Création d'images de matériaux granulaires avec la plateforme de développement de jeux Unity et prédiction de la granulométrie à l'aide de textures et de réseaux de neurones

par

Mehdi TEMIMI

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE DE LA CONSTRUCTION
M. Sc. A.

MONTRÉAL, LE 4 DÉCEMBRE 2020

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

©Tous droits réservés, Mehdi Temimi, 2020

©Tous droits réservés
Cette licence signifie qu'il est interdit de reproduire, d'enregistrer ou de diffuser en tout ou en partie, le présent document. Le lecteur qui désire imprimer ou conserver sur un autre média une partie importante de ce document, doit obligatoirement en demander l'autorisation à l'auteur.

PRÉSENTATION DU JURY

CE RAPPORT DE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. François Duhaime, directeur de mémoire
Département de génie de la construction à l'École de technologie supérieure

M. Matthew Toews, codirecteur de mémoire
Département de génie des systèmes à l'École de technologie supérieure

M. Jean-Sébastien Dubé, président du jury
Département de génie de la construction à l'École de technologie supérieure

Mme Judith Bouchard, membre du jury
Hydro-Québec

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 18 NOVEMBRE 2020

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

J’ai vraiment apprécié mes moments passés à l’ÉTS que ce soit d’abord pour mon PFE et ensuite pour ma maîtrise. J’ai rencontré de belles personnes, découvert beaucoup de nouvelles choses et surtout bien grandi. Je tiens à profiter de cette vitrine pour remercier les gens qui m’ont appuyé de près ou de loin dans mon projet.

S’il y a bien une personne qui m’a offert la possibilité de faire mes études au Canada c’est bien toi François. On s’est rencontré via le programme Mitacs Globalink, que je recommande fortement d’ailleurs, et tous les deux on avait envie d’aventure et on a fini par faire de l’analyse d’images, de l’intelligence artificielle et surtout d’utiliser des plateformes de jeux vidéo. Quelle expérience et un grand merci pour tout François. Matthew quant à toi merci de nous avoir accompagnés dans ce projet, et d’avoir pris la peine de nous donner des conseils très pertinents.

J’adresse également mes remerciements à tous mes collègues du laboratoire de recherche en génie de la construction à l’ÉTS : Javad, Céline, Pouyan, Amirhossein, Ghassan, Maxime, Dania, Telcy d’avoir rendu le laboratoire vivant. Je tiens à souligner la disponibilité et la rigueur de Mirela Sona, Jean-Sébastien Dubé et Alexis Vadeboncoeur ainsi que de leur générosité et leur engagement.

Mes remerciements vont aussi à tous mes amis qui n’ont pas hésiter à m’apporter leur bonne humeur et m’épauler : Régis, Lina, Omar, Philippe, Wissal, Imen, Dahmen, Oussama, Seif, Yasser, Sened, Raphael, Mohsen, Salem, les Peaky blinders et la liste est longue. De plus j’aimerais aussi remercier Rachit et Sasha avec qui cette aventure a commencé.

Je tiens à remercier du fond du cœur ma mère, mon père, mon frère ainsi que toute ma famille pour leur indéniable soutien et leur dévouement. Je suis chanceux de vous avoir.

Finalement, je tiens à remercier tous ceux, malheureusement oublié ici, qui ont contribué à leur façon à me faire sourire.

Création d'images de matériaux granulaires avec la plateforme de développement de jeux Unity et prédiction de la granulométrie à l'aide de textures et de réseaux de neurones

Mehdi TEMIMI

RÉSUMÉ

Les techniques d'analyse d'images peuvent être utilisées pour déterminer la granulométrie des matériaux granulaires à partir d'images. Les méthodes de granulométrie par techniques d'analyse d'images les plus récentes utilisent des paramètres texturaux comme intrants pour des réseaux de neurones artificiels. La performance de ces méthodes dépend en partie de la qualité et de la quantité des images utilisées pour l'apprentissage. En parcourant la littérature, on constate un manque important dans les jeux de données disponibles.

Le premier objectif de ce mémoire est de créer une base de données d'images réalistes de matériaux granulaires avec la plateforme de création de jeux Unity. Les images ont été créées de façon à ce que les particules aient des formes et des tailles différentes, qu'elles respectent une série de granulométries imposées et qu'elles apparaissent le plus naturel possible. Au total, 3003 images avec des particules dont la taille varie entre 75 et 1180 µm ont été préparées.

Le deuxième objectif du mémoire est de vérifier l'influence de la base de données utilisée sur les méthodes de prédiction en se basant sur des paramètres texturaux. Pour ce faire, deux bases de données supplémentaires ont été préparées : Yade, logiciel d'éléments discrets, et Unity+Yade. Celle de Yade comprend 3003 images choisies au hasard dans la base de données de Pirnia, Duhaime, & Manashti (2018) dont les particules sont sphériques. La base de données Unity+Yade combine les images de Yade et Unity pour un total de 6006 images. Les images des trois bases de données ont la même taille, la même échelle et la même masse de particules.

Trois séries de paramètres texturaux ont été extraits de chaque image : la moyenne et l'écart type de l'entropie locale, la moyenne et l'écart type des transformées par ondelettes de Haar et les textures d'Haralick. Ces caractéristiques ont été utilisées comme intrants pour des réseaux de neurones. Les réseaux ont été entrainés pour prédire le pourcentage passant pour les tamis de 106, 150, 250, 425 et 710 µm. Les performances des réseaux de neurones ont été évaluées avec la racine carrée de l'erreur quadratique moyenne (RMSE) pour les pourcentages passants.

Pour les trois types de paramètres texturaux, le RMSE moyen est de 7,0% pour les images de Yade, 8,4% pour les images de Unity et 8,5 % pour Unity+Yade. Pour les trois jeux de données, l'entropie locale a donné de meilleurs résultats (7,3%) que l'ondelette de Haar (7,9%) et les textures d'Haralick (8,7%). Ces résultats sont légèrement moins bons que ceux de Manashti, Duhaime, Toews, & Pirnia (2020) qui pour les mêmes caractéristiques ont travaillé sur 53130 images de Yade et obtenu un RMSE entre 5,6% et 7,0% pour les trois mêmes méthodes.

Mots-clés : Granulométrie par analyse d'images, Unity, Réseau de neurones artificiels, matériaux granulaires, images de matériaux granulaires.

Creation of a dataset of granular materials with the Unity game engine and prediction of particle size distribution using textural parameters and artificial neural networks

Mehdi TEMIMI

ABSTRACT

Image analysis techniques can be used to determine the particle size distribution of granular materials from images. The most recent methods for grain size analyses using image analysis techniques use textural parameters as inputs for artificial neural networks. The performance of these methods depends in part on the quality and quantity of the images used for training the neural networks. There is currently a significant gap in the available datasets in the literature.

The first objective of this thesis is to create a database of realistic images of granular materials with the Unity game engine. The images were created so that the particles have different shapes and sizes. These images had to respect a series of imposed particle size distributions, and had to appear as natural as possible. A total of 3003 images were prepared with particle sizes ranging from 75 to 1180 µm.

The second objective of this thesis is to verify the influence of the dataset on particle size predictions using textural parameters. To do so, two additional databases were prepared: Yade and Unity+Yade. The Yade database includes 3003 randomly selected images from the Pirnia et al. (2018) database. The images in this database show spherical particles created using the discrete element method. The Unity+Yade database combines the Yade and Unity images for a total of 6006 images. The images of the three databases have the same size, scale, and particle mass.

Three sets of textural parameters were extracted from each image: the mean and standard deviation of local entropy, the mean and standard deviation of Haar wavelet transforms, and Haralick textures. These characteristics were used as inputs for neural networks. The networks were trained to predict the percentage passing for sieves of 106, 150, 250, 425, and 710 µm. The neural network performance was evaluated with the root mean square error (RMSE) of the percentage passing.

For the three sets of textural parameters, the mean RMSE is 7.0% for the Yade images, 8.4% for the Unity images and 8.5% for the Unity+Yade dataset. For the three datasets, local entropy performed better (7.3%) than Haar wavelet transforms (7.9%) and Haralick textures (8.7%). These results are slightly worse than those of Manashti et al. (2020) who worked with 53130 Yade images and obtained an RMSE between 5.6% and 7.0% for the same three textural parameters.

Keywords: Grain size analysis with images, Unity, Artificial neural network, Granular materials, Images of granular materials.

TABLE DES MATIÈRES

Page

INTRODUCTION 1

CHAPITRE 1 REVUE DE LITTÉRATURE 5
1.1 La courbe granulométrique et son importance en géotechnique 5
1.2 Méthodes pour déterminer la courbe granulométrique 6
1.2.1 Description des méthodes 6
1.2.2 Intérêt de la granulométrie par analyse d'images 8
1.3 Approche déterministe pour l'analyse d'images 9
1.3.1 Principaux logiciels 10
1.3.1.1 FragScan 11
1.3.1.2 Split 12
1.3.1.3 WipFrag 14
1.3.2 Inconvénients de la méthode déterministe 15
1.4 Approche statistique pour l'analyse d'images 17
1.4.1 Entropie 17
1.4.2 Textures d'Haralick 19
1.4.3 Transformation en ondelettes 21
1.4.4 Utilisation de l'intelligence artificielle 27
1.5 Ensembles d'images pour les granulométries par techniques d'analyse d'images 29
1.6 Plateforme de développement de jeux vidéo 32
1.6.1 Principales plateformes de jeux disponibles 32
1.6.2 Utilisation de Unity en géotechnique 33

CHAPITRE 2 MÉTHODOLOGIE 37
2.1 Présentation de Unity 37
2.1.1 Interface 38
2.1.2 Caractéristiques des objets 39
2.1.3 Utilisation des scripts 40
2.1.4 Caméra 40
2.1.5 Éclairage 40
2.1.6 Asset Store 41
2.1.7 Notion d'échelle 42
2.1.8 Le réalisme de Unity à travers un exemple 42
2.1 Choix des caractéristiques des spécimens 43
2.1.1 Système d'axes 43
2.1.2 Choix des courbes granulométriques 45
2.1.3 Volume et masse du spécimen 46
2.2 Création des spécimens dans Unity 48
2.2.1 Choix des particules 48
2.3 Création des images 53
2.3.1 Générer les particules 54

2.3.2 Accélérer le modèle 55
2.3.3 Changer la masse 56
2.3.4 Affecter la granulométrie et nombre de particules 56
2.3.5 Donner une couleur réaliste 57
2.3.6 Prendre une photo du haut 59
2.3.7 Prendre une image du bas 62
2.3.8 Enregistrer la taille des particules 63

CHAPITRE 3 TRAITEMENT D'IMAGES ET INTELLIGENCE ARTIFICIELLE 65
3.1 Traitement d'images 65
3.1.1 Redimensionnement des images 66
3.1.2 Résultats intermédiaires et égalisateur d'histogramme..... 67
3.1.3 Résultat du traitement final..... 69
3.2 Extraction des caractéristiques.....69
3.3 Création des réseaux de neurones artificiels.....74
3.3.1 Bases de données 74
3.3.2 Réseaux de neurones..... 75

CHAPITRE 4 ANALYSE DES RÉSULTATS ET DISCUSSIONS 77
4.1 Images de Unity 77
4.2 Comparaison avec les images de la base de données de Pirnia et al. (2018).....80
4.3 Prédiction de la granulométrie à partir de réseaux de neurones 83
4.3.1 Images de Unity 84
4.3.2 Images de Yade..... 87
4.3.3 Les images de Unity et Yade combinées 90
4.3.4 Discussion..... 94

CONCLUSION.....101

RECOMMANDATIONS 105

ANNEXE I LES TEXTURES D'HARALICK 109

ANNEXE II LES SCRIPTS DE UNITY 111

ANNEXE III DETERMINATION DE L'ENTROPIE LOCALE,
L'ONDELETTE DE HAAR ET LES TEXTURES D'HARALICK 123

ANNEXE IV RÉSEAU DE NEURONES 127

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES.....129

LISTE DES TABLEAUX

Page

Tableau 1.1 Jeux d'images présentés dans la littérature 30

Tableau 2.1 Exemples de pourcentages passants pour une granulométrie par incréments de 10 % 46

Tableau 2.2 Nombre d'images générées en fonction de l'intervalle des pourcentages passants 46

Tableau 2.3 Tamis et masse moyenne des particules 48

Tableau 2.4 Noms des assets et références 49

Tableau 2.5 Granulométrie de l'image 6 61

Tableau 3.1 Base de données utilisées pour l'entraînement des réseaux de neurones 75

Tableau 4.1 Pourcentages passants pour les exemples d'images choisis 78

Tableau 4.2 Granulométrie des quatre images pour la comparaison entre Unity et Yade 81

Tableau 4.3 Valeurs du RMSE pour l'ensemble des tamis pour les différents cas obtenus 94

Tableau 4.4 Résultats obtenus par Pirnia et al. (2018) 97

Tableau 4.5 Résultats obtenus par Manashti et al. (2020) 97

LISTE DES FIGURES

Page

Figure 1.1 Exemple de courbe granulométrique pour un sable....6

Figure 1.2 Gamme de tailles de particules pour les différentes méthodes d'analyse granulométrique....8

Figure 1.3 Exemple de traçage du contour des particules avec un filtre de Sobel pour une méthode déterministe....10

Figure 1.4 Image de base (a) et image après traitement 2D (b)....12

Figure 1.5 Principe de la ligne de partage des eaux....13

Figure 1.6 Image initiale (a) et résultat de la segmentation avec Split (b)....14

Figure 1.7 Exemple de résultats obtenus avec WipFrag....15

Figure 1.8 a) 4 voisinages b) 8 voisinages ou voisinage 3×3 et c) voisinage 5×5....18

Figure 1.9 Photographies utilisées par Ghalib et al. (1998)....20

Figure 1.10 La corrélation en fonction du PPD....21

Figure 1.11 Exemple de réduction d'échelle pour a) et b)....22

Figure 1.12 Exemple de particules utilisées....25

Figure 1.13 Utilisation des ondelettes après sédimentation....26

Figure 1.14 Comparaison entre pourcentage passant réel et prédit....28

Figure 1.15 Exemples d'images utilisées par a) Yaghoobi et al. (2019) b) Thurley (2011) c) Andersson (2010) d) Ghalib et al. (1998) e) Ferrari et al. (2008) et f) Pirnia et al. (2018)....31

Figure 1.16 Recréation d'un site en Grèce avec Unity....34

Figure 1.17 Intérieur d'une mine....35

Figure 1.18 Simulation d'un l'éboulement grâce à Unity....36

Figure 1.19 Modélisation d'une roche en VR et visualisation sur smartphone....36

Figure 2.1 Interface de Unity 38

Figure 2.2 Exemples de a) vue de profil b) vue orthogonale et c) vue en perspective pour une même configuration 41

Figure 2.3 a) Particule de base, b) Exemple de changement de texture, c) Exemple de changement d'éclairage, d) Exemple de changement d'angle, e) Maillage convexe de la particule et f) Exemple de changement de taille.....44

Figure 2.4 Boîte contenant les spécimens 47

Figure 2.5 a) Rock1, b) Rock2, c) Rock3, d) Rock4, e) Rock5, f) Rock6, g) Rock7, h) Rock8, i) Rock9, j) Rock10 50

