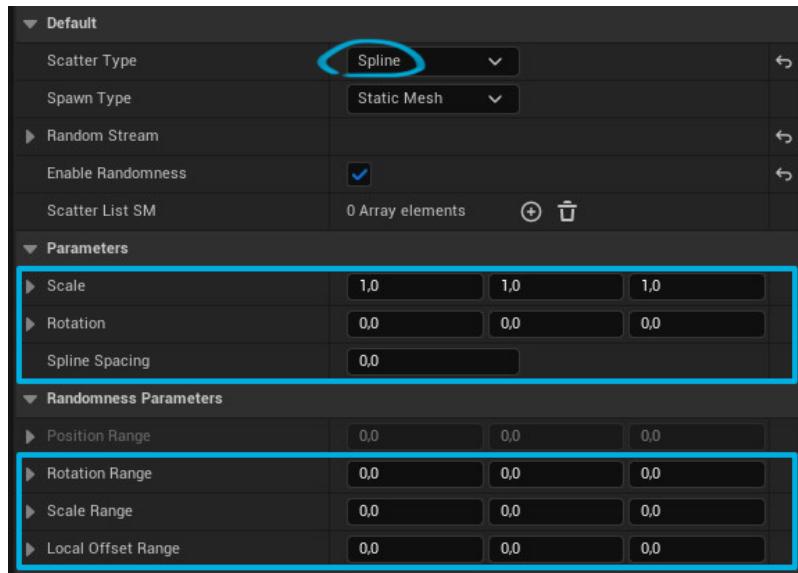


FIGURE 29 – Paramètres disponibles pour une dispersion sur spline.

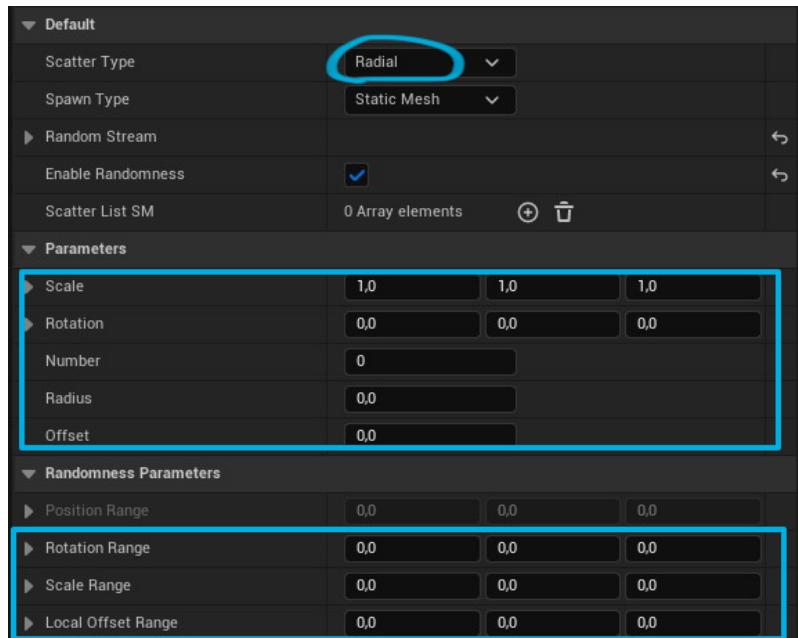


FIGURE 30 – Paramètres disponibles pour une dispersion circulaire.

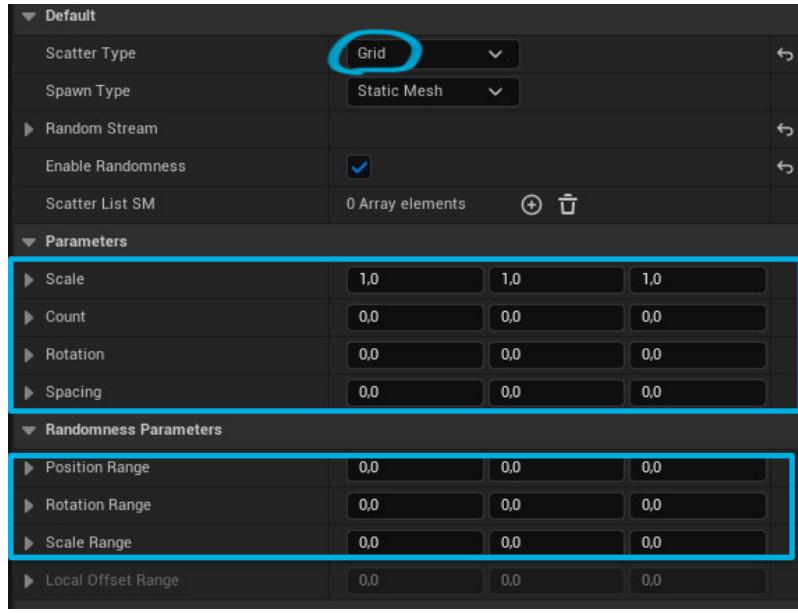


FIGURE 31 – Paramètres disponibles pour une dispersion en grille.

4.3.4 Résultats

Une fois fini, l'outil de dispersion permet de distribuer un relativement grand nombre d'objets de manière régulière ou irrégulière, de manière efficace mais tout en gardant le contrôle grâce aux paramètres disponibles.

J'ai pris comme exemple une carte extraite du package "Greenwood Fantasy Village" [28], disponible sur le Marketplace d'Unreal, dans laquelle j'ai disposé des légumes, des plantes et des maisons à l'aide de mon outil. Sur la figure ci-dessous, on peut observer différentes dispersions : circulaires, en grille, sur spline, régulières/irrégulières, avec/sans variété.