Figure 2.6 a) Rock1, b) Rock1 avec *box collider*, c) Rock1 avec *sphere collider*, d) Rock1 avec *mesh collider* 51

Figure 2.7 Exemple du contenu de Rock1 53

Figure 2.8 Particules générées qui vont dans la boîte 54

Figure 2.9 Exemple de particules colorées.....58

Figure 2.10 Caractéristique de la vue.....59

Figure 2.11 Exemples d'images obtenues avec a) la caméra du haut, b) la caméra de profondeur 61

Figure 2.12 Image obtenue avec la caméra du bas.....63

Figure 2.13 Exemple d'enregistrement de la taille des particules 64

Figure 3.1 Exemples d'images obtenues avec Unity : a) vue du haut et b) vue du bas 66

Figure 3.2 Exemple de résultat du traitement pour les images de la figure 3.1 68

Figure 3.3 Égalisateur d'histogramme pour a) les images du haut b) les images du bas 68

Figure 3.4 Résultats du traitement d'images après correction.....69

Figure 3.5 Exemple des disques a) r=1 et b) r=3 71

Figure 3.6 Résultats de l'entropie pour un filtrage avec un disque a) r=3 et b) r=9 71

Figure 3.7 Sous échantillonnage a) niveau 2 et b) niveau 4 72

Figure 3.8 Choix du pixel voisin et de l'angle par rapport à l'horizontal 73

Figure 4.1 Vue du haut (colonne de gauche) et du bas (colonne de droite) pour les granulométries 1 (a et b), 500 (c et d), 1000 (e et f), 1500 (g et h), 2000 (i et j) et 3003 (k et l) 79

Figure 4.2 Images combinées pour les granulométries a) 1, b) 500, c) 1000, d) 1500, e) 2000 et f) 3003 81

Figure 4.3 Comparaison des images de Unity (à gauche) et Yade (à droite) pour la même granulométrie a) 1 c) 3003 e) 2401 g) 268 a) et leurs équivalents respectifs avec Yade, b), d), f) et h) 82

Figure 4.4 Résultats de l'entropie pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis, pour les images de Unity 85

Figure 4.5 Résultats de l'ondelette pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis, pour les images de Unity 86

Figure 4.6 Résultats des textures d'Haralick pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis, pour les images de Unity 87

Figure 4.7 Résultats de l'entropie avec les images de Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis 88

Figure 4.8 Résultats de l'ondelette avec les images de Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis 89

Figure 4.9 Résultats des textures d'Haralick avec les images de Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis 90

Figure 4.10 Résultats de l'entropie avec les images combinées de Unity et Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis 91

Figure 4.11 Résultats de l'ondelette avec les images combinées de Unity et Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis 92

Figure 4.12 Résultats des textures d'Haralick avec les images combinées de Unity et Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis 93

Figure 4.13 Résultats des images de Yade dans le réseau de Unity pour
a) l'entropie, c) l'ondelette, e) les textures d'Haralick,
et des images de Unity dans le réseau de Yade pour b) l'entropie,
d) l'ondelette, f) les textures d'Haralick 96

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

GLCM	Matrice de co occurrence de niveaux de gris
PPD	Pixel Par Diamètre
RMSE	Erreur quadratique moyenne
RVB	Rouge Vert Bleu

LISTE DES SYMBOLES ET UNITÉS DE MESURE

$\%O_i$	Pourcentage observé de l'élément i
%passant	Pourcentage passant
$\%P_i$	Pourcentage prédit de l'élément i
B	Matrice de niveau de gris
d	Densité
d_x	Taille suivant x
d_y	Taille suivant y
d_z	Taille suivant z
D	Moyenne des diamètres des particules
E	Transformation de Haar
$H_b(X)$	Entropie du voisinage X
i	Incrément
I	Matrice de l'image
j	Deuxième incrément
m	Masse d'une particule
Masse_tot	Masse totale de l'échantillon
n	Nombre de particules
n_t	Nombre d'éléments total
N	Matrice de niveau de gris
N1	Base de données des images de Unity
N2	Base de données des images de Yade
N3	Base de données des images de Unity+Yade

N_G	Nombre de niveaux de gris
$p(i,j)$	Matrice normalisée de P
P	Matrice de Co occurrence
P_i	Probabilité
t	Temps
v	Volume d'une particule
xi	Élément du voisinage X
$x(t)$	Fonction réelle utilisée pour le calcul des ondelettes
x,y, z	Coordonnées cartésiennes
X	Ensemble d'éléments d'une image
X_i	Pixel situé à l'abscisse i
Y_i	Pixel situé à l'ordonnée n
α	Échelle
β	Position
μ	Écart type
ρ	Masse volumique
ρ_s	Masse volumique des solides
σ	Moyenne
$\psi\alpha,\beta\ (\ t\ )$	Ondelette fille
Ψ	Ondelette mère

Unités de mesure

cm	Centimètre
g	Gramme
kg	Kilogramme
m	Mètre
mg	Milligramme
mm	Millimètre
nm	Nanomètre
µm	Micron
°	Degré

INTRODUCTION

L'analyse granulométrique a pour objet la mesure de la taille des particules dans un matériau granulaire. Ses résultats sont représentés par l'intermédiaire d'une courbe granulométrique. En géotechnique, expériences et études ont montré que la granulométrie a une importance capitale. En effet, elle a la capacité de fournir beaucoup d'informations sur le comportement d'un sol d'une manière très simple. On peut citer la classification des matériaux et la prédiction des différentes propriétés des sols, comme la conductivité hydraulique et l'angle de frottement interne. La courbe granulométrique est une source importante d'information sur les paramètres géotechniques d'un sol (Chapuis, 2017).

Les méthodes permettant de déterminer la granulométrie sont nombreuses et dépendent de la taille des grains et de la précision nécessaire. Parmi les méthodes, il est possible de citer en exemple le tamisage par voie sèche (> 100 μm), le tamisage humide (> 25 μm), la sédimentométrie (1 à 75 μm), la centrifugation (0,1 à 2 μm) et la diffraction laser (0,02 à 900 μm). Ces méthodes, vu leur importance, doivent être reproductibles et respecter des normes spécifiques, ce qui rend le travail assez lent et difficile.

Parmi les méthodes les plus rapides, on retrouve l'analyse granulométrique par techniques d'analyse d'images. Cette famille de méthodes consiste à déterminer la taille des grains à partir de photographies. L'avantage principal de la granulométrie par analyse d'images se ressent au niveau du gain de temps lors du traitement. Comparée aux autres techniques, elle ne nécessite pas une mise en place ou un appareillage particulier. Elle permet aussi la réalisation d'analyses granulométriques in situ qui peuvent donner de l'information sur la variabilité spatiale de la distribution granulométrique sur le terrain. De plus certains matériaux sont difficiles à mesurer in situ, compte tenu de la grande taille de leurs particules ou de leur utilisation. Par exemple, sur un ouvrage de retenue de type seuil déversant en exploitation, il est impossible d'excaver les matériaux en place. Aussi, pour des matériaux d'enrochement de dimensions métriques, les volumes à excaver pour obtenir un échantillon représentatif sont très grands et nécessitent de la machinerie lourde. L'analyse d'image est aussi très utile dans ces situations.

En revanche, le lien entre les photographies et les analyses granulométriques n'est pas trivial. Pour que les méthodes basées sur les techniques d'analyse d'images soient performantes, elles nécessitent la mise en place d'un processus qui permet d'aboutir à la granulométrie à partir des images. Il existe présentement deux approches principales. Les logiciels commerciaux comme WipFrag, FragScan et Split utilisent une segmentation de l'image (Maerz, Palangio, & Franklin, 1996). En d'autres mots, ces logiciels tracent le contour des particules. Une autre approche consiste à déduire la granulométrie à partir de caractéristiques des images reliées à la texture comme les filtres d'entropie, la décomposition en ondelettes et les textures d'Haralick. Ces caractéristiques sont parfois utilisées comme intrants pour des réseaux de neurones (Manashti et al., 2020).

Le développement des méthodes d'analyses granulométriques par techniques d'analyse d'images montre un problème majeur : la vérification des méthodes et l'entrainement des réseaux de neurones demandent des jeux de données, mais ceux-ci sont assez rares dans la littérature. C'est dans ce cadre que s'inscrit l'un des objectifs majeurs de ce mémoire. Il vise à créer une base de données réaliste sur laquelle on peut faire une granulométrie par analyse d'images et avoir de bons résultats.

Le peu de bases données présentées dans la littérature peut être divisé en deux catégories. On retrouve premièrement les bases de données de photographies réelles de matériaux granulaires, comme celle de Ghalib, Hryciw, & Shin (1998). Ces bases de données comportent souvent un nombre limité de photographies en raison du temps nécessaire pour reconstituer les matériaux, les séparer, prendre les photographies et réaliser les analyses granulométriques de référence par tamisage. Pour pallier à ce désavantage, certains auteurs ont produit des bases de données d'images synthétiques de matériaux granulaires. C'est le cas de Pirnia et al. (2018) qui ont utilisé le code d'éléments discrets Yade pour produire plus de 50 000 images de matériaux granulaires. Les images contenues dans ces bases de données sont toutefois beaucoup moins réalistes avec des particules ayant des formes de sphères parfaites.

Dans cette optique, l'objectif principal de ce projet était d'utiliser une plateforme de développement de jeux pour construire une base de données plus réaliste. Cette idée découle du développement des plateformes de jeux qui sont capables de créer des environnements de plus en plus réalistes avec une capacité à traiter le détail sans équivoque. À notre connaissance, la base de données présentée dans ce mémoire est la première à utiliser une plateforme de développement de jeux pour produire des images réalistes de matériaux granulaires.

La plateforme Unity a été utilisée pour créer les images (Unity, 2019). Pour y parvenir, un programme a été conçu pour créer un total de 3003 images avec des granulométries différentes. Les granulométries ont été créées en fonction des tamis 75, 106, 150, 250, 425 710 et 1180 μm. Ces tamis correspondent à ceux utilisés dans la base de données de Pirnia et al. (2018).

Le deuxième objectif du mémoire était de vérifier l'influence de la base de données sur la performance des méthodes d'analyse granulométriques basées sur l'utilisation de caractéristiques texturales dans des réseaux de neurones artificiels. Est-ce que l'utilisation d'images plus réalistes en termes de forme et d'apparence diminue les performances des méthodes d'analyse d'images? La base de données de Pirnia et al. (2018) a été utilisée pour la comparaison. Les performances obtenues avec une base de données hybrides combinant les images de Yade et Unity ont aussi été vérifiées.

Le mémoire est divisé en quatre chapitres. Le chapitre 1 présente la revue de littérature. Dans ce chapitre, il est question de l'importance de la granulométrie. On y trouve également un justificatif de l'intérêt porté à l'analyse d'images ainsi qu'une description des deux processus utilisés pour déterminer la granulométrie à partir de photographies : l'approche déterministe et l'approche statistique. Par la suite le choix de Unity est expliqué ainsi que son utilisation en géotechnique. Le chapitre 2 traite de la méthodologie employée dans le cadre de cette étude. L'accent est mis sur la manière dont le modèle a été construit avec Unity. Le chapitre 3 explique les traitements d'images qui ont été faits sur les images de Unity afin de les préparer pour les réseaux de neurones. Cette partie traite aussi de l'extraction des caractéristiques qu'on va utiliser soit l'entropie, la décomposition en ondelettes et les textures d'Haralick. Le chapitre

4 expose les résultats obtenus, soit les images elles-mêmes et les résultats obtenus avec les réseaux de neurones pour les bases de données de Unity et de Pirnia et al. (2018). Finalement, une conclusion et certaines recommandations sont proposées ainsi que des propositions concernant les axes de recherche dans des projets ultérieurs.

CHAPITRE 1

REVUE DE LITTÉRATURE

Cette revue de littérature aborde les principaux thèmes qui entourent la création d'une base de données avec la plateforme de création de jeux vidéo Unity pour l'analyse granulométrique par technique d'analyse d'images. D'abord, l'importance des courbes granulométriques et les méthodes classiques utilisées pour déterminer la granulométrie seront présentées. Par la suite, les deux principales approches pour l'analyse granulométrique par techniques d'analyse d'images, les approches déterministe et statistique, seront explicitées. L'importance des bases de données de photographies pour ces deux approches sera ensuite présentée. Finalement, l'utilisation dans le domaine de la recherche de plateformes de développement de jeux vidéo, comme Unity, sera présentée.

1.1 La courbe granulométrique et son importance en géotechnique

La courbe granulométrique détermine la distribution de la masse de particules dans un matériau granulaire en fonction de la taille des particules. En géotechnique, elle est essentiellement déterminée par deux méthodes : le tamisage pour les grains dont la taille est supérieure à 75 μm, et la sédimentométrie pour les grains dont la taille est inférieure à 75 μm.

La figure 1.1 présente un exemple de courbe granulométrique pour un sable. L'axe des abscisses présente la taille des particules sur une échelle logarithmique. L'axe des ordonnées présente le pourcentage de la masse totale du matériau dont les grains sont plus petits que la taille en abscisse.

La courbe granulométrique est une grande source de renseignement sur le comportement des matériaux granulaires (Holtz & Kovacs, 1981). Elle est utilisée pour faire la classification des matériaux (ASTM International 2017a). La courbe granulométrique permet aussi de prédire différentes propriétés des sols, comme la conductivité hydraulique (Chapuis, 2012) et l'angle de frottement interne (Combarieu, 1996). Les méthodes de prédiction sont souvent basées sur

des tailles caractéristiques comme le D_{10}, la taille qui correspond à un pourcentage passant de 10 %. La courbe granulométrique permet aussi de vérifier la compatibilité des matériaux (conditions de filtre) dans les ouvrages en remblai (Durand, Royet, & Meriaux, 1999).

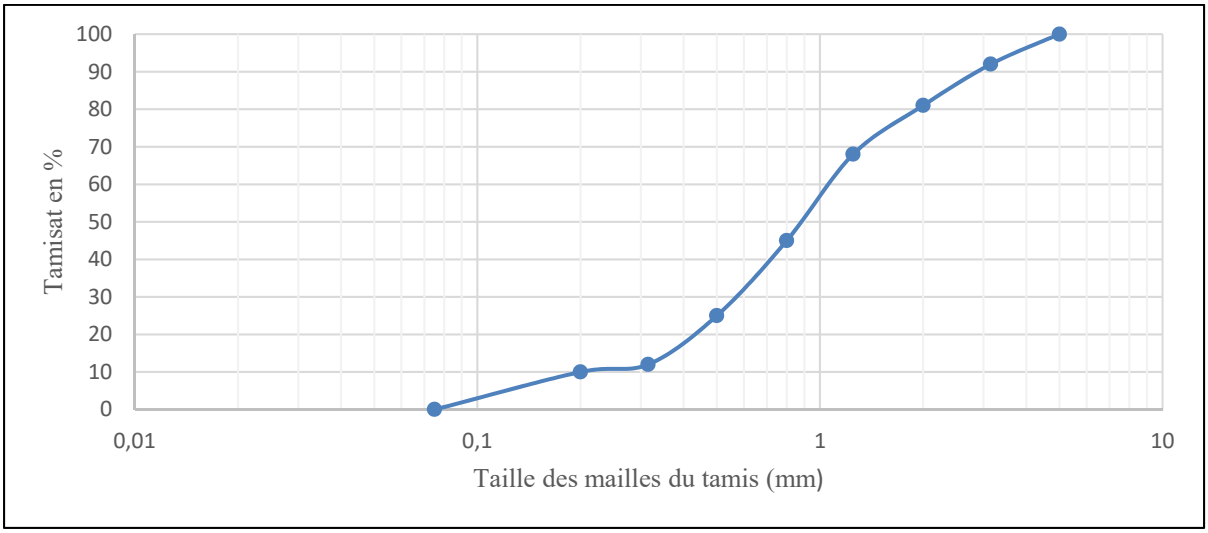


Figure 1.1 Exemple de courbe granulométrique pour un sable

1.2 Méthodes pour déterminer la courbe granulométrique

1.2.1 Description des méthodes

La courbe granulométrique est divisée en plusieurs parties. En se référant à la norme ASTM D2477 (ASTM International, 2017a), on trouve :

1. le passant 75 µm (silt et argile);
2. la fraction entre 75 µm et 4,75 mm (sable);
3. la fraction entre 4,75 mm et 75 mm (gravier);
4. la fraction retenue sur le tamis 75 mm (cailloux et blocs).