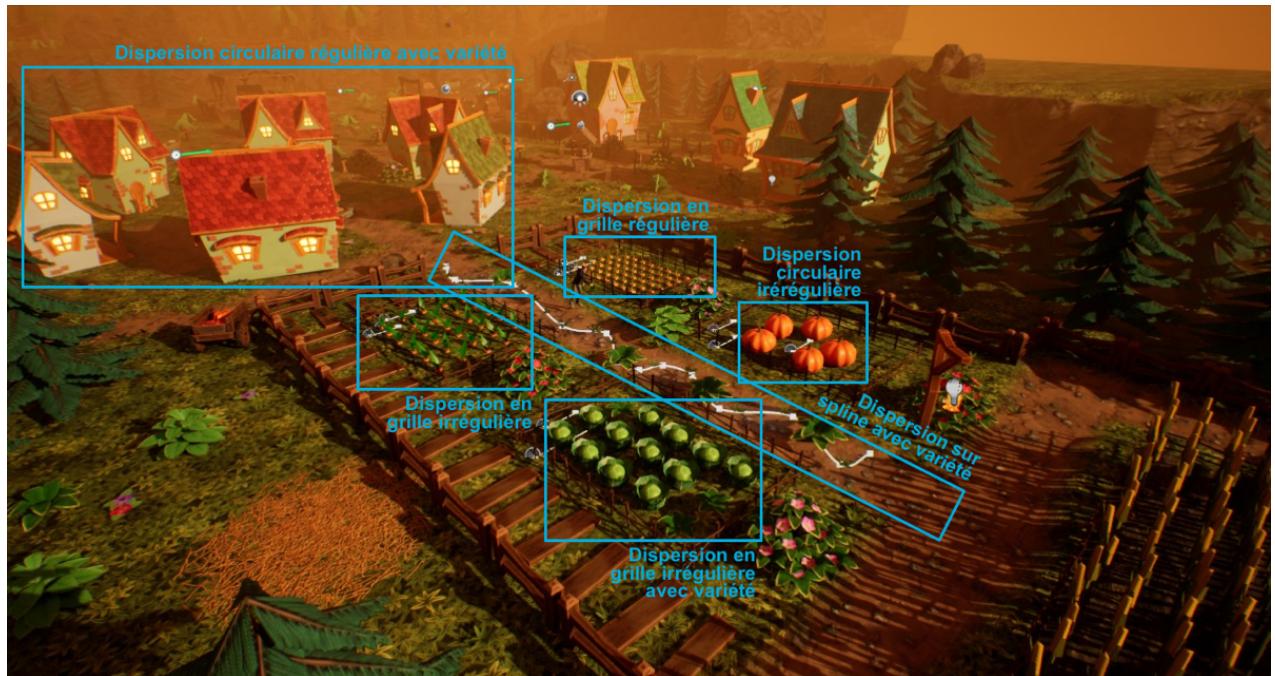
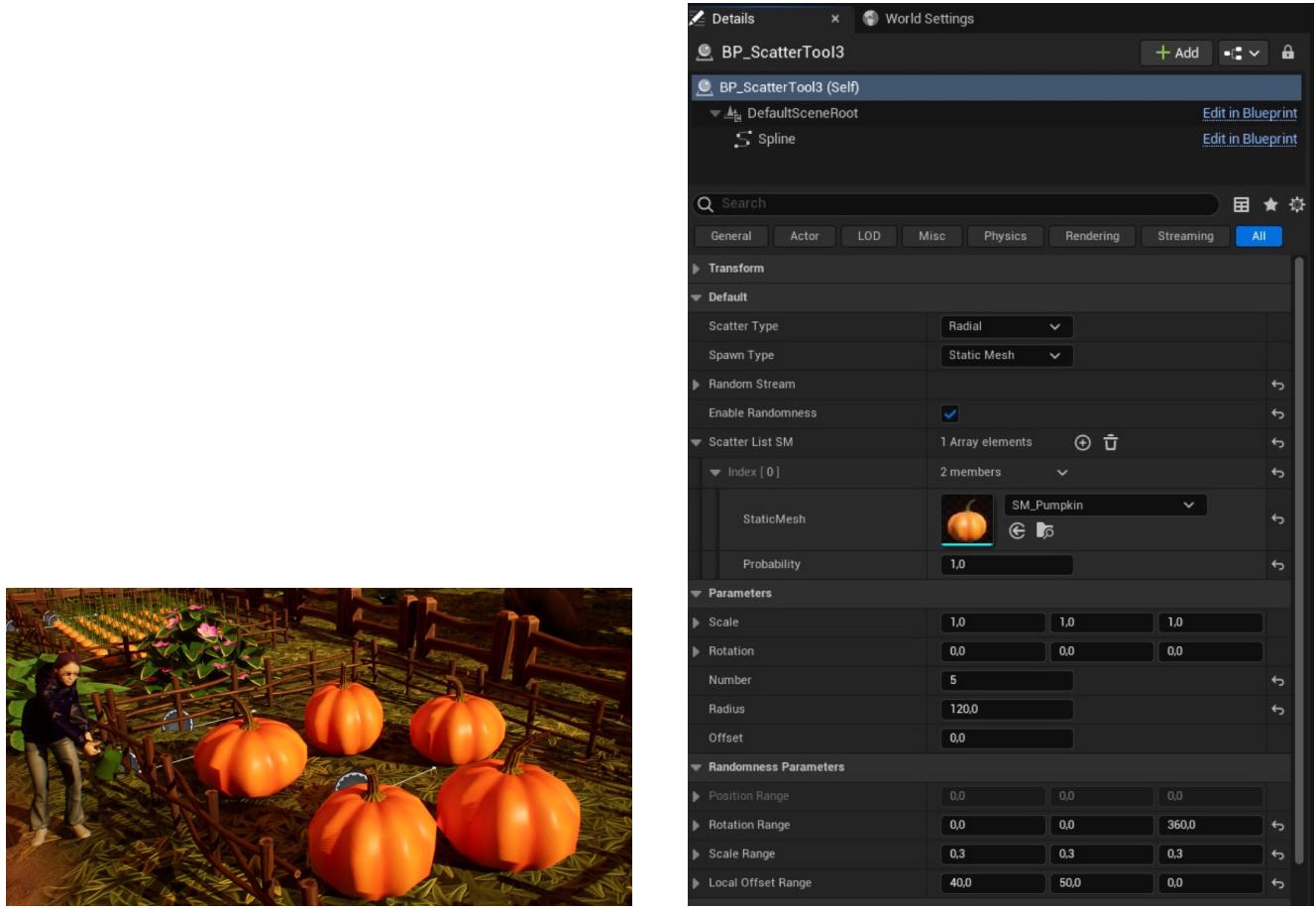