Les tamis 75 µm, 4,75 mm et 75 mm sont aussi identifiés dans la norme ASTM E11 respectivement comme les tamis No. 200, No. 4 et 3 pouces (ASTM International, 2020).

Plusieurs méthodes permettent de déterminer la courbe granulométrique des matériaux. Des essais différents sont utilisés selon la taille des particules et la précision nécessaire. Pour les tailles de grains visibles à l'œil nu (> 75 µm), on utilise le tamisage et les techniques d'analyses d'images. Pour les particules fines (< 75 µm), trois méthodes principales peuvent être utilisées : la sédimentométrie, la centrifugation analytique et la granulométrie laser. La figure 1.2 montre la gamme de tailles de particules couverte par chaque méthode.

Le tamisage est la méthode traditionnelle la plus connue (ASTM International 2017b). L'essai consiste à faire passer le matériau à travers une colonne de tamis. Ces derniers ont des ouvertures qui rétrécissent vers le bas de la colonne. Le tamis du haut a la plus grande ouverture alors que le dernier tamis du bas à la plus petite ouverture. Le tamisage existe depuis le début du 20^{e} siècle et est généralement utilisé pour les particules dont la taille est supérieure à 75 µm, même si des tamis plus petits existent (p. ex. tamis #500 de 25 µm de la norme ASTM E11 (ASTM International 2020).

La granulométrie par analyse d'images consiste à photographier le matériau granulaire, puis à analyser cette photographie à travers un logiciel. Ces méthodes seront présentées plus en détail dans les sections suivantes.

La sédimentométrie permet d'obtenir la distribution des tailles pour les particules inférieures à 75 µm. Elle est souvent utilisée en complément du tamisage. Elle est basée sur la loi de Stokes qui permet de calculer la vitesse limite d'une particule sous l'action de la gravité dans une colonne d'eau.

La centrifugation analytique détermine la masse, la forme et la composition des granulats de très petite taille comme les nano particules généralement inférieur à 2 µm (Scott, 1989). Elle sépare les composés du mélange suivant leur densité grâce à la force centrifuge.

La granulométrie laser est une méthode récente et rapide (Michel & Courard, 2006). Elle est basée sur le principe de la diffraction de la lumière. Elle permet de déterminer la taille des particules de 0,05 à 900 µm.

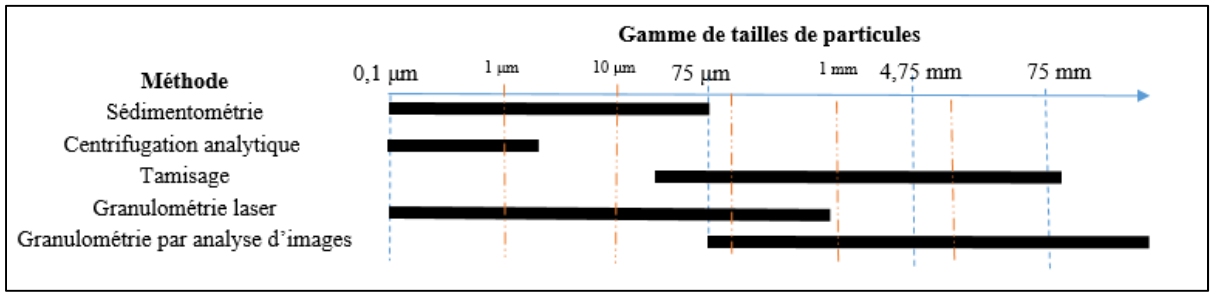


Figure 1.2 Gamme de tailles de particules pour les différentes méthodes d'analyse granulométrique

1.2.2 Intérêt de la granulométrie par analyse d'images

Plusieurs facteurs font que la granulométrie par analyse d'images montre un grand intérêt. Le tamisage nécessite beaucoup de temps. Un technicien doit préparer le spécimen, lancer le tamisage et relever les masses retenues sur chaque tamis. De plus, dans de nombreuses industries, les analyses granulométriques doivent être réalisées en ligne, par exemple sur une chaîne de production. Les techniques d'analyse d'images sont alors plus appropriées parce que plus rapides et abordables (Hamzeloo, Massinaei, & Mehrshad, 2014). Par contre, même si cette technique est en croissance, elle présente un défaut important : la vérification des méthodes demande des jeux de données réalistes, mais ceux-ci sont assez rares dans la littérature.

On peut regrouper les méthodes d'analyses granulométriques en deux catégories (Shin & Hryciw, 2004). L'approche déterministe ou encore méthode morphologique fait une segmentation des particules sur les photographies et se base sur la description des formes. L'approche statistique utilise plutôt des fonctions statistiques pour extraire les caractéristiques de l'image reliées à la texture. Ces deux familles de méthodes seront présentées dans les prochaines sections.

1.3 Approche déterministe pour l'analyse d'images

L'approche déterministe ou encore approche morphologique se base sur la description des formes des particules (Cocquerez & Philipp, 1995). Différentes techniques d'analyse d'images peuvent être utilisées pour tracer le contour des particules (Basseville, 1979). Parmi ces techniques, on cite les opérateurs morphologiques et les filtres. Les opérateurs morphologiques comme la dilatation et l'érosion agissent directement sur les formes et permettent de détecter les variations de niveau de gris aux frontières entre les grains. La dilatation agit dans le but d'élargir la figure, l'érosion dans le but de la rétrécir. Ces deux procédés sont antagonistes tels que si on appliquait une dilatation suivie d'une érosion ou inversement on aboutirait à l'image initiale. Les filtres sont des matrices, ou noyaux, qui sont appliquées sur l'image originale à travers un produit de convolution. L'application d'un filtre produit une nouvelle image pour laquelle les niveaux de gris ou les couleurs dépendent du type de filtre et de la valeur des pixels dans le voisinage de chaque pixel. Par exemple, si on veut utiliser un filtre pour produire une image qui présente la composante horizontale du gradient des intensités de gris, les pixels de chaque voisinage seront multipliés par une matrice qui contient des valeurs négatives à gauche, nulles au centre et positives à droite de manière à obtenir une valeur du gradient centrée sur le pixel central.

La figure 1.3 présente un exemple d'utilisation des filtres de Sobel. Un filtre a été appliqué à l'image de la figure 1.3a pour produire l'image de la figure 1.3b. Les filtres de Sobel calculent le gradient de l'intensité de chaque pixel. Ils permettent de détecter les changements soudains de luminosité dans une direction souhaitée en appliquant un produit de convolution sur l'image (Sobel & Feldman, 1973). Le filtre de Sobel est utilisé à titre d'exemple. Il est à noter qu'il existe plusieurs filtres permettant d'accéder aux contours. Pour plus de détails sur l'utilisation de filtres pour déterminer les contours d'un objet, le lecteur peut se référer à Gonzalez & Woods (2018).

Figure 1.3 Exemple de traçage du contour des particules avec un filtre de Sobel pour une méthode déterministe

1.3.1 Principaux logiciels

Plusieurs logiciels permettent de déterminer la granulométrie des matériaux granulaires avec une approche déterministe. Les logiciels existants ont été développés pour déterminer la granulométrie des roches fragmentées en utilisant des images obtenues sur le site. Les images sont généralement obtenues de deux manières : soit avec des caméras haute résolution installées sur des engins de transport ou des convoyeurs à bandes, soit certaines compagnies utilisent ces méthodes en prenant des photos manuellement, sur une face visible du matériau (pas nécessairement sur un convoyeur). L'échelle des images peut être obtenue en plaçant deux échelles horizontales de mêmes dimensions sur les roches.

Parmi les logiciels disponibles, on retrouve par exemple CIAS, IPACS, FragScan, Split et WipFrag. Les trois derniers sont les plus utilisés et seront présentés dans cette section.

1.3.1.1 FragScan

Développé à l'École des mines de Paris, le logiciel FragScan se base sur les travaux de Cheimanoff, Chavez, & Schleifer (1993) et Schleifer & Tessier (1996). Ce logiciel a été conçu pour estimer la distribution granulométrique des fragments de roches produits lors des opérations de dynamitage.

Les images obtenues par les caméras (figure 1.4a) nécessitent un traitement d'images. FragScan commence par utiliser un filtre passe bas pour homogénéiser localement les images. Ensuite un seuillage automatique est fait sur l'image. Le principe du seuillage est le suivant. Pour un seuil donné, si la valeur du pixel est supérieure à ce seuil il sera blanc, en revanche s'il est inférieur il sera noir. Le seuillage permet de créer une image uniquement a deux valeurs ou une image binaire (figure 1.4b). La valeur du seuil est choisie en fonction des tamis ce qui permet de mettre en valeur les particules dont la taille est au-dessus de celle du tamis. Ensuite une conversion est faite de la surface (espace 2D) au volume (espace 3D). Cette reconstruction se fait en supposant des grains sphériques et en se basant sur les principes de triangulation. Finalement FragScan simule le tamisage en utilisant plusieurs ouvertures successives pour obtenir la granulométrie. On parle ici de l'opérateur morphologique ouverture qui est une érosion suivie d'une dilatation. L'ouverture permet de faire disparaitre les petites particules et de séparer les grosses particules des petites.

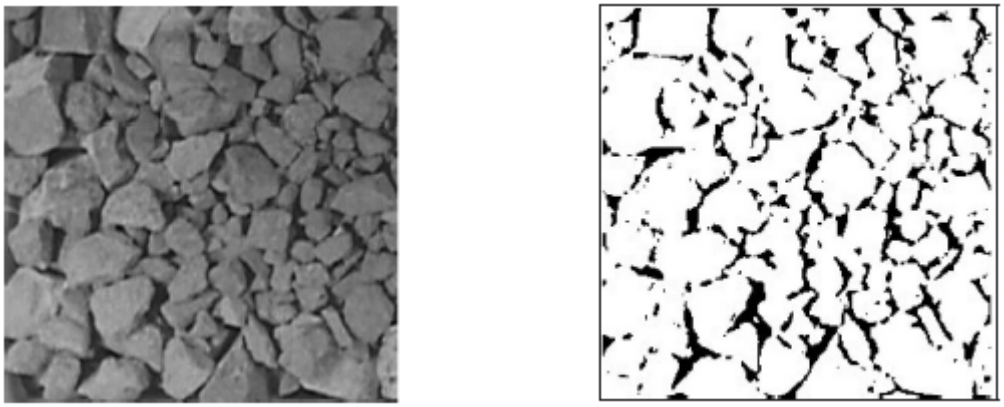


Figure 1.4 Image de base (a) et image après traitement 2D (b)
Tirée de Outal (2006)

1.3.1.2 Split

Split est un logiciel qui a été développé au début des années 90 à l'Université d'Arizona pour déterminer la distribution granulométrique des roches fragmentées (Girdner, Kemeny, Srikant, & McGill, 2018).

Split réalise la segmentation avec la méthode de la ligne de partage des eaux (LPE). Cette méthode est apparue dans les travaux de Digabel & Lantuéjoul (1978), puis Beucher (1990). Elle est inspirée du concept de ligne de partage des eaux en hydrologie. Le principe consiste à percevoir l'image comme un relief où l'altitude est remplacée par les niveaux de gris. La figure 1.5 montre un exemple de la représentation des niveaux de gris pour une colonne ou une ligne dans une image. Par analogie la ligne de partage des eaux représente le maximum local et le croisement des bassins versants les minimums locaux. Ces maximums locaux représentent les contours des objets dans une image. La LPE permet ainsi de déterminer les maximums locaux qui représentent en réalité les contours.

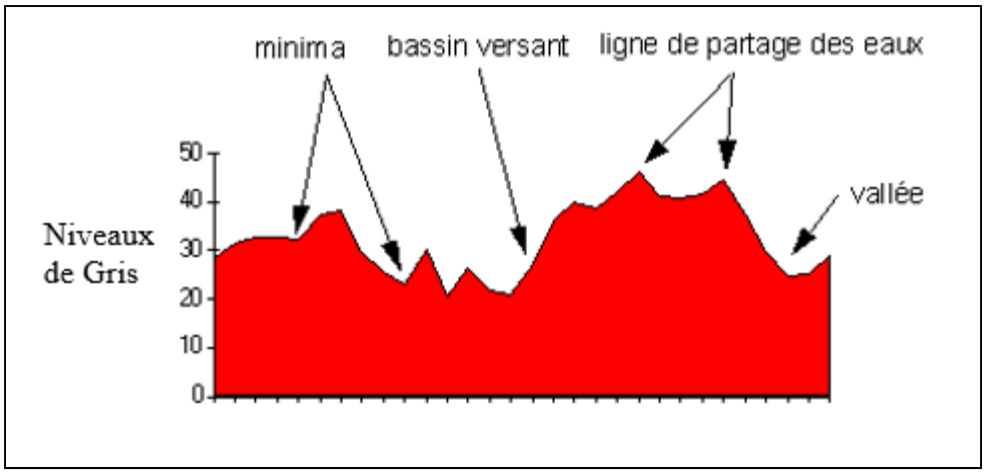
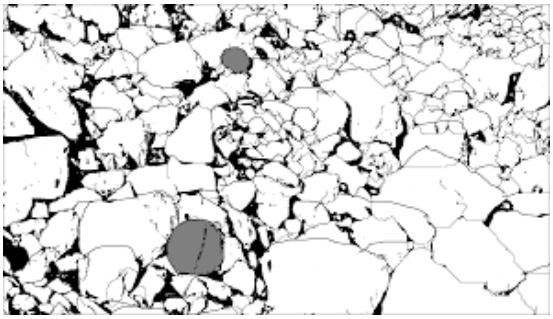


Figure 1.5 Principe de la ligne de partage des eaux

Le problème le plus fréquemment rencontré lorsqu'on utilise la LPE est la sur-segmentation. Des solutions existent comme l'utilisation de marqueur (Lefèvre, 2009). Le principe des marqueurs vient imposer un nombre à partir duquel on prend les maximums et les minimums locaux, ce qui évite la sur-segmentation. Ce nombre est choisi en fonction des objets que contient l'image, et dans le cas de split cette valeur sera prise en fonction de la taille des tamis.