FIGURE 32 – Exemples de différentes dispersions dans le décor "Greenwood Fantasy Village" [28].

On peut zoomer sur la dispersion circulaire des citrouilles afin de voir plus clair comment les paramètres sont spécifiés :



(a) Dispersion circulaire irrégulière d'assets de citrouilles.

(b) Panneau de détails de l'outil utilisé pour la dispersion circulaire des citrouilles.

FIGURE 33 – Exemple d'une dispersion et de ses paramètres.

Sur la Figure 33 (b), on peut voir que le booléen "Enable Randomness" a été coché, donnant accès aux "Randomness Parameters" qui permettent de définir les bornes des valeurs que l'on veut aléatoires. Dans cet exemple, on peut voir par exemple que les citrouilles peuvent avoir une rotation autour de l'axe z qui varie entre 0 et 360°, une échelle qui peut varier entre 1.0 et 1.3, ainsi qu'un décalage local (*local offset*) de 40 unités perpendiculairement à la tangente du cercle et de 50 unités le long de la circonference du cercle.

4.4 Creation d'un outil de detourage

4.4.1 But

Dans les effets visuels, les compositeurs, c'est-a-dire les artistes qui font du *compositing* (voir section 2.2), ont souvent besoin de faire de la rotoscopie faciale, c'est-a-dire detourer un visage dans une sequence d'images. Cela peut tre necessaire pour plusieurs raisons, comme pour ajouter du maquillage numerique (*digital makeup*), remplacer les visages des cascadeurs par ceux des acteurs principaux dans des scenes d'action, ou encore appliquer des effets de vieillissement ou de rajeunissement. Detourer un visage sur une sequence d'images est une tâche extrêmement chronophage car, contrairement  une image fixe, une sequence peut contenir des centaines d'images, chacune necessitant une attention individuelle pour assurer un detourage propre et consistant tout au long des frames.

Actuellement, plusieurs technologies utilisees dans le studio permettent de realiser du detourage intelligent, comme Nuke et Adobe Photoshop. Ces outils permettent non seulement de sélectionner une zone de l'image efficacement, mais aussi, de reporter et retravailler le masque detecte sur les frames suivantes. Cependant, ces outils de detourage ne sont pas specifiques aux visages.

Il existe toutefois une serie d'outils comme KeenTools^[22] permettant de construire et traquer des maillages sur une sequence de frames. C'est le cas du "FaceBuilder for Nuke" qui offre la possibilite de construire le maillage 3D d'un visage, en deformant et repositionnant un maillage initial. On peut ensuite utiliser l'outil "FaceTracker for Nuke" qui permet de traquer ce visage tout au long des frames, donc de deplacer ce maillage de maniere automatique. Ces outils, disponibles sous forme de plugins payants, necessitent neanmoins toujours une supervision humaine, en particulier lorsqu'il s'agit de placer le maillage sur la premiere frame  la main.

La conception d'un outil de detection faciale (que j'appelle "*Face Tracker*") vise donc  faciliter la detection de ces maillages 3D tout au long des frames, mais aussi de creer des masques de rotoscopie en 2D. L'avantage de cet outil reside donc dans sa capacite  effectuer rapidement et efficacement la majeure partie du travail, tout en permettant aux artistes de contrôler et de modifier les resultats obtenus par apres, le tout  moindre cot. Son utilisation permettrait, par exemple, de remplacer plus facilement des visages dans une sequence d'images.

4.4.2 Description

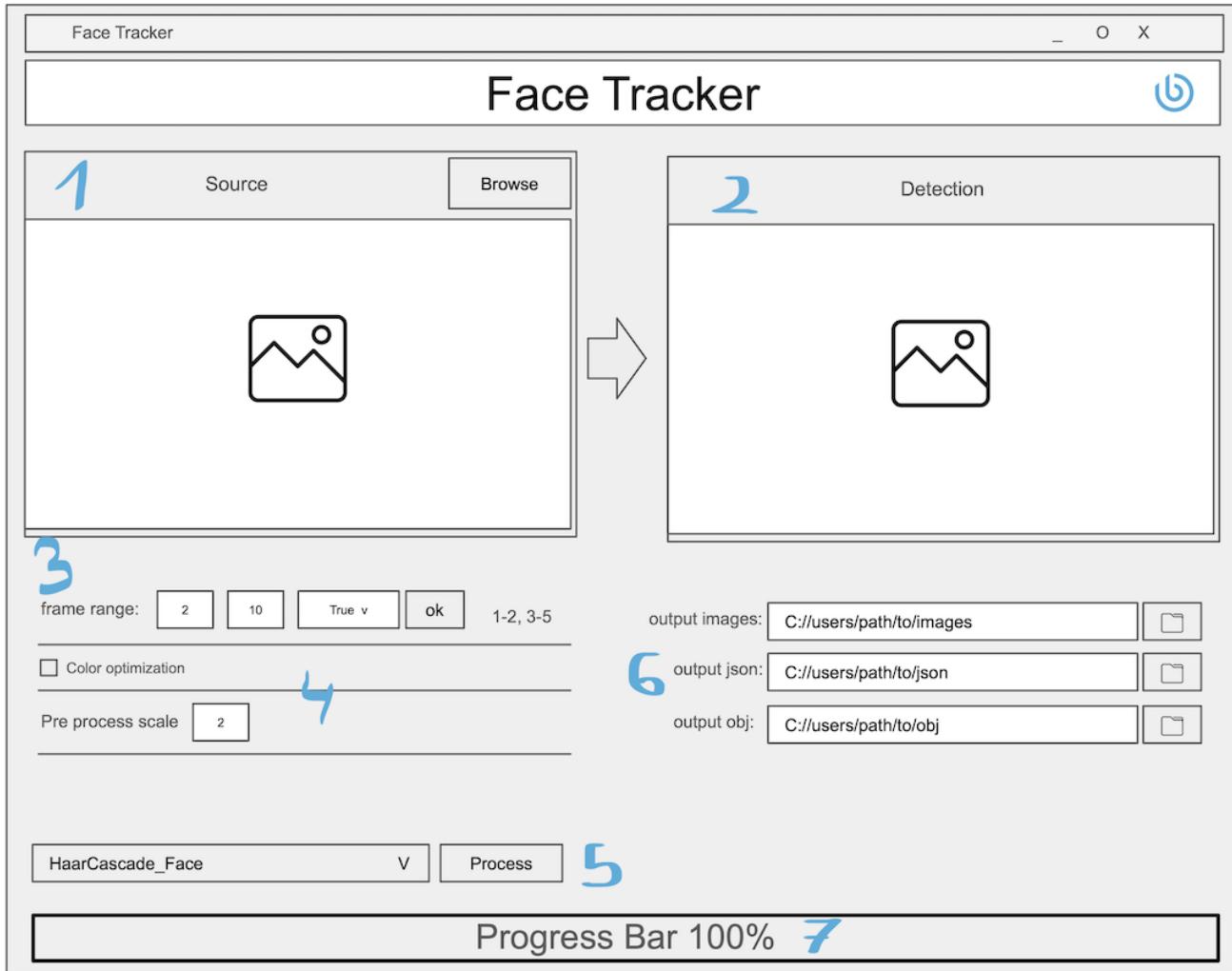


FIGURE 34 – Schéma de l'outil souhaité.

L'outil de détection faciale que j'ai développé consiste en une interface utilisateur (voir Figure 34) à travers laquelle l'utilisateur peut sélectionner une séquence d'images (Figure 34 en 1). Le *frame range* est automatiquement détecté, mais peut être modifié via des boîtes à nombres (*spin boxes*) (Figure 34 en 3), permettant de récupérer les images de la séquence qui seront ensuite affichées de manière animée (Figure 34 en 2).

Sur la Figure 34 en 5, l'utilisateur peut choisir un modèle de détection faciale parmi la liste suivante :

- **Haarcascade (HC)**[29] pour la détection du cadre du visage ;
- **Mediapipe Face (MPF)**[3] pour la détection du cadre du visage ;
- ou **Mediapipe Landmarks (MPL)**[4] pour la détection d'un maillage précis du visage.

Après sélection, le bouton "Process" (Figure 34 en 5) permet de lancer la détection.

Sur la Figure 34 en 4, l'utilisateur peut cocher l'option "Color optmization" afin d'optimiser l'histogramme des couleurs de l'image en pré-traitement. Il peut également spécifier une échelle à laquelle l'image sera traitée. Ces deux paramètres sont susceptibles d'améliorer les résultats et la rapidité de calcul. La partie droite de l'interface montre les images de sorties avec les détections par dessus (Figure 34 en 2).

Sur la Figure 34 en 6 sont spécifiés les chemins des fichiers des données de sortie :

- les images générées avec détection ;
- les coordonnées en pixels des détections (sous forme de fichier JSON)
- les maillages 3D détectés (un par frame, en format Wavefront).

Enfin, sur la Figure 34 en 7, une barre de progression permet de voir où en est le traitement des images.

4.4.3 Implémentation

Pour réaliser cette interface, j'ai utilisé PySide2[25] (aussi connu sous le nom de "Qt for Python"), qui est un *binding* (ou liaison) qui permet de lier le langage Python avec la bibliothèque Qt permettant de créer des interfaces graphiques.

Pour le modèle de détection faciale, j'ai utilisé deux approches :

- **Haarcascade** : Un modèle pré-entraîné fourni par OpenCV, capable de détecter des caractéristiques clés de l'image telles que des contours, permettant ainsi de détecter un visage.
- **Mediapipe** : Un framework open source développé par Google, qui propose de nombreuses solutions de détection, dont la détection rapide et précise de visages dans les images et vidéos, ainsi que la génération d'un maillage 3D détaillé des visages. Mediapipe offre un bon compromis entre rapidité de calcul et précision des résultats[23].

Certains pré-traitements peuvent également améliorer les résultats, tels que l'optimisation des couleurs pour la précision ou la réduction du nombre de pixels pour augmenter la rapidité de calcul.

4.4.4 Résultats

Sur la figure ci-dessous, on peut voir l'interface utilisateur de l'outil de détection faciale, qui suit précisément la description précédemment donnée.