Concernant l'approche 3D, le logiciel apporte deux corrections aux images. La première en se basant sur des corrélations d'analyses d'images pour la reconstruction granulométrique des roches. Split suppose une forme elliptique des roches et fait des ajustements avec la taille réelle des roches. La deuxième correction est faite concernant les particules fines : un pourcentage des zones d'ombres des images binaires est attribué aux particules fines, ce qui permet de compléter la reconstruction de la granulométrie. La figure 1.6b donne un exemple de segmentation obtenue avec Split pour l'image brute présentée à la figure 1.6a (Esen & Bølgøn, 2000).

a)



b)

Figure 1.6 Image initiale (a) et résultat de la segmentation avec Split (b)
Tirée de Esen & Bølgøn (2000)

1.3.1.3 WipFrag

WipFrag est un logiciel qui a été développé à l'Université de Waterloo au Canada et qui fut le premier à déterminer la granulométrie à partir d'images de fragments de roches (Maerz, Franklin, Rothenburg, & Linncoursen, 1987).

Lors du traitement des images, WipFrag commence par utiliser un algorithme de détection de contours comme le Laplacien ou le gradient. Cet algorithme détecte les points où il y a un changement brusque de luminosité, ce qui en réalité montre la présence de contours. Par la suite une fois ces contours détectés, un algorithme vient mesurer les surfaces obtenues puis reconstruit la granulométrie suivant les principes de probabilité qui figurent au niveau des travaux de Maerz et al. (1996). Ces principes ne sont pas clairement expliqués, mais il est mentionné qu'ils se basent sur des probabilités de types géométriques c'est-à-dire qu'ils utilisent des notions géométriques dans le rapport des probabilités comme par exemple la longueur, la largeur etc. Par la suite ces valeurs sont utilisées pour reconstruire le volume. Pour ce faire, WipFrag a recours à un calibrage empirique qui prend en compte les particules fines et les grosses particules pour lesquelles la détection des contours n'a pas été possible. Ceci n'est possible qu'après avoir traité une grande quantité d'images de la même source et après avoir fait une calibration. Si on analyse une seule ou quelques images, la calibration ne

se fait pas adéquatement. La figure 1.7 présente un exemple de détermination de la granulométrie avec WipFrag (Singh et al. 2016). La figure montre l'image de base, le résultat de la segmentation et la courbe granulométrique associée.

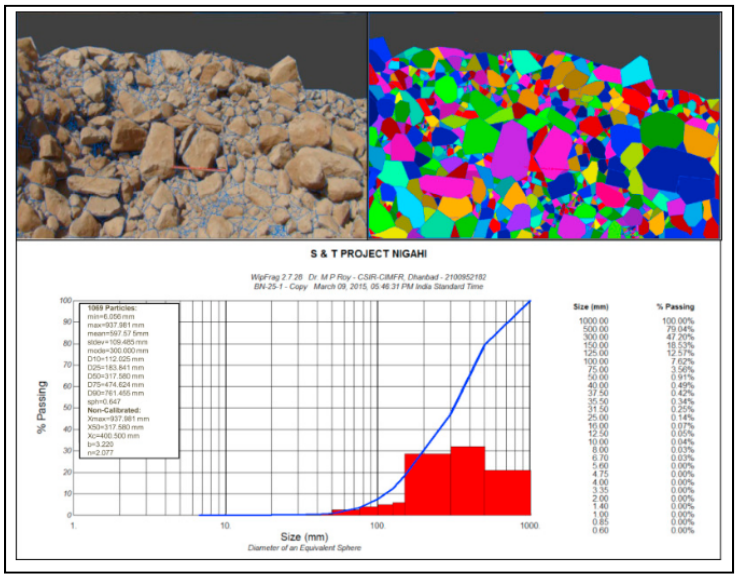


Figure 1.7 Exemple de résultats obtenus avec WipFrag
Tirée de Singh et al. (2016)

1.3.2 Inconvénients de la méthode déterministe

Cette partie présente les inconvénients de la méthode déterministe à travers les sources d'erreurs des logiciels FragScan, Split et WipFrag. Les problèmes rencontrés par ces logiciels peuvent être généralisés pour les autres logiciels et permettent de déduire les limites de l'approche déterministe.

Certaines erreurs sont tout d'abord liées aux sites. La luminosité de la photo doit être optimale. Si le temps est ensoleillé, l'ombre sur les roches sera importante. Les résultats peuvent aussi être affectés par la présence d'eau s'il a plu la veille et si les particules sont humides (Bouchard, 2016).

Certaines erreurs sont reliées à la prise d'images. Il est conseillé de prendre les photos avec un support stable et en positionnant la caméra de manière à ce que son axe soit perpendiculaire à la surface qui doit être analysée. Prendre les photos avec un angle différent peut fausser les résultats (Wipware Inc, 2013).

La ségrégation des matériaux granulaires est une autre source d'erreurs importante. Les blocs les plus gros vont rester à la surface, alors que les plus petites roches vont se glisser entre les plus grosses. Selon les méthodes de mise en place des matériaux et selon la granulométrie, l'inverse peut aussi se produire. Les blocs les plus gros se retrouvent alors au fond (Leps, 1973).

La présence de particules fines peut aussi fausser les résultats (Katsabanis, 1999; Liu & Tran, 1996; Maerz, 1998). Les méthodes déterministes n'arrivent pas toujours à détecter les particules fines. Ceci s'explique tout simplement par le fait que les images ont un nombre fini de pixels et que la taille des particules fines peut être inférieure à celle d'un pixel. Un problème de fusion apparait alors, c'est-à-dire que les particules fines sont considérées comme une seule entité au lieu de plusieurs.

Plusieurs auteurs ont évalué la précision des logiciels basés sur des méthodes déterministes. Liu & Tran (1996) ont souligné un problème de surestimation de la taille des particules. Ils ont montré que pour les tailles inférieures à 25 mm (1"), FragScan et WipFrag surestimaient la taille des particules. L'erreur était d'autant plus importante que la taille diminuait. La taille médiane D_{50} pouvait être surestimée de 50 % pour Split et WipFrag et 100 % pour FragScan. Holmberg (2003) a quantifié les erreurs liées à la sensibilité des logiciels. Il a montré que pour une distribution granulométrique étalée, l'erreur sur les pourcentages passants pouvait atteindre 10 à 20 % pour le logiciel FragScan. Enfin, Maerz (1998) a montré les conséquences d'un mauvais calibrage. Mal calibrer le logiciel en fonction du matériau peut donner un taux d'erreur dans l'estimation du D_{50} pouvant atteindre les 35 % avec WipFrag (Maerz, 1998). En contrepartie, un bon calibrage réduit ce taux d'erreur. L'erreur sur le D_{50} peut aller jusqu'à 4% pour le cas d'un calcaire traité avec WipFrag (Maerz et al., 1996).

1.4 Approche statistique pour l'analyse d'images

L'approche statistique vient explorer les textures de l'image pour en extraire des données statistiques. Ces données peuvent ensuite être utilisées pour trouver la taille des grains (Shin & Hryciw, 2004). La partie qui suit présente des exemples des paramètres texturaux qui peuvent être utilisés pour déterminer la granulométrie à partir d'images.

1.4.1 Entropie

L'entropie a été introduite par Shannon (1948) et porte aussi le nom d'entropie de Shannon. L'entropie est un concept plus vaste que l'entropie en thermodynamique, d'où elle tient son nom. Dans les deux cas, l'entropie mesure le désordre d'un système.

L'entropie de Shannon représente la quantité d'information que contient une source d'information. Elle témoigne de la complexité d'une image. Une image uniforme qui ne contient qu'un seul niveau de gris a une entropie nulle. Plus la valeur de l'entropie est élevée plus les niveaux de gris de l'image sont variables. L'entropie d'une image est définie par l'équation suivante, en supposant l'image comme une variable aléatoire discrète :

$$H_b(X) = -\sum_{i=1}^{N_G} P_i \log P_i \tag{1.1}$$

où :

- $H_b(X)$*:* L'entropie du voisinage X;
- N_G : Le nombre de niveaux de gris;
- P_i *:* La probabilité d'occurrence du niveau de gris *xi*, élément de l'ensemble du voisinage *X*.

Le signe moins permet de donner une valeur positive à l'entropie. Le logarithme d'une probabilité est toujours négatif vu que $0 \leq P_i \leq 1$.

La valeur de N_G correspond au nombre de séquences de niveaux de gris considérés (*bins* en anglais). Les pixels sont codés de 0 à 255 niveaux de gris dans une image en noir et blanc. Il est important de rassembler les niveaux de gris dans des séquences. Autrement, l'entropie est maximum dès que les niveaux de gris du voisinage sont différents, même s'ils varient très peu. Par exemple, il est possible de définir deux séquences qui contiennent chacune 128 niveaux de gris, ou encore de prendre 256 séquences qui contiennent chacune un niveau de gris. Plus le nombre de séquences est élevé, plus l'entropie est grande.

L'entropie peut être calculée pour une image complète ou au voisinage de chaque pixel comme c'est le cas avec l'équation 1.1. La figure 1.8 présente des exemples de voisinages. Dans chaque cas, l'entropie est calculée avec tous les pixels du voisinage et est assignée au pixel central du voisinage (pixel rouge). Si tous les pixels du voisinage ont le même niveau de gris, l'entropie est nulle. Si tous les pixels ont un niveau de gris différent, l'entropie est maximale. Par la suite, la moyenne et l'écart type des valeurs d'entropie pour l'image peuvent être calculées (Larkin, 2016). La moyenne et l'écart type de l'entropie peuvent être calculés pour différents voisinages, par exemple pour des voisinages dont la taille varie.

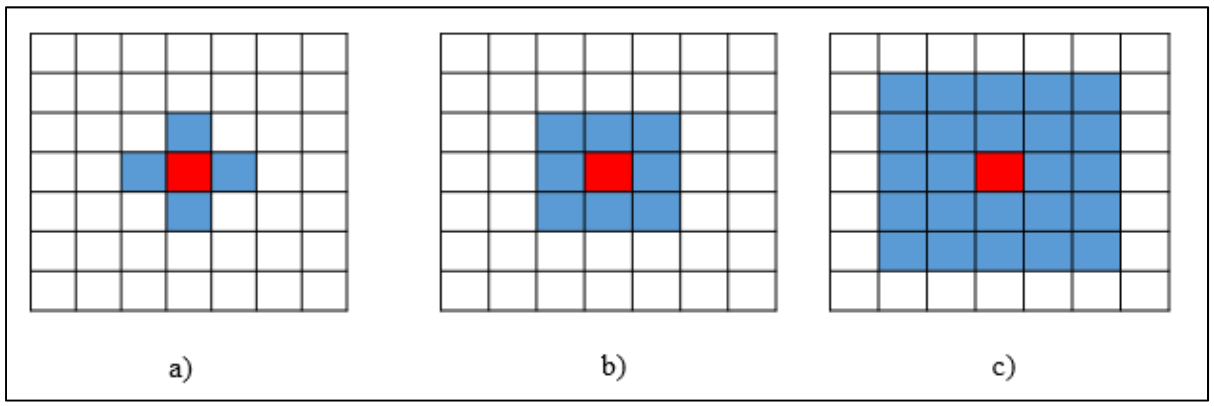


Figure 1.8 a) 4 voisinages b) 8 voisinages ou voisinage 3×3 et c) voisinage 5×5

Manashti et al. (2020) ont utilisé la moyenne et l'écart type de l'entropie locale pour différents voisinages comme intrants dans un réseau de neurones pour déterminer la granulométrie. Des images synthétiques de matériaux granulaires dont la taille des particules varie de 75 à

1180 µm ont été utilisées. Ils ont montré que le choix de la séquence et du voisinage affecte la valeur de l'entropie et la qualité des prédictions de la granulométrie. Pour l'entropie, plus le choix de la séquence est petit, plus la valeur de l'entropie est sensible aux faibles variations des niveaux de gris. En ce qui concerne le voisinage, plus celui-ci est grand, plus il est possible d'accéder facilement aux plus grandes particules.

1.4.2 Textures d'Haralick

Les textures d'Haralick sont apparues en 1973 (Haralick, Shanmugam, & Dinstein, 1973). Elles regroupent quatorze caractéristiques de l'image. Ces caractéristiques sont des statistiques des niveaux gris de l'image.

L'image couleur est tout d'abord convertie en niveaux de gris. La matrice de co-occurrence des niveaux de gris (en anglais *Gray-Level Cooccurence Matrix*, GLCM) est ensuite déterminée. Chaque élément $P_R(i, j)$ de cette matrice dénombre le nombre de paires de pixels qui correspondent aux niveaux de gris i et j et dont la position relative peut être décrite par une relation R. Par exemple, la GLCM peut être calculée pour des pixels (x_1,y_1) et (x_2, y_2) voisins sur une ligne horizontale ($x_1 = x_2 +1$, $y_1 = y_2$). La GLCM est ensuite normalisée, en la divisant par la norme de cette matrice, de manière à ce que la somme des éléments de la matrice soit égale à 1.

Quatorze textures sont calculées à partir de la GLCM. On citera en exemple trois textures qui sont souvent utilisées, soit le moment angulaire, la corrélation et la variance. Les autres textures sont présentées à l'annexe I.

$$\text{Le moment angulaire } f_1 = \sum_i \sum_j p(i,j)^2 \tag{1.2}$$

$$\text{La corrélation } f_3 = \frac{\sum_i \sum_j (ij)\, p(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y} \tag{1.3}$$

$$\text{La variance } f_4 = \sum_i \sum_j (i-\mu)^2 p(i,j) \tag{1.4}$$

sachant que :

- $p(i, j)$ est la matrice normalisée (divisée par la norme) de la GLCM;
- $\sigma_x, \sigma_y, \mu_x \ \ et \ \ \mu_y$ sont respectivement les moyennes et les écarts types des probabilités relatives suivant x et y, p_x et p_y.

Les textures d'Haralick et al. (1973) ont été extraites par Ghalib et al. (1998) pour des photographies de sable angulaire dont la taille des particules varie de 0,25 à 0,42 mm (figure 1.9). Il est à noter que les auteurs ont utilisé la notion de PPD (*pixels per diameter*) qui représente la moyenne des diamètres des particules en pixels. Cette notion sera expliquée plus en détails à la section 1.4.3. Les textures d'Haralick ont été comparées aux valeurs de PPD et des relations ont été trouvées. La figure 1.10 présente par exemple la relation entre la corrélation et les valeurs de PPD. Ces résultats ont été utilisés dans un réseau de neurones et ont permis de déterminer la taille des grains de sable.

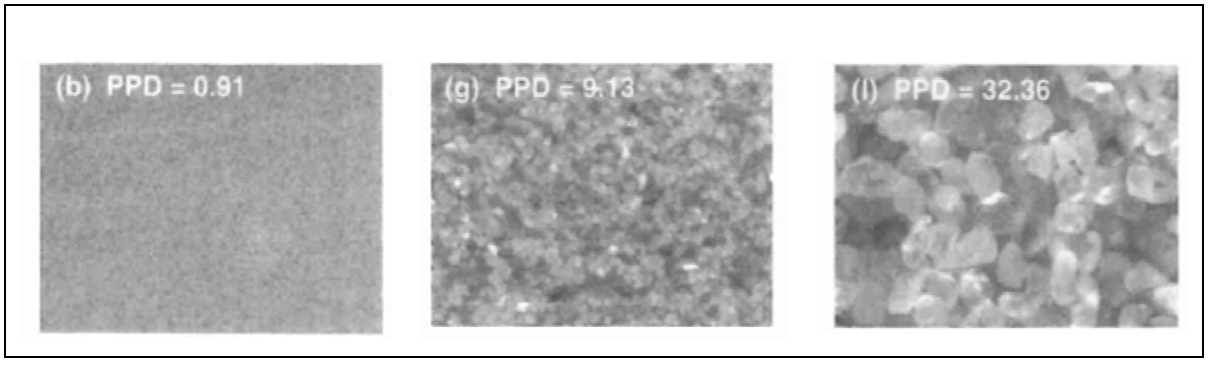


Figure 1.9 Photographies utilisées par Ghalib et al. (1998)
Tirée de Ghalib et al. (1998)

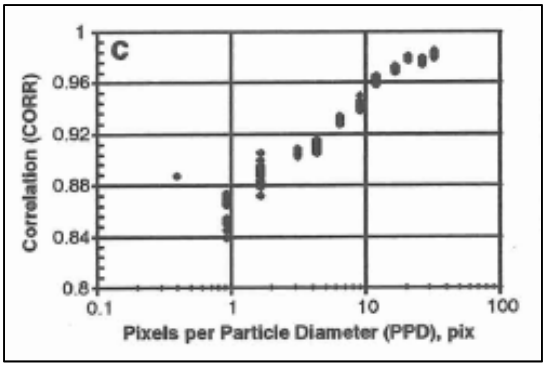


Figure 1.10 La corrélation en
fonction du PPD
Tirée de Ghalib et al. (1998)

Plus récemment Manashti et al. (2020) ont utilisé les textures d'Haralick pour déterminer la granulométrie sur des images de sols synthétiques dont la granulométrie varie de 75 à 1180 µm. Les textures ont été utilisées comment intrants dans un réseau de neurones dans le but de prédire le pourcentage passant. L'erreur sur les pourcentages passant est de l'ordre de 6 %.

Les textures d'Haralick ont aussi certaines limites. Hryciw, Ohm, & Zhou (2013) ont montré qu'elles sont dépendantes de la luminosité et de la couleur des particules.

1.4.3 Transformation en ondelettes

La transformation en ondelettes est une technique qui permet de décomposer et de reconstruire un signal sans perdre d'informations. Cette transformation complète la transformation de Fourier et comble certains aspects non traités par cette méthode. Le terme ondelette vient du fait que cette méthode utilise une petite onde ou oscillation. Cette technique fut inventée en 1909 par Alfred Haar (Haar, 1910) qui définit la transformation de Haar : un signal facile à comprendre et dont la fonction statistique est simple à utiliser. Du fait de sa simplicité la transformation de Haar est très utilisée dans les projets de recherche courants. En analyse d'images, la transformation par ondelettes est appliquée à la compression d'images.

L'utilisation de la transformation par ondelettes pour déterminer la granulométrie a débuté avec Ool & Soria (1992) pour extraire une image d'une particule d'une taille souhaitée, et s'est développée par la suite (Amankwah & Aldrich, 2011). La manière d'utiliser la transformation par ondelettes est généralement la même avec les étapes suivantes :

- une réduction d'échelle : Procédure qui permet d'aller chercher le détail à grande résolution de l'image en diminuant sa taille. La figure 1.11 montre un exemple de réduction d'échelle pour un sable. L'image a) à gauche a une taille de 5,22 Mo, contre 80,3 Ko pour l'image b) à droite sachant que les deux images ont la même taille soit 3648×2736;

a)

b)

Figure 1.11 Exemple de réduction d'échelle pour a) et b)

- utilisation de la fonction statistique de la transformation en ondelette : d'ordre général la fonction utilisée est de la forme suivante :

$$g(\alpha,\beta)=\frac{1}{\sqrt{\alpha}}\int_{t=-\infty}^{t=+\infty}x(t)\overline{\psi_{\alpha,\beta}(t)}dt \tag{1.5}$$

tels que :

- $\psi\alpha,\beta\ (t)$ est obtenue par translation et dilatation de l'ondelette mère;
- $\psi\alpha,\beta\ (t)=\Psi((t-\beta)/\alpha);$
- $x(t)$:une fonction réelle;

- β :position;
- α : échelle.

Plusieurs fonctions permettant de définir les ondelettes existent comme l'ondelette de Haar. Cette fonction qui est très utilisée se base sur les étapes suivantes :

- faire une moyenne des pixels horizontaux deux à deux ;

$$H(x) = \frac{X_i + X_{i+1}}{2} \tag{1.6}$$

- calculer suivant la direction horizontale l'erreur entre l'image sous échantillonnée et l'image originale ;

$$G(x) = \frac{X_i - X_{i+1}}{2} \tag{1.7}$$

- faire une moyenne des pixels verticaux deux à deux des images obtenues ;

$$H(y) = \frac{Y_i + Y_{i+1}}{2} \tag{1.8}$$

- calculer suivant la direction verticale l'erreur entre les images obtenues.

$$H(y) = \frac{Y_i - Y_{i+1}}{2} \tag{1.9}$$

où

- X_i et X_{i+1} représentent les pixels situés respectivement à la place i et $i+1$ sur l'axe horizontal;
- Y_i et Y_{i+1} représentent les pixels situés respectivement à la place i et $i+1$ sur l'axe vertical;

- déterminer les textures des ondelettes : Tout comme les textures d'Haralick, les ondelettes exploitent les textures des équations mathématiques. On cite par exemple l'énergie de la transformation de Haar (Haar, 1910) définie comme :

$$E_{I(x,y)} = \sum I(x,y)^2 \tag{1.10}$$

Avec :

- I : matrice de l'image;
- E : matrice de la transformée de Haar.

Une fois les caractéristiques extraites, les méthodes divergent quant à l'utilisation des ondelettes pour déterminer la granulométrie. Shin & Hryciw (2004) ont utilisé la transformation en ondelette et l'ont associé au PPD ou pixels per diameter en anglais. Cette notion est très intéressante car elle représente la moyenne des diamètres des particules par pixel (équation 1.13) et permet d'exprimer le diamètre comme un paramètre intrinsèque à l'image :

$$D(mm/diamètre) = \frac{PPD(pixels/diamètre)}{grossissement(pixels/mm)} \tag{1.11}$$

Où le grossissement représente la résolution et D représente la taille moyenne des particules en mm / diamètre. De ce fait, connaissant le diamètre moyen des particules utilisées et la résolution de la caméra, il est facile de déterminer le PPD. Cette dernière a montré des résultats intéressants sur un sable fin dont la taille des particules varie de 0,15 mm à 0,18 mm et un autre sable dont la taille des particules varie de 0,249 mm et 0,419 mm. D'autres recherches comme Hryciw et al. (2013), ont montré qu'on pouvait utiliser la transformation de Haar pour trouver une relation entre l'énergie qui lui est associée et le PPD. Cette étude a été faite sur des particules dont la taille varie de 0,075 mm à 2,0 mm (ce qui correspond aux tamis US standards No. 200 et No. 10). La figure 1.12 donne un exemple des particules utilisées. La résolution est de 36/37 pixels /mm ce qui situe la particule de 0,075 mm à 3 PPD et 2,0 mm à 73 PPD.

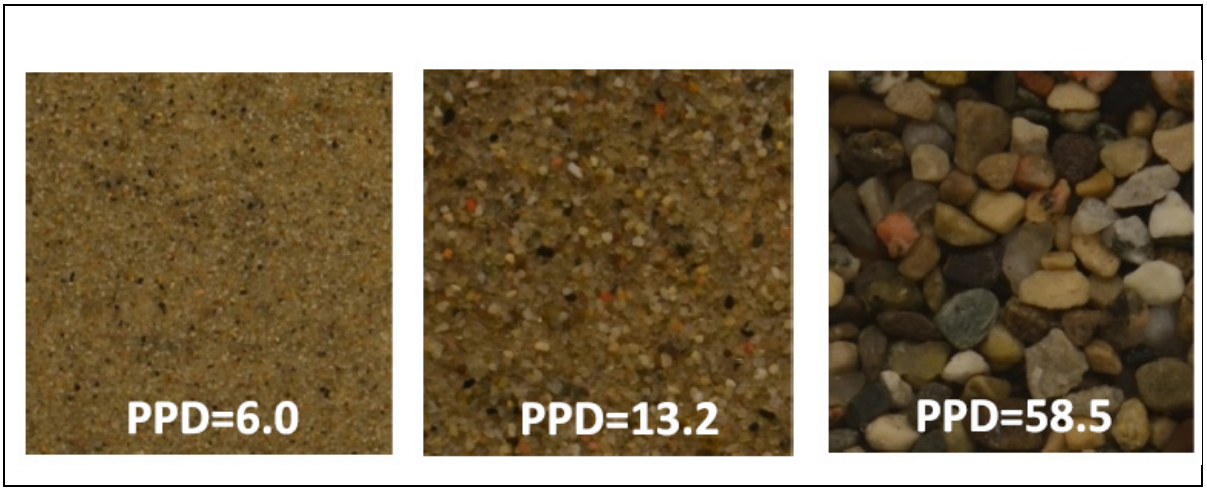


Figure 1.12 Exemple de particules utilisées
Tirée de Hryciw et al. (2013)

La transformation en ondelette a aussi été utilisée pour la classification AASHTO des sols. Hryciw & Jung (2009) ont utilisé la décomposition en ondelettes sur des images dont le sol a subi une sédimentation pour les tailles 0,075; 0,425 et 2,0 mm. C'est-à-dire qu'une fois que le sol a été sédimenté des images ont été prises pour les plages citées précédemment (figure 1.13). Les $PPD_{10\ 40\ 200}$ sont les PPD associées aux tamis US 10, 40 et 200 qui représentent les tailles 0,075; 0,425 et 2,0 mm. Ces résultats ont montré qu'il est possible d'utiliser la méthode des ondelettes pour donner un équivalent de la classification AASHTO.

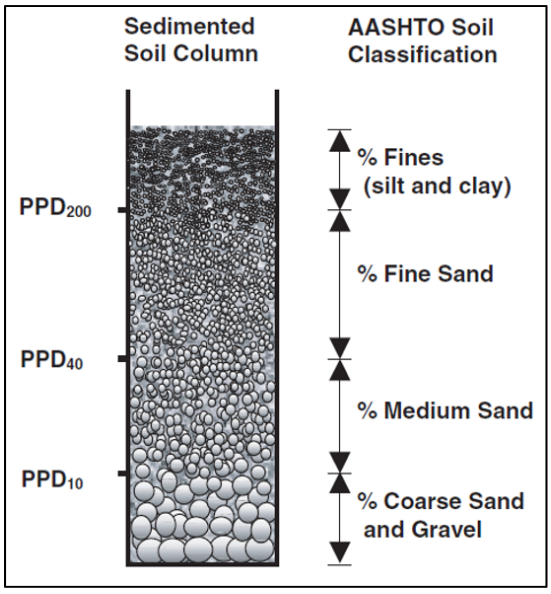


Figure 1.13 Utilisation des ondelettes
après sédimentation
Tirée de Hryciw & Jung (2009)

Plus récemment Manashti et al. (2020) ont utilisé les ondelettes comme intrant d'un réseau de neurones d'images de sols synthétiques dont la granulométrie varie de 75 à 1180 µm . Ces textures ont été extraites de façon à ce qu'elles soient utilisées dans un réseau de neurones qui prédit le pourcentage passant. Les résultats obtenus dans ces travaux ont donné une erreur de 6 % dans la prédiction du pourcentage passant par rapport pourcentage réel.

Cette méthode est prometteuse mais présente néanmoins une contrainte majeure. Il n'est pas possible d'appliquer cette technique sur un matériau dont la granulométrie varie (Hryciw et al., 2013). Il est important d'avoir une répartition uniforme, c'est-à-dire que chaque image pour laquelle la décomposition en ondelette est faite, doit contenir plus ou moins la même granulométrie et non une seule image qui contient des particules dont la taille varie de 0,075 à 2 mm. Ceci explique les gammes de mesures utilisées dans les travaux précédemment cités ou le besoin d'utiliser la sédimentation dans la classification AASHTO.

Ainsi dans le but d'améliorer cet aspect-là, un autre courant pour déterminer la granulométrie par analyse d'images basé sur l'intelligence artificielle a vu le jour.

1.4.4 Utilisation de l'intelligence artificielle

L'intelligence artificielle ou IA se base essentiellement sur l'auto apprentissage, là où le modèle créé corrige automatiquement et itérativement les problèmes rencontrés en fonction des nouvelles informations obtenues.

L'IA s'utilise essentiellement à travers un réseau de neurones, qui a le même fonctionnement des neurones biologiques pour apprendre. Parmi les réseaux de neurones les plus utilisés on cite :

- le réseau classique dit « *feed forward* » : La propagation de l'information passe de l'entrée à la sortie en passant par les nœuds de traitement;
- le réseau de neurones récurrents : même principe que le réseau classique sauf que les nœuds de traitement enregistrent les informations et les renvoient au modèle;
- le réseau de neurones à convolution : réseau de neurone qui prend en entrée des images et les traite.

On retrouve quelques exemples d'utilisation de l'IA pour la prédiction de la granulométrie dans la littérature. Ghalib et al. (1998) ont utilisé un réseau de neurones artificiels pour lier les textures d'Haralick au PPD, ce qui a permis d'estimer la taille de particules dont la taille varie de 0,25 à 0,42 mm avec une erreur de 3 % sur la valeur du PPD. Facco, Santomaso, & Barolo (2017) ont estimé le pourcentage passant des particules à partir de matériaux en vrac. Les résultats montrent qu'on peut estimer le pourcentage passant des mélanges granulaires avec une précision élevée en utilisant la régression partielle des moindres carrés Andersson (2010) a utilisé la vision par ordinateur pour mettre l'accent sur les différentes sources d'erreurs lors de l'échantillonnage des particules sur les bandes transporteuses. Cette étude a traité les boulettes de minerai de fer dont la taille varie de 6,3 à 14 mm, des roches de basaltes fragmentées avec une taille de 11,2 à 31,5 mm et finalement du calcaire broyé avec une taille de 10 à 100 mm. Ferrari, Piuri, & Scotti (2008) ont utilisé un réseau de neurones RBF (de l'anglais Radial basis function network) ou un réseau de fonctions de base radiale pour déterminer la taille des particules. Le RBF est un type de réseau de neurones qui a comme

fonction d'activation des fonctions à base radiale. Il est essentiellement constitué de trois couches : La couche d'entrée, de sortie et une couche cachée. Enfin Pirnia et al. (2018) ont élaboré un réseau de neurones de convolution dans le but d'estimer le pourcentage passant des particules pour les cinq tamis suivants : 106, 150, 250, 425 et 710 µm. Les résultats montrent un taux d'erreur global faible de l'ordre de 6 % sur la prédiction du pourcentage passant par rapport au pourcentage réel (figure 1.14). Finalement, Manashti et al. (2020) ont utilisé des réseaux de neurones sur des caractéristiques extraites tels les ondelettes, les filtres de Gabor, la transformée de Fourier et autre, directement sur ces images avec les mêmes tamis. Le taux d'erreur moyen obtenu par ces travaux a même atteint 3,8 % pour certaines caractéristiques.