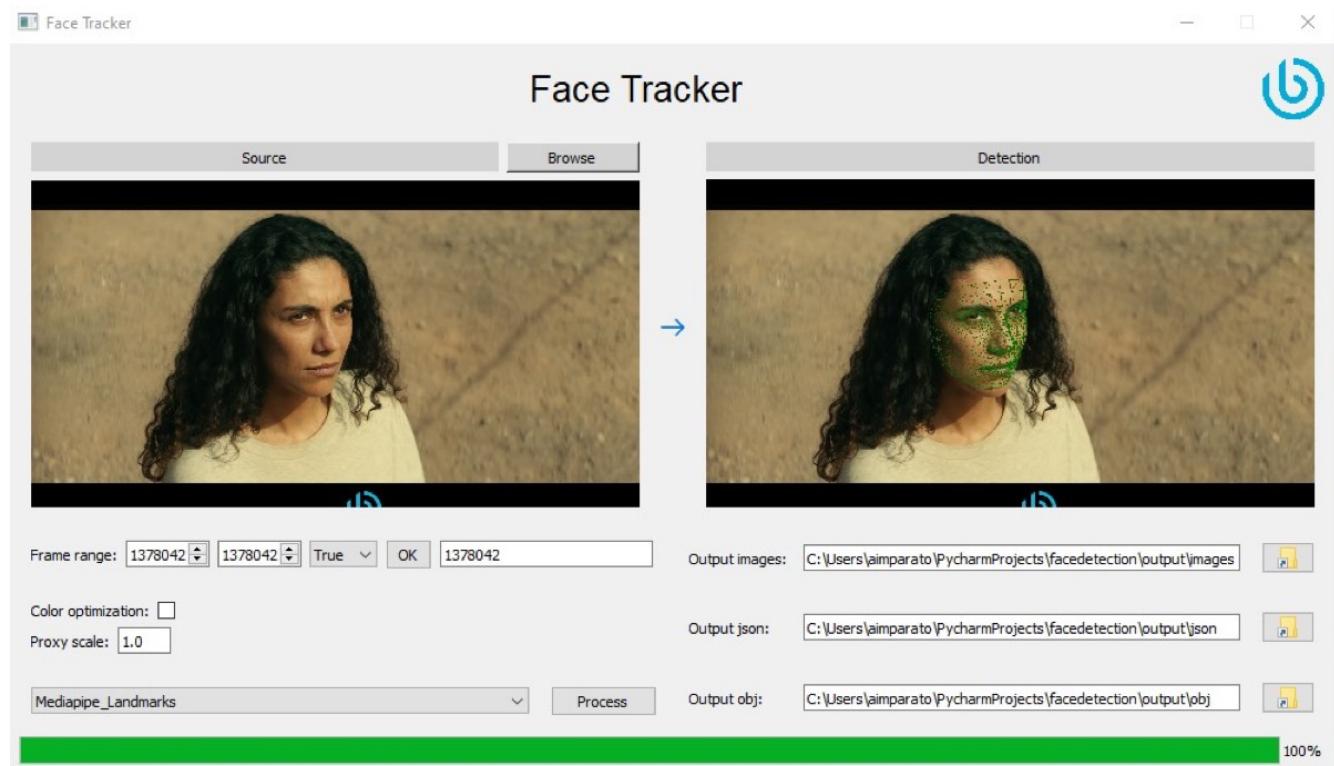


FIGURE 35 – Interface utilisateur de l'outil de détection faciale.

Voici les résultats obtenus pour les trois modèles sur une image extraite du film "Le Salaire de la Peur" [24] :



FIGURE 36 – Détection du visage avec Haarcascade (HC). Le cadre bleu affiche le résultat.



FIGURE 37 – Détection du visage avec Mediapipe Face (MPF). Le cadre rouge affiche le résultat.



FIGURE 38 – Détection du visage avec Mediapipe Landmarks (MPL). Les traits verts affichent le maillage résultant.

Puisque le modèle MPL permet d'extraire un maillage 3D dans un fichier Wavefront (.obj), on peut ensuite le visualiser sur Blender. Sur la Figure 39, on peut observer le résultat.

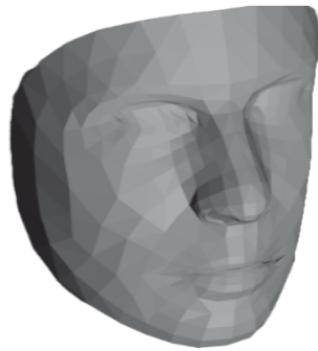


FIGURE 39 – Maillage 3D extrait en format .obj importé sur Blender.

L'outil de détection faciale permet donc de détecter des visages avec une assez bonne précision dans des cas simples et ce, en un court laps de temps. Néanmoins, cet outil présente plusieurs limitations. Tout d'abord, les *landmarks* (repères) détectés par Mediapipe ne sont pas toujours détectés dans les cas complexes, comme lorsque le visage est partiellement caché (voir Figure 40) ou bien que le contraste n'est pas optimal et qu'il y a du flou de mouvement (voir Figure 41). C'est d'ailleurs la spécificité de l'algorithme Mediapipe : il a été conçu pour être rapide et efficace et permet donc la détection faciale en temps réel, mais manque de précision comparé à d'autres algorithmes existants[23].



FIGURE 40 – Détection du visage avec Mediapipe Landmarks, échec pour un visage partiellement visible.



FIGURE 41 – Détection du visage avec Mediapipe Landmarks, échec pour un visage avec du flou de mouvement et un manque de contraste.