Quels que soient les travaux mentionnés précédemment, l'utilisation de l'IA à travers la construction d'un réseau de neurones artificiels ou de convolution ou la vision par ordinateur, il est important d'avoir un ensemble d'images de qualité. La partie qui suit traite cette idée.

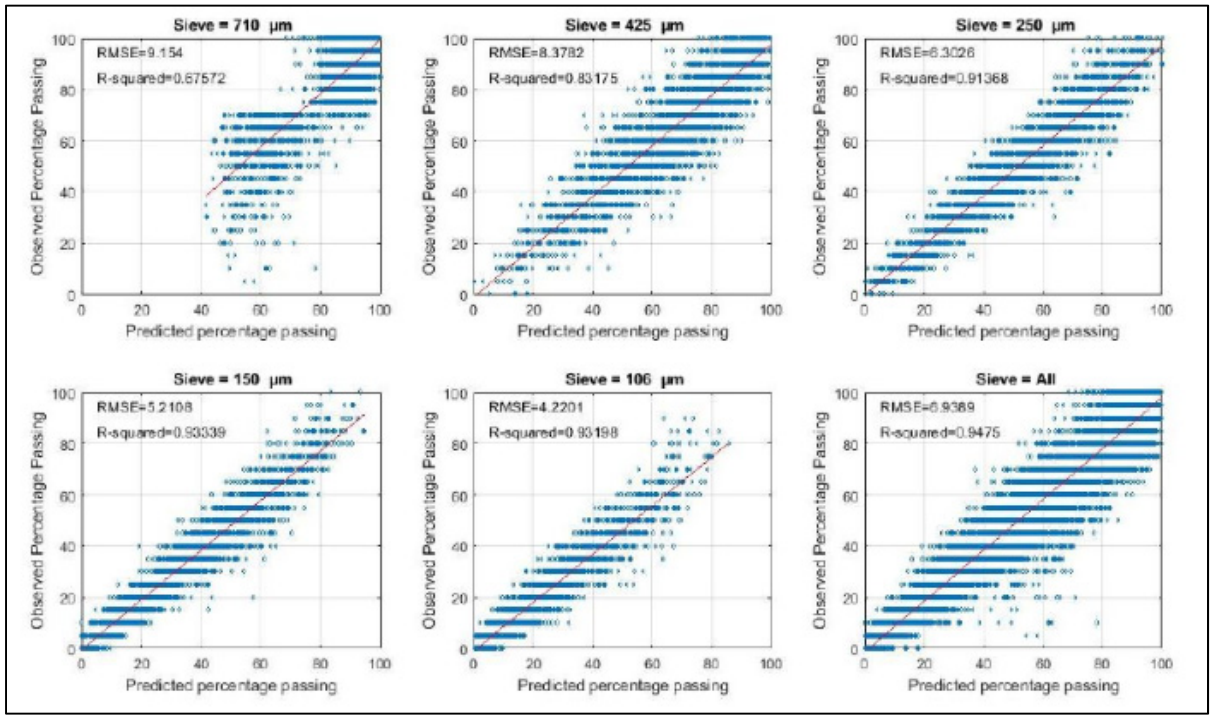


Figure 1.14 Comparaison entre pourcentage passant réel et prédit
Tirée de Pirnia et al. (2018)

1.5 Ensembles d'images pour les granulométries par techniques d'analyse d'images

Plusieurs ensembles d'images ont été utilisés dans la littérature pour valider les méthodes d'analyses granulométriques par techniques d'analyse d'images. Les ensembles d'images peuvent être utilisés pour l'apprentissage des réseaux de neurones et pour la validation des méthodes déterministes. Les ensembles d'images se distinguent par le nombre d'images qu'ils contiennent, le type du matériau photographié, et le type d'images. Le nombre d'images varie grandement d'une référence à l'autre. Par exemple, Ko & Shang (2011) présentent seulement 12 images alors que Pirnia et al. (2018) en présentent 53 130. Au niveau des types de matériaux, les images montrent principalement des matériaux granulaires naturels en géotechnique et de la pierre concassée dans le domaine minier. Les images peuvent être des photographies ou des images synthétiques. Le réalisme de ces dernières varie. Le tableau 1.1 présente une synthèse des ensembles d'images disponibles dans la littérature.

Les travaux 1,2 et 3 (respectivement a), b) et c) dans la figure 1.15) même si le nombre des images contenues dans la base de données est bien, la taille des particules est assez grande et ne contient que très peu de matériaux granulaires. Le travail 5 (image d) dans la figure 1.15) traite les particules granulaires, en revanche le nombre n'est pas suffisant. Les travaux 4 et 6 traitent les particules granulaires avec une base de données très riche. En revanche les images sont synthétiques et ne sont donc pas réelles. La figure 1.15 montre un exemple pour les travaux de Ferrari et al. (2008) et Pirnia et al (2018) (respectivement a), b) où les particules ont une forme sphérique, ce qui n'est pas le cas pour des particules réelles.

De ce fait on arrive à remarquer qu'il n'est pas évident de générer une base de données consistante. Soit les images sont de sols réels mais en petite quantité, soit elles sont synthétiques mais de taille importante.

Ce travail s'inscrit dans ce cadre et vise à trouver un juste milieu : Une base de données assez importante avec des images de sols beaucoup plus réelles qui définissent les particules comme à l'état naturel et non sphérique.

Tableau 1.1 Jeux d'images présentés dans la littérature

Numéro	**Travaux**	**Type de roches**	**Taille**	**Taille de la base de données**
1	Yaghoobi, Mansouri, Ebrahimi Farsangi, & Nezamabadi-Pour (2019)	Quartz Gneiss	0,02 à 63,5 cm	226
2	Thurley (2011)	Calcaire	10 à 100 mm	600
3	Andersson (2010)	Minerai de fer roches de basaltes et du calcaire broyé	20 à 90 mm	1549
4	Ferrari et al. (2008)	Sable bitumineux, les copeaux de bois et les grains de café	3,15 à 45 mm	7000 images synthétiques
5	Ghalib et al. (1998)	Sable de carrière	0,25 à 0,42 mm	133
6	Pirnia et al. (2018)	Sable	75 à 1180 µm	53130 images synthétiques

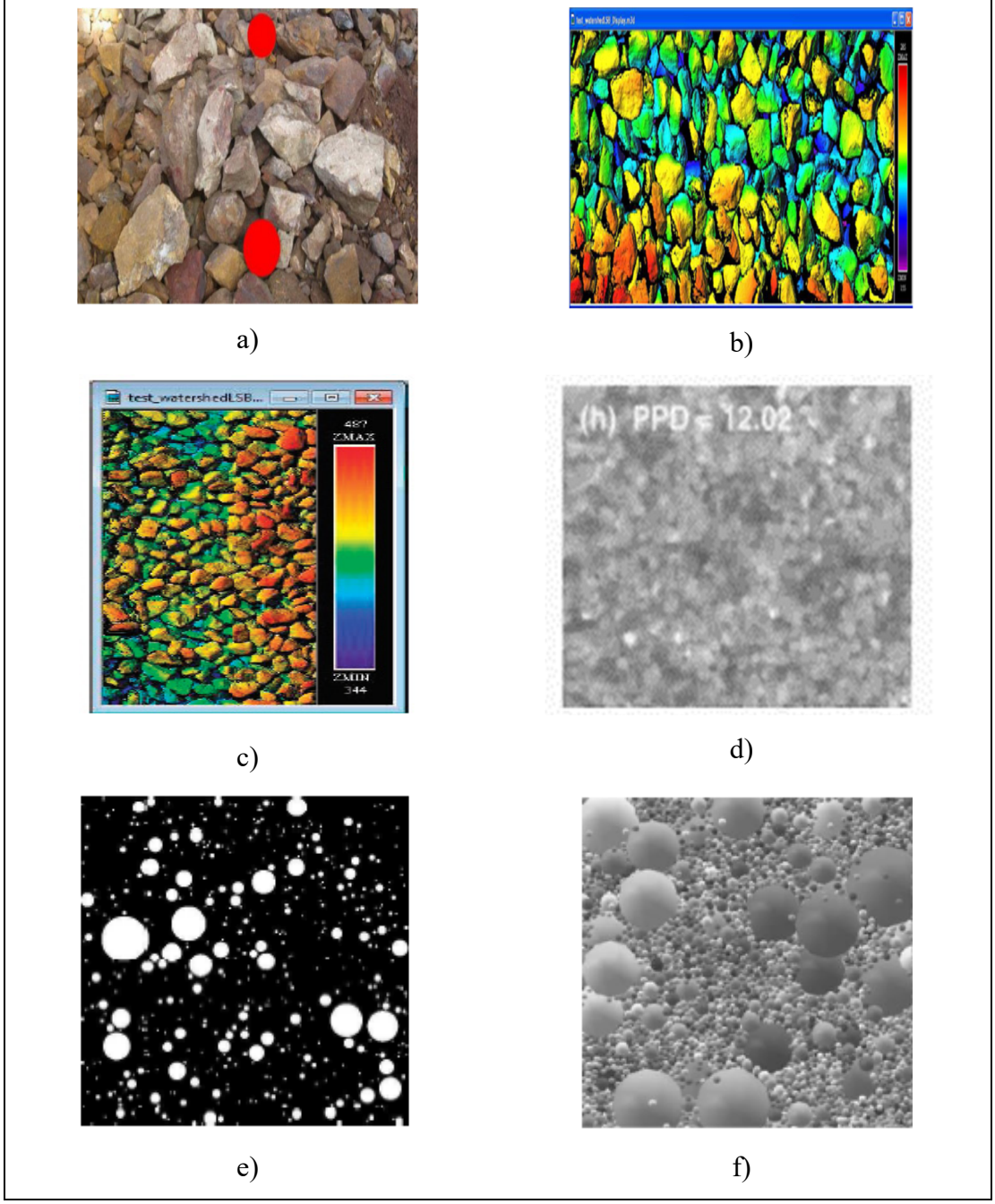


Figure 1.15 Exemples d'images utilisées par a) Yaghoobi et al. (2019) b) Thurley (2011) c) Andersson (2010) d) Ghalib et al. (1998) e) Ferrari et al. (2008) et f) Pirnia et al. (2018)

1.6 Plateforme de développement de jeux vidéo

Dans le but de représenter au mieux la réalité ce travail s'est tourné vers les logiciels de jeux. L'idée consiste à reproduire une simulation où des particules qui tiennent plus compte des paramètres naturels comme l'ombre, une forme quelconque des grains etc, sont prises en photo. La meilleure option est la plateforme de développement de jeux. La base de données qu'on souhaite construire a pour objectif d'être assez complète en terme de nombre et de représentativité. De ce fait elle serait très utile et pourrait même servir pour tester des résultats d'anciens travaux.

Les logiciels de jeux ont le crédit d'être intéressants et très performants lorsqu'il s'agit de créer un monde ou un univers avec les caractéristiques qu'on veut. Ils apportent une liberté importante et une réalisation ou finition très précise de ce que l'on désire faire. En plus de cela les jeux vidéo sont réputés pour offrir une excellente qualité audio visuelle au point de rendre un rendu très proche de la réalité. Par ailleurs les jeux vidéo sont connus pour être puissants, l'utilisation des cartes graphiques telles Nvidia rend la performance assez bonne.

Finalement, contrairement aux logiciels qui permettent d'analyser la granulométrie ou le rendu est statique, les logiciels de jeux offrent plus de liberté et de choix lors de sa création, ce qui est très utile dans le cas d'une granulométrie par analyse d'images.

Pour résumer utiliser les logiciels de jeux va permettre de créer des granulométries dont la taille des particules n'a aucune contrainte tout en ayant une image de ces particules de très bonne qualité, on aura donc une base de données qui répond aux critères cités dans la partie précédente.

1.6.1 Principales plateformes de jeux disponibles

Plusieurs moteurs de jeux vidéo existent. Les moteurs 2D comme Game Maker, Game develop, et RPG maker, et des moteurs 3D tels Unreal Engine, Unity 3D et Cry Engine.

Comme le but de ce projet est de simuler des photographies de sols qui impliquent des particules 3D, la description suivante se focalise sur les moteurs 3D.

Chaque plateforme de jeux à ses caractéristiques, il y a :

- Unreal Engine est un logiciel de jeux qui est utilisé dans les projets de grandes envergures. Il est considéré comme le plus développé et le plus puissant des plateformes de jeux et nécessite un PC performant avec une carte graphique puissante;
- Cry Engine est un logiciel spécialisé dans les jeux de tir;
- Unity est la plateforme de jeux la plus répandue et la plus facile à utiliser.

Le choix d'utiliser Unity pour ce projet s'est fait pour sa simplicité : nous n'avons pas besoin d'un PC performant, ni d'une utilisation spécifique d'un mode de jeux. Il convient parfaitement à ce projet.

La plateforme de développement de jeux Unity est parmi les plus connues. Il a une communauté importante et un accès gratuit. Unity est aussi très accessible car il est possible de l'exporter sur les systèmes d'exploitation comme Windows, Mac et Android et sur les consoles de jeux tels PlayStation ou Xbox. Le langage de programmation de Unity est le Csharp et présente une dualité interface codage. De plus Unity se démarque de la concurrence par sa facilité d'utilisation, un asset store accessible et un nombre de fans importants.

On parlera plus en détails de Unity dans le chapitre 2 et de son interface et de la façon de l'utiliser. Le but de cette partie est d'introduire le logiciel afin de le lier à la granulométrie par analyse d'images.

1.6.2 Utilisation de Unity en géotechnique

Utiliser les plateformes de jeux en dehors des jeux vidéo est une idée qui a déjà été utilisée dans d'autres secteurs. Khalifa, Nguyen, Simoff, & Catchpoole (2015) ont utilisé Unity dans la visualisation bio médicale ou encore Smid (2017) en géoscience pour visualiser de zones

d'étude géologique à partir de jeux de données géospatiales. Comme on peut le constater l'utilisation de Unity en dehors des jeux vidéo est récente, soit à partir de 2015, et est peu développée.

En ce qui concerne la géotechnique les recherches sont moindres et sont récentes aussi. Chimos, Chavakis, Papacharalampou, Douligeris, & Kallergis (2014) ont utilisé Unity pour recréer un site qui se trouve en Grèce pour visualiser les aquifères présents (figure 1.16)

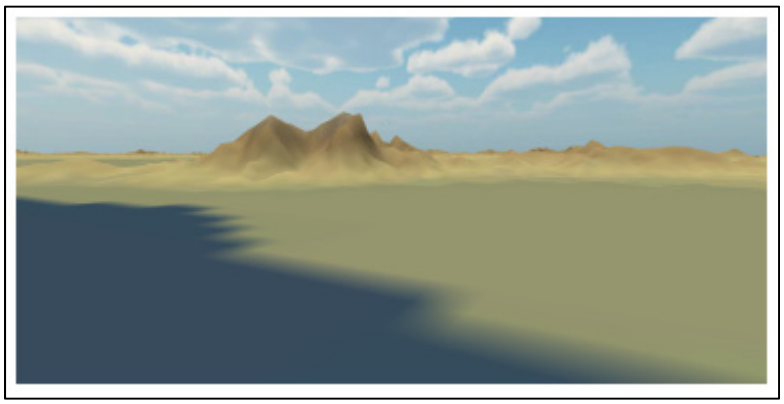


Figure 1.16 Recréation d'un site en Grèce avec Unity
Tirée de Kallergis et al. (2014)

Aussi, Orr, MacDonald, Iverson, & Hammond (2016) ont développé une vue à la première personne afin de visualiser l'intérieur d'une mine et de pouvoir en changer les paramètres (figure 1.17). Ceci leur a permis de développer une application exécutable.

Figure 1.17 Intérieur d'une mine
Tirée de Orr et al. (2016)

Un peu plus dans le domaine, l'utilisation de Unity pour recréer des roches est récente aussi et se fait rare. Principalement, Ondercin (2016) qui a réussi à démontrer qu'il était possible d'utiliser Unity pour simuler et modéliser un éboulement pour une ligne de train située au sud-ouest de Colombie-Britannique comme le montre la figure 1.18. Les résultats ont montré qu'il était possible d'atteindre une très bonne résolution d'un ordre de grandeur inférieur au mètre. Aussi, dans la continuité de ce travail, Sala (2018) a utilisé Unity pour modéliser un éboulement avec des roches avec un comportement mécanique plus réaliste que celles Ondercin (2016).

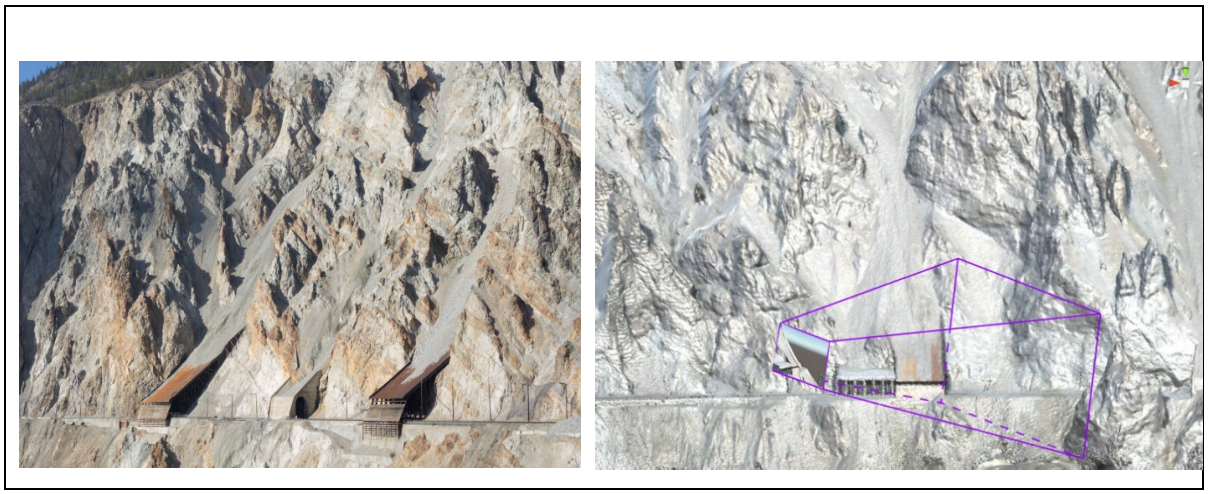


Figure 1.18 Simulation d'un l'éboulement grâce à Unity
Tirée de Ondercin (2016)

Il est à noter que ces travaux n'ont pu faire la simulation à travers Unity que par l'utilisation du lidar, photogrammétrie et Sfm (de l'anglais *structure from motion*) pour avoir une précision des lieux.

Blanco-Pons, Carrión-Ruiz, & Lerma (2016) ont eu recours à Unity pour recréer une roche en réalité augmentée (AR) et en réalité virtuelle (VR) dans le but d'améliorer la visualisation 3D des roches (figure 1.19).

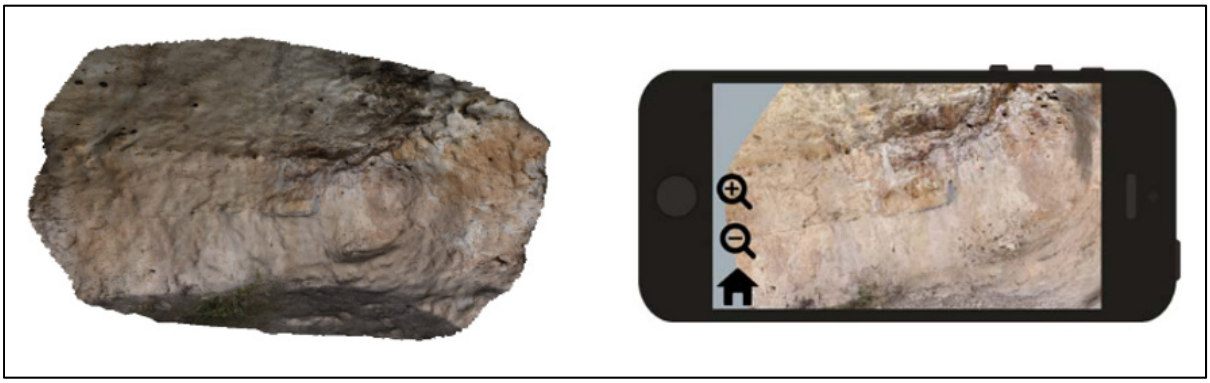


Figure 1.19 Modélisation d'une roche en VR et visualisation sur smartphone
Tirée de Blanco-Pons et al. (2016)

CHAPITRE 2

MÉTHODOLOGIE

La revue de littérature a permis de constater que les bases de données sont très importantes pour le développement de méthodes d'analyses granulométriques par techniques d'analyse d'images. Par contre, le nombre d'images dans les bases de données existantes est souvent limité. Le premier objectif du mémoire est de créer une base de données contenant un grand nombre d'images, comme celle de Pirnia et al. (2018) et Manashti et al. (2020), mais avec des images plus réalistes.

L'idée est de créer des images de matériaux granulaires virtuels dont la courbe granulométrique est connue. L'interface de programmation de jeu Unity sera utilisée pour préparer cette base de données. La première section de ce chapitre présente l'interface de Unity. La deuxième section décrit les granulométries et les spécimens virtuels qui seront créés (courbe granulométrique, masse, volume). La troisième section aborde la création des particules. On y trouve une description des critères qu'on souhaiter respecter, des particules ainsi que leurs propriétés. Finalement, la quatrième section décrit la création des spécimens dans Unity et son automatisation.

2.1 Présentation de Unity

Cette partie présente la plateforme Unity. Cette plateforme a été choisie parce qu'elle propose une version gratuite et parce qu'elle figure parmi les plus utilisées. Comme mentionné précédemment, cette plateforme est déjà utilisée pour certaines applications en recherche.

2.1.1 Interface

L'interface de Unity est divisée en cinq parties (figure 2.1) :

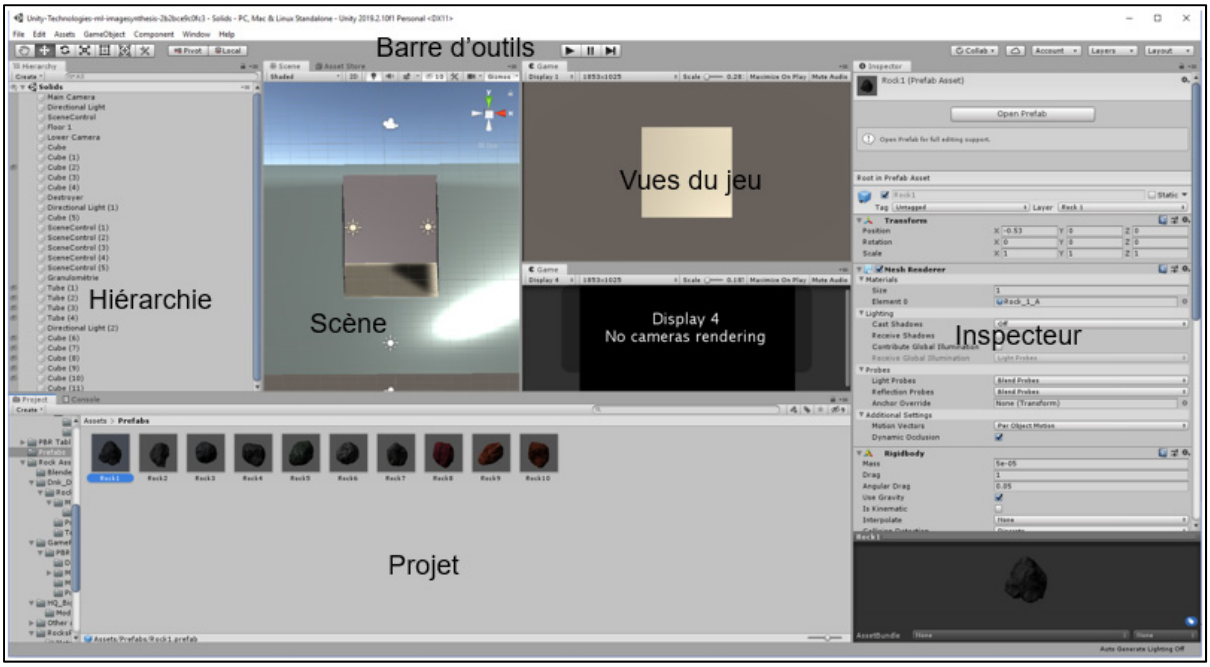


Figure 2.1 Interface de Unity

La scène : C'est l'interface interactive pour l'éditeur où se crée le jeu jusqu'au résultat final. Dans la scène, il est possible de choisir la position, l'environnement, les personnages, manipuler les objets, changer les caméras, etc.

La hiérarchie : Elle contient les objets qui apparaissent dans la scène. Ceux-ci peuvent être ordonnés par date d'ajout, l'option par défaut, ou aussi par ordre alphabétique. Le terme hiérarchie vient du fait que les objets peuvent contenir d'autres objets, on parle alors de lien de parenté ce qui crée une hiérarchie.

La vue du jeu : C'est une fenêtre qui permet de voir le rendu final du jeu, celui que les joueurs verront. La vue se base sur ce que choisit l'éditeur. Elle est contrôlée par les caméras qui ont été positionnées et paramétrées dans la scène.

Le projet : Cette fenêtre permet d'accéder aux dossiers et aux fichiers liés au projet. Ces fichiers contiennent toutes les caractéristiques de tous les objets qui seront utilisés dans le jeu (p. ex. caméras, particules et effets visuels).

L'inspecteur : Cette fenêtre permet de modifier les caractéristiques des objets, comme les matériaux, la texture, la masse, les collisions, etc. Cette fenêtre permet aussi de définir les caractéristiques des caméras. L'inspecteur permet d'assigner les scripts aux objets. L'utilisation de scripts sera développée en détails par la suite.

2.1.2 Caractéristiques des objets

Parmi les paramètres les plus importants qui peuvent être modifiés avec l'inspecteur, il y a :

Rigidbody **(corps rigide)** : Cette propriété permet de donner un comportement physique aux objets. Elle permet par exemple de fixer leur masse, d'appliquer la gravité et de spécifier une méthode pour la détection des collisions. Ainsi, la propriété de corps rigide permet de contrôler l'objet de manière physiquement réaliste.

Collider **(collisionneur)**: Cette propriété permet de définir la forme des objets pour la détection des collisions.

Mesh **(maillage)**: Cette propriété permet de générer le maillage nécessaire à la simulation.

Material **(matériau)**: Cette propriété permet d'ajouter une texture aux objets. Cette texture peut aller d'une simple couleur à une texture plus complexe.

2.1.3 Utilisation des scripts

Il est possible d'attacher aux objets des scripts. Les scripts permettent d'assigner des tâches aux objets. Par exemple, il est possible d'écrire un script pour faire disparaître un objet lorsqu'il touche le fond d'une boîte. De même, il est possible de créer un script qui enregistre certaines propriétés des particules lorsqu'elles touchent le fond de la boîte. Les scripts sont codés en C#, un langage de programmation orienté objet. Un objet peut avoir plusieurs scripts.

2.1.4 Caméra

La caméra représente le point de vue de la scène et des joueurs. Elle offre une vue 3D (figure 2.2a), des vues 2D, une vue orthogonale (figure 2.2b), et une vue en perspective (figure 2.2c). Les caméras ont d'autres caractéristiques. Par exemple, elles peuvent ne montrer que les objets appartenant à la même couche, ou encore changer la profondeur, la taille ou la résolution. Il est également possible d'ajouter plusieurs caméras. Dans le cadre de ce projet, cette possibilité permet de générer des images du même spécimen, mais de différents points de vue.

2.1.5 Éclairage

Unity donne la possibilité de contrôler l'éclairage, un élément très important lors de la création des jeux. En complément aux caractéristiques des objets qui donnent l'allure de la scène, la lumière affecte la couleur et l'éclairage de celle-ci. En effet il est possible de changer la lumière en n'importe quelle couleur, ou même de changer l'incidence de façon à produire l'ombre souhaitée sur les objets. Ceci permet d'obtenir le rendu le plus réel possible.