Étant donné les faiblesses de Mediapipe, il serait intéressant pour la suite de tester un autre modèle plus performant tel que Yunet qui permet d'obtenir de meilleurs résultats mais moins rapidement[23]. De plus, il reste à déterminer les cas de figure pour lesquels cet outil est susceptible d'être utilisé puisque son utilisation n'est pas encore pratique pour les compositeurs. Plusieurs options sont envisageables. Premièrement, il est possible de lire les points du contour du visage dans Nuke et d'en extraire un masque de rotoscopie. Deuxièmement, il peut être intéressant d'extraire une texture du visage, qui peut être modifiée puis ré-appliquée sur le maillage 3D obtenu avec MPL. On peut aussi imaginer faire fonctionner l'outil depuis les lignes de commande, afin que l'utilisateur puisse envoyer des requêtes depuis le terminal et récupérer les fichiers des données de détection. Les intérêts de cet outil ont donc évolué au fur et à mesure du temps, il est encore à déterminer précisément quelle utilisation vaut la peine d'être étudiée pour continuer son développement dans ce sens là.

5 Réflexions

Puisque c'était ma première expérience en entreprise, j'ai réellement eu l'impression d'apprendre énormément de choses, tant sur le plan professionnel et technique que personnel. Le monde du travail que j'avais imaginé n'est finalement pas si éloigné de ce à quoi j'ai déjà été confrontée dans mes projets de groupe à l'université. Au bout du compte, il s'agit simplement de personnes qui cherchent à collaborer efficacement et à prendre des décisions ensemble pour atteindre les meilleurs résultats possibles.

L'entreprise où j'ai effectué mon stage était particulièrement accueillante. J'ai été bien intégrée et j'ai eu l'impression que mon rôle était pris assez au sérieux et que les résultats de mon travail seront utiles au studio. Cela m'a permis de gagner confiance en mes compétences, ce qui est pour moi essentiel.

J'ai également pris conscience des difficultés spécifiques au monde du cinéma. Le recours fréquent à des contrats de courte durée et le manque de budget dans le secteur culturel rendent le marché instable et coûteux.

Enfin, la question de l'intelligence artificielle (IA) est omniprésente. Il y a une volonté manifeste d'utiliser l'IA, presque une pression pour montrer que l'on est à la pointe de la technologie. Cependant, cette technologie suscite aussi des craintes puisque, dans un monde où la productivité et le profit sont au centre, les artistes sont susceptibles d'être remplacés par des outils d'automatisation comme l'IA. Selon moi, l'IA devrait être utilisée en complément pour permettre aux artistes de se concentrer davantage sur les aspects créatifs, en déléguant les tâches répétitives à la machine.

6 Conclusion

Mon stage chez Benuts en tant que développeuse a été une expérience riche et formatrice sur différents aspects. J'ai eu l'opportunité de plonger au cœur de la pipeline de production des VFX, d'acquérir une compréhension approfondie d'Unreal Engine, et de développer divers outils destinés à optimiser le flux de production du studio.

Il me semble cependant que le succès de mon stage ne dépende pas uniquement des tâches que j'ai accomplies, mais également de mon évolution personnelle et professionnelle. J'ai non seulement été capable de répondre aux attentes et de réaliser les missions qui m'ont été confiées, mais j'ai également pris le temps de me poser des questions essentielles sur mes aspirations et mes objectifs pour le futur. Il me paraît aujourd'hui évident que le domaine que j'ai eu la chance d'explorer au cours de ce stage correspond à mes intérêts, tant sur l'aspect développement que sur la collaboration avec des profils créatifs et artistiques.

Cette expérience m'a permis de m'affirmer tant sur le plan technique que personnel, renforçant ma confiance en mes compétences et en ma capacité à évoluer dans le domaine des effets visuels. En somme, cette expérience chez Benuts a été extrêmement bénéfique, car elle m'a donné les outils nécessaires pour envisager de continuer à travailler avec l'équipe de Benuts, une perspective qui m'enthousiasme et m'encourage à poursuivre dans cette voie.

Références

- [1] ADOBE. *Adobe Photoshop*. URL : <https://www.adobe.com/products/photoshop.html>.
- [2] ADOBE. *Substance Painter*. URL : <https://www.adobe.com/products/substance3d-painter.html>.
- [3] Google AI. *Mediapipe Face Detection*. URL : https://mediapipe.readthedocs.io/en/latest/solutions/face_detection.html.
- [4] Google AI. *Mediapipe Face Landmarker*. URL : https://ai.google.dev/edge/mediapipe/solutions/vision/face_landmarker?hl=fr.
- [5] ANGÈLE. *La Thune*. 2018. URL : <https://www.youtube.com/watch?v=hzpql9gENgk>.
- [6] AUTODESK. *Flow Production Tracking (ShotGrid)*. URL : <https://www.autodesk.com/products/shotgrid/overview>.
- [7] AUTODESK. *Maya*. URL : <https://www.autodesk.com/products/maya/overview>.
- [8] BENUTS. *IBA*. URL : <https://benuts.be/fr/projets/iba/>.
- [9] BENUTS. *Making of Astérix et Obélix, l'Empire du Milieu*. URL : <https://benuts.be/fr/projets/asterix-et-obelix-lempire-du-milieu/>.
- [10] BENUTS. *Un Centenaire en VR : Bourdelle*. URL : <https://benuts.be/fr/projets/un-centenaire-en-vr-bourdelle/>.
- [11] *Benuts website*. BENUTS. URL : <https://benuts.be>.
- [12] Dany BOON. *8 Rue de l'Humanité*. 2021. URL : https://www.imdb.com/title/tt13834006/?ref_=vp_close.
- [13] Guillaume CANET. *Astérix et Obélix : L'Empire du Milieu*. 2023. URL : <https://www.imdb.com/title/tt11265980/>.
- [14] Meta COMPANY. *Casques Meta*. URL : <https://www.metavision.com/>.
- [15] DOUBLEDEEZ. *MDMetaEditor*. Jan. 2024. URL : <https://github.com/DoubleDeez/MDMetaEditor>.
- [16] Unreal ENGINE. *What is Virtual Production ?* URL : <https://www.unrealengine.com/en-US/explainers/virtual-production/what-is-virtual-production#>.
- [17] Blender FOUNDATION. *Blender*. URL : <https://www.blender.org>.
- [18] FOUNDRY. *Nuke*. URL : <https://www.foundry.com/products/nuke>.
- [19] Epic GAMES. *Merging Actors*. URL : https://dev.epicgames.com/documentation/en-us/unreal-engine/merging-actors-in-unreal-engine?application_version=5.3.
- [20] Discord INC. *Discord*. URL : <https://discord.com>.
- [21] Notion Labs INC. *Notion*. URL : <https://www.notion.so>.
- [22] KEENTOOLS. *Smart tools for VFX and 3D artists*. URL : <https://keentools.io/>.
- [23] LEARNOENCV. *What is Face Detection ? The Ultimate Guide*. 2019. URL : [https://learnopencv.com/what-is-face-detection-the-ultimate-guide/#MediaPipe-\(June-2019\)](https://learnopencv.com/what-is-face-detection-the-ultimate-guide/#MediaPipe-(June-2019)).
- [24] Julien LECLERCQ. *Le Salaire de la Peur (2024)*. 2024. URL : <https://www.imdb.com/title/tt15789038/>.
- [25] The Qt Company LTD. *PySide2*. URL : <https://pypi.org/project/PySide2/>.
- [26] Olivier MASSET-DEPASSE. *Largo Winch : Le prix de l'argent*. 2024. URL : <https://www.imdb.com/title/tt23049322/>.

- [27] Bettina Oberli MATHIAS GLASNER. *Night in Paradise*. 2024. URL : <https://www.imdb.com/title/tt22189180/>.
- [28] NOTLONELY. *Greenwood Fantasy Village*. 2017. URL : <https://www.unrealengine.com/marketplace/en-US/product/resubmission-for-release-greenwood-fantasy-village>.
- [29] OPENCV. *Cascade Classifier*. 2018. URL : https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.
- [30] PIXOLOGIC. *ZBrush*. URL : <https://pixologic.com/>.
- [31] Capturing REALITY. *Reality Capture*. URL : <https://www.capturingreality.com>.
- [32] SCIENCE.D.VISIONS. *3DEqualizer*. URL : <https://www.3dequalizer.com>.
- [33] SIDEFX. *Houdini*. URL : <https://www.sidefx.com/products/houdini/>.
- [34] STROMAE. *Quand c'est ?* 2015. URL : <https://www.youtube.com/watch?v=8aJw4chksqM>.
- [35] Jeffrey A. Okun SUSAN ZWERMAN. *The VES Handbook of Virtual Production*. Routledge, 2023.
- [36] *Unreal Engine*. Epic Games. URL : <https://www.unrealengine.com/fr>.
- [37] UNVIERSITÉ POPULAIRE DES IMAGES. *Histoire des trucages et effets spéciaux cinématographiques*. URL : <https://upopi.ciclic.fr/apprendre/1-histoire-des-images/histoire-des-trucages-et-effets-speciaux-cinematographiques>.

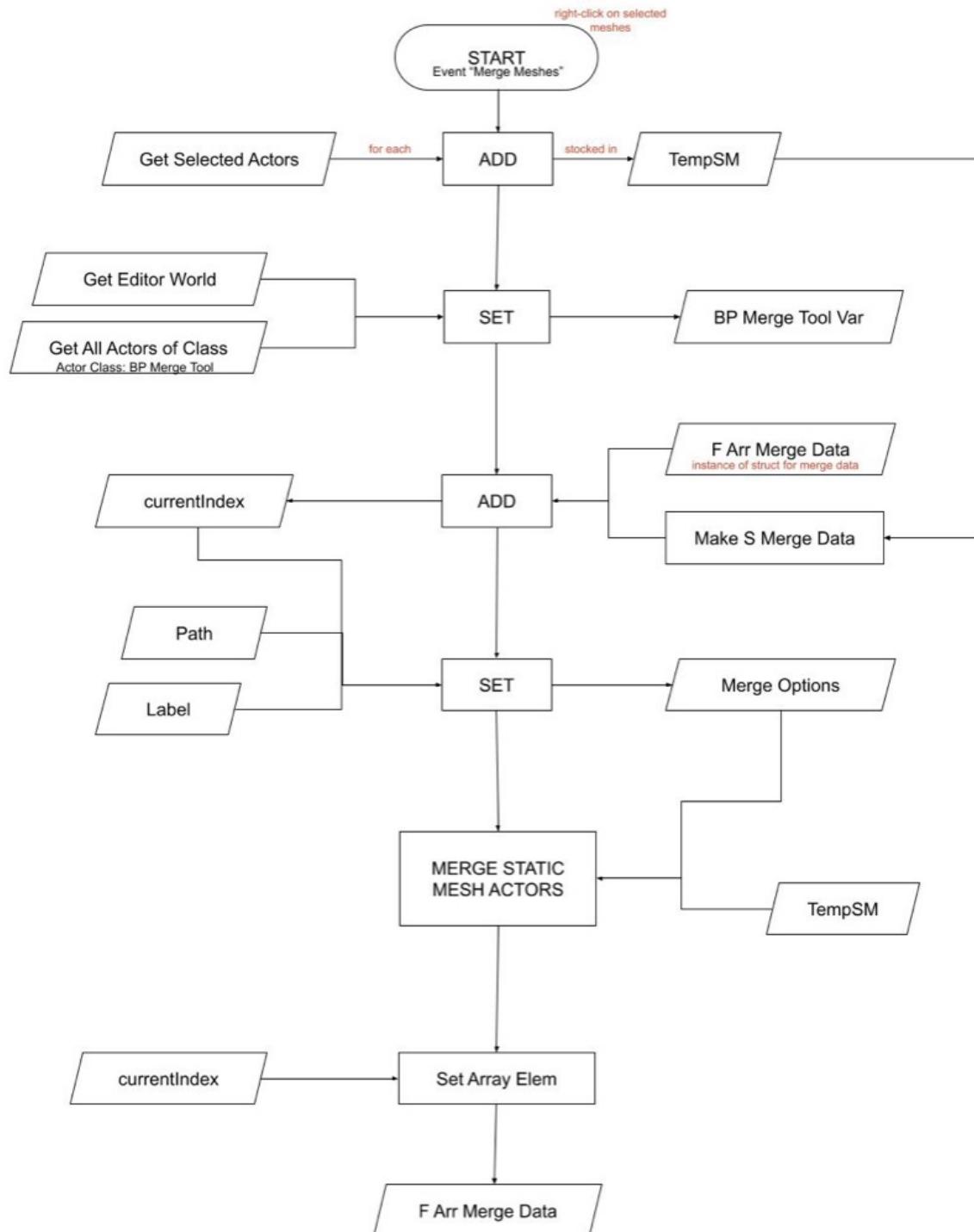
Annexes

A Tableau de ressources RnD sur Unreal Engine

Lien et titre	Tags	Type
Blueprints vs. C++: How They Fit Together and Why You Should Use Both	BP, C++, UE	video
Programming QuickStart	C++, Doc, EpicGames, UE	site
Search across the dev community	Dev Community, EpicGames, Forum, UE	site
Building Unreal Engine from source	Doc, EpicGames, GitHub, UE	site
Unreal Engine Essentials for Games: programming and Scripting	Dev Community, EpicGames, UE	site
Begin Play Tutorials	Dev Community, EpicGames, UE	site, video
Learning Library	Dev Community, EpicGames, Forum, UE	site
Programming with C++	Doc, EpicGames, UE	site
Intro to Unreal Engine C++ Tutorial	C++, UE	video
Balancing Blueprints and C++	BP, C++, Dev Community, EpicGames, UE	site
Python in Unreal Engine — The undocumented parts	Python, UE	site
Chaine Youtube Ghislain Girardot	UE	video
How to migrate assets in Unreal Engine	UE	video
How do I create an asset pack .zip file with several uassets in it?	Forum, UE	site
Ben Ui : Unreal Engine UI tutorials	BP, C++, UE	site
Unreal Directive: Quality Unreal Engine resources	Dev Community, UE	site, video
Understanding and Teaching Python for Unreal Engine	Python, UE	video
Scripting the Unreal Editor Using Python	Doc, Python, UE	site
How to create an Unreal Engine Plugin: A Step-By-Step Guide with Examples	C++, Plugin, UE	site
Unreal Python API Documentation	Doc, Python, UE	site
Editor UI Menu Names	EpicGames, Forum, UE	site
Unreal Engine C++ Complete Guide (Tom Looman)	C++, UE	site
Populating Perforce With an Unreal Project Built from Source	C++, Perforce	site
Ben Ui: How I stopped being terrified of Slates	C++, UE	site
Slate UI Framework	C++, Doc, EpicGames, UE	site
Slate UI framework slides (Gerke Max Preussner)	C++, UE	slides
UPROPERTY List of all Specifiers (ben ui)	C++, UE	site
Begin Play : Engine Structure	EpicGames, UE	video

Begin Play : Programming	EpicGames, UE	video
isaacoster: Unreal and Python	Dev Community, Python, UE	site
Understanding and Teaching Python for Unreal Engine — Unreal Educator Livestream	EpicGames, Python, UE	video
Performance Optimization for Environments — Inside Unreal	EpicGames, UE	video
Slate UI Programming	C++, Dev Community, EpicGames	site
Utilizing Python for Editor Scripting in Unreal Engine	Dev Community, EpicGames, Python	site

B Organigramme pour la conception de l'outil de fusion



C Documentation pour l'outil de fusion (extrait de Notion)



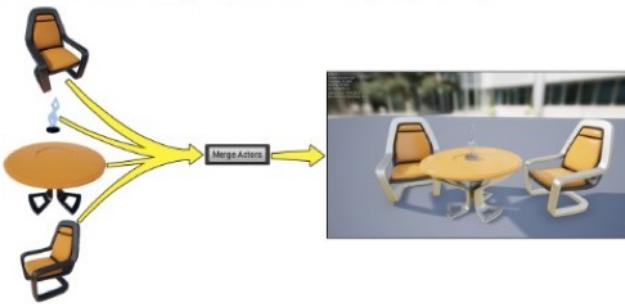
Merge Tool - Documentation

↳ 6 backlinks

- 📁 ds
 - ↳ Concept & Purpose
 - ↳ What post-settings does it apply ?
 - ↳ How to use it ?
 - ↳ Before starting:
 - ↳ How to update a cluster:
 - ↳ How to remove a cluster:
 - ↳ Good to know

Concept & Purpose

Merging Static Mesh Actors together with automated post-processing to avoid having to do too many steps to ensure a proper merge. It also allows to revert merges with automated revert post-processing.



- ▶ What post-settings does it apply ?

How to use it ?

- ▶ Before starting:
- ▶ How to create a cluster:

Once it is created, you can either update a cluster (i.e. edit its name, remove/add elements from it) or remove a cluster.

- ▶ How to update a cluster:
- ↳ ...
- ▶ How to remove a cluster:
 - ▶ "Print Duplicates" Button
 - ▶ "Restore Data from Save" Button

D Diagramme de Gantt