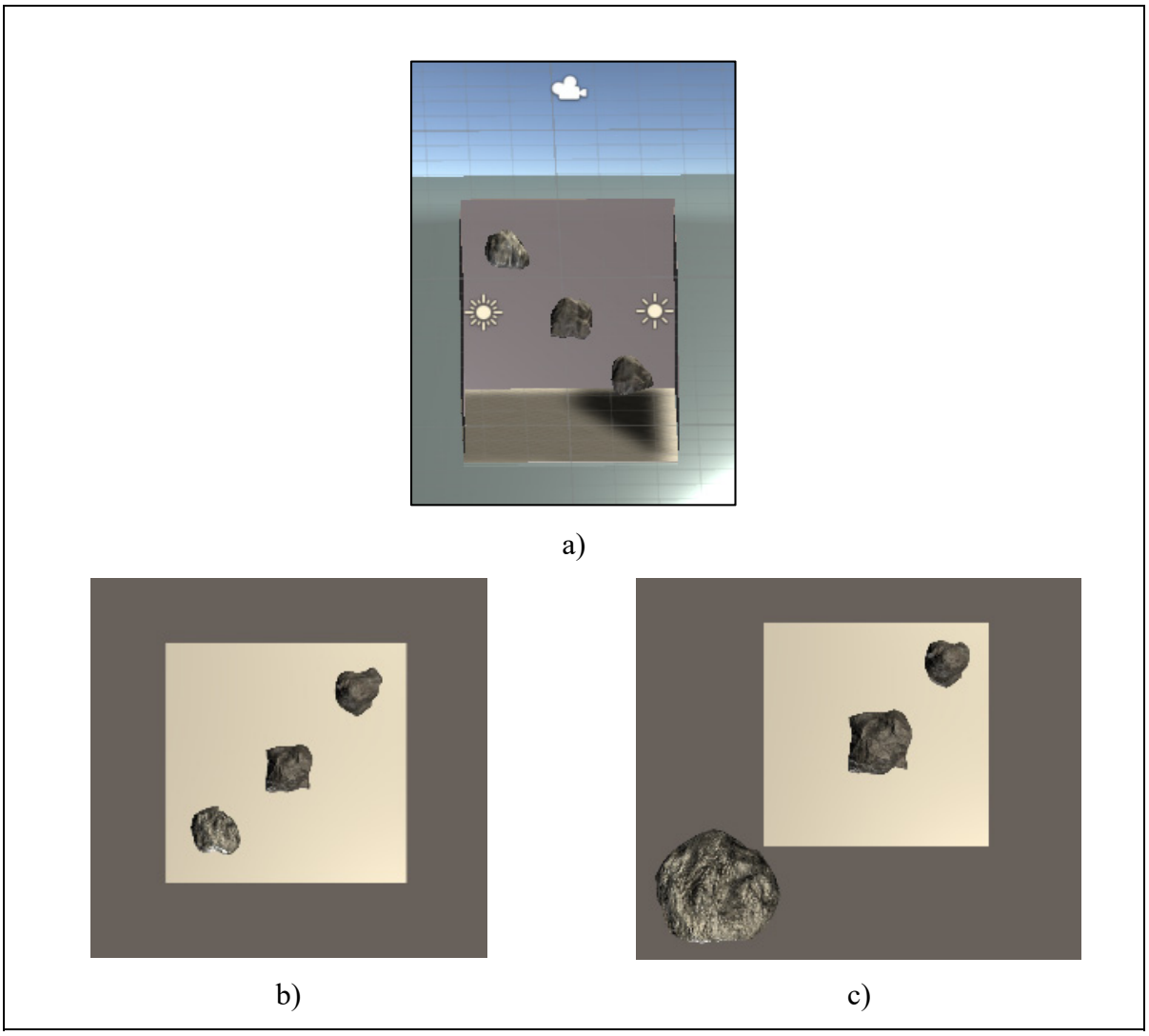


Figure 2.2 Exemples de a) vue de profil b) vue orthogonale et c) vue en perspective pour une même configuration

2.1.6 Asset Store

L'Asset Store fonctionne selon le même principe que Google Play pour Android ou l'App Store pour l'iOS. Il permet de télécharger et d'importer des objets, des textures, de l'éclairage, des scripts, voire même des scènes complètes dans le but de les utiliser directement sur Unity. Les *assets* (éléments) sont gratuits ou payants. Ils peuvent être exportés vers différentes versions de l'interface Unity.

2.1.7 Notion d'échelle

La notion d'échelle sur Unity est très importante. Par défaut les tailles sont représentées en mètres et les masses en kilogrammes. Néanmoins il est possible de ne pas se référer aux unités physiques précédemment citées mais, plutôt à ce qui est appelé la taille Unity ou plutôt la taille qu'on utilisera dans Unity. De ce fait il est possible de créer des particules dont la taille est de l'ordre du millimètre et la masse en gramme à condition de garder le rapport qui dans ce cas est la masse volumique. Dans ce projet, le millimètre sera la taille Unity pour les longueurs et le gramme sera la taille Unity pour les masses. Ainsi il est possible de créer des objets ou des particules de la taille qu'on veut à condition de garder la taille Unity en mémoire et de respecter le rapport avec les autres variables.

2.1.8 Le réalisme de Unity à travers un exemple

Comme expliqué précédemment, l'intérêt d'utiliser Unity pour créer des images de particules granulaires est de tirer profit du réalisme qu'offre ce moteur de jeu vidéo. Cette section donne quelques exemples des paramètres qui peuvent être modifiés pour changer l'apparence des particules granulaires.

La figure 2.3a présente tout d'abord la particule originale. Cette particule subit un changement de texture, en l'occurrence ici le changement de couleur en rouge (figure 2.3b). La figure 2.3c montre ensuite un changement de luminosité qui rend la particule plus sombre. La figure 2.3d montre un changement d'angle de caméra. La figure 2.3e met l'accent sur la possibilité de dupliquer les particules, de visualiser leur maillage qui est dans ce cas est convexe, ainsi que de l'ombre sur la particule du bas. Finalement la figure 2.3f montre que l'on peut changer la taille des particules. Dans ce cas, la particule de gauche a une taille deux fois plus grande que la particule de droite. Toutes ces transformations peuvent être réalisées manuellement dans l'interface de Unity ou à l'aide d'un script.

Comme on peut le constater, Unity permet de changer les paramètres liés aux caméras, à l'éclairage (luminosité, direction et ombre) et aux particules (taille et texture) afin de recréer un environnement réel et ainsi avoir une base de données réaliste. La suite du chapitre présente la démarche utilisée dans le but de créer les particules et les images.

2.1 Choix des caractéristiques des spécimens

Cette section se focalise sur les principales caractéristiques des spécimens virtuels qui ont été préparés avec Unity.

2.1.1 Système d'axes

Le système d'axe est défini dans le but de situer le projet. Les axes x et z forment le plan horizontal, et l'axe y représente l'élévation. La gravité agit selon l'axe y. C'est l'orientation par défaut dans Unity.

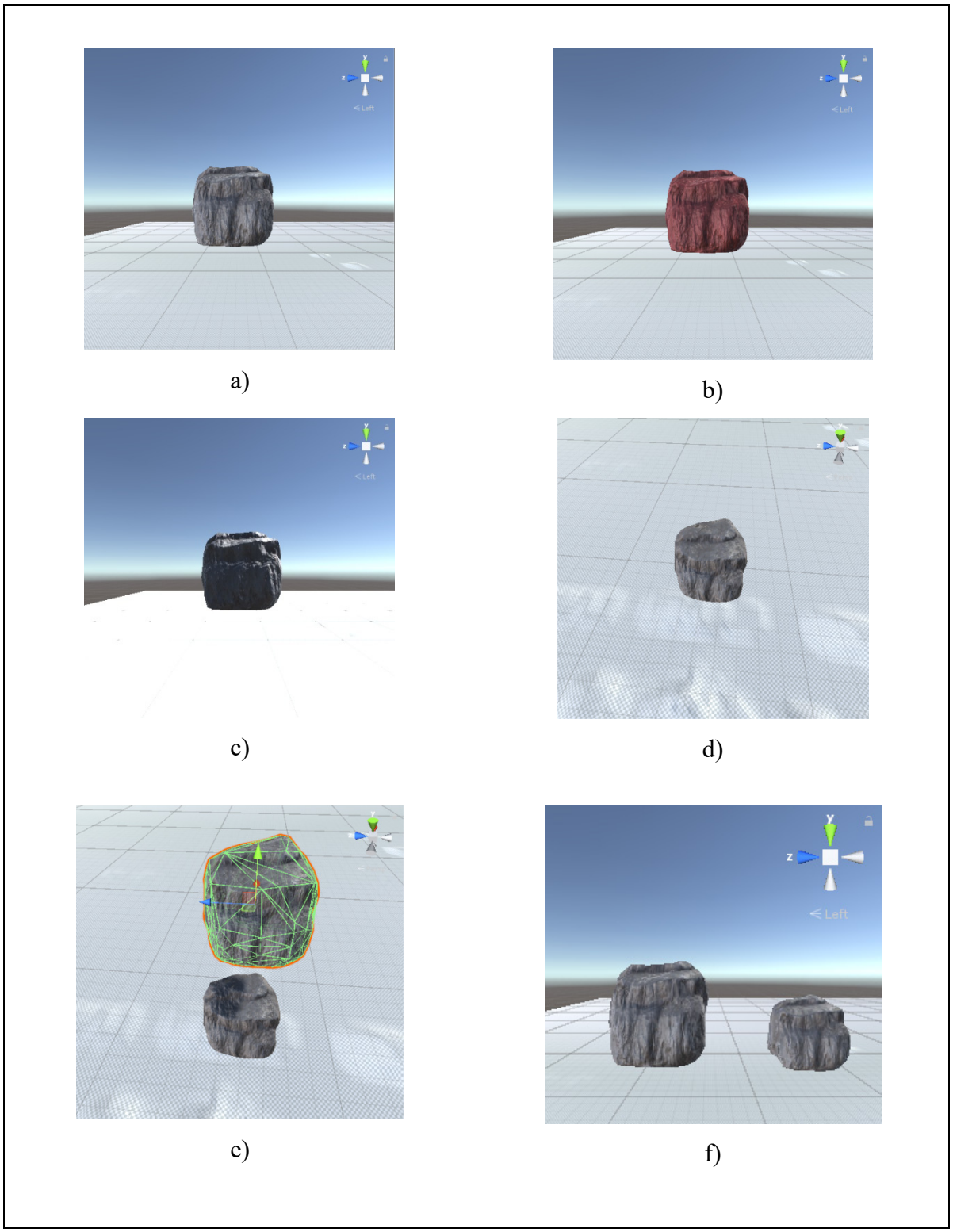


Figure 2.3 a) Particule de base, b) Exemple de changement de texture, c) Exemple de changement d'éclairage, d) Exemple de changement d'angle, e) Maillage convexe de la particule et f) Exemple de changement de taille

2.1.2 Choix des courbes granulométriques

Les paramètres des spécimens ont principalement été choisis de manière à produire des images ayant la même échelle et les mêmes tailles de particules que Pirnia et al. (2018) et Manashti et al. (2020). Ce choix facilite la comparaison des deux bases de données. Les tamis suivants ont été utilisés : 75, 106, 150, 250, 425, 710 et 1180 µm (ASTM International 2017a). Dans le système de classification USCS (ASTM International 2017a) le tamis 75 µm correspond à la limite inférieure du sable. La limite supérieure, la frontière entre le sable et le gravier, est fixée à 4750 µm. Le tamis de 1180 µm a plutôt été utilisé comme limite supérieure pour restreindre le nombre de particules. Lorsque la taille maximale des particules augmente par un facteur *t*, la taille des spécimens doit aussi être augmentée par un facteur *t*. Comme le nombre de particules est proportionnel au volume, l'augmentation de la taille du spécimen par un facteur *t* produit une augmentation du nombre de particules les plus fines par un facteur t^3. Par exemple, l'augmentation de la taille maximale de 1180 à 4750 µm augmenterait le nombre de particules par un facteur 65 pour la courbe granulométrique la plus fine.

Les courbes granulométriques ont été créées à partir de l'ensemble des combinaisons possibles de pourcentages passants aux tamis 75, 106, 150, 250, 425, 710 et 1180 µm par incrément de 10 %. Le choix de l'incrément affecte la valeur du nombre d'images. Le tableau 2.1 donne les pourcentages passants pour les sept premières images en exemple. Pirnia et al. (2018) et Manashti et al. (2020) ont utilisé des incréments de 5 %. Plus l'incrément est faible, plus le nombre d'images est grand. Le tableau 2.2 présente le nombre d'images à générer pour différents incréments. L'incrément utilisé dans ce projet produit 3003 images au lieu des 53 130 images produites par Pirnia et al. (2018) et Manashti et al. (2020). Le temps de calcul pour chaque image varie de quelques secondes à un maximum de deux heures en fonction du nombre de particules.

Tableau 2.1 Exemples de pourcentages passants pour une granulométrie par incréments de 10 %

1180 µm	710 µm	425 µm	250 µm	150 µm	106 µm	75 µm
100	0	0	0	0	0	0
100	10	0	0	0	0	0
100	10	10	0	0	0	0
100	10	10	10	0	0	0
100	10	10	10	10	0	0
100	10	10	10	10	10	0
100	20	0	0	0	0	0

Tableau 2.2 Nombre d'images générées en fonction de l'intervalle des pourcentages passants

Intervalle	5%	10%	20%	25%
Nombre d'images	53 130	3003	252	126

2.1.3 Volume et masse du spécimen

Dans le but de contenir les particules et de prendre des images à partir du haut et du bas, le spécimen est mis en place dans une boîte de 5 mm × 5 mm × 5 mm (figure 2.4). La boîte est constituée de cinq façades infiniment minces. La boîte n'a pas de couvert. Les particules tombent dans la boîte du haut par gravité. L'image du bas est prise à travers la façade du bas qui est transparente. Afin de minimiser les pertes, une inclinaison de 5 ° des parois latérales force les particules à s'écouler vers le fond de la boîte. Les parois sont minces pour que les particules ne restent pas accrochées au-dessus des façades au moment où elles tombent dans la boîte.

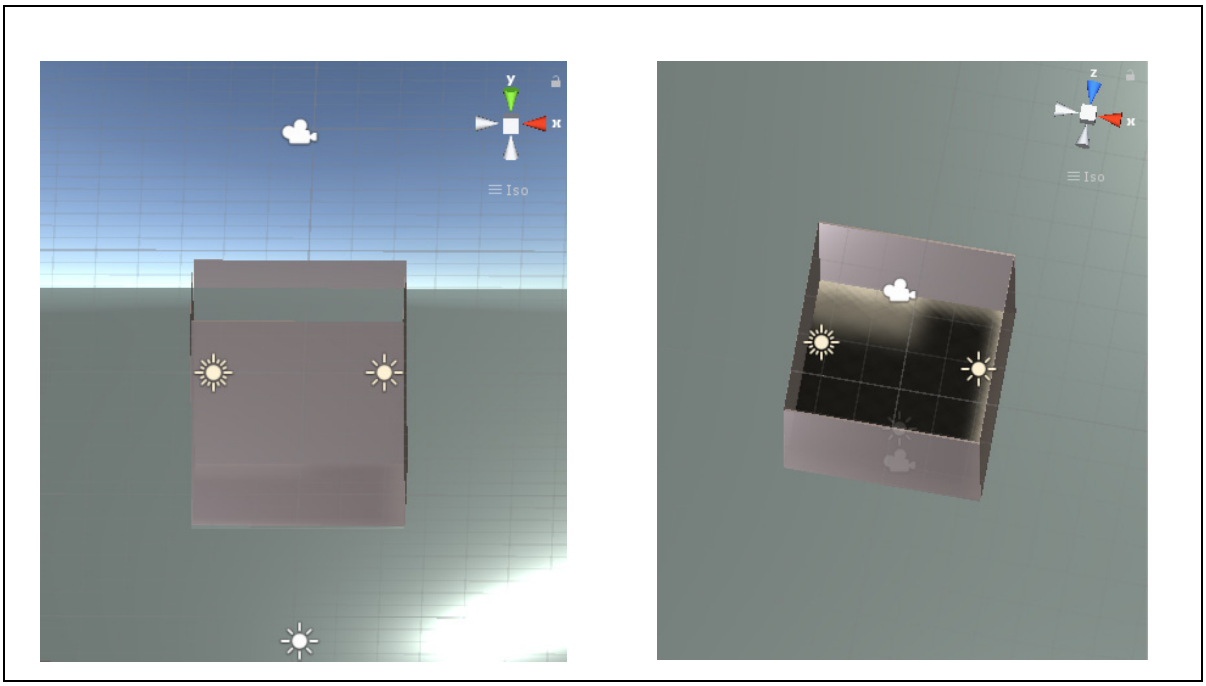


Figure 2.4 Boîte contenant les spécimens

Cette boîte va contenir des particules dont la masse totale des solides est constante et vaut 50 mg (Pirnia et al. 2018; Manashti et al. 2020). La masse de chaque particule est définie de la façon suivante :

$$m = \rho_s \, v \tag{2.1}$$

où

- m est la masse de chaque particule;
- ρ_s est masse volumique des solides (2,5 g/cm^3);
- v est le volume de chaque particule.

La masse de chaque particule, la masse totale des spécimens (50 mg) et le pourcentage de la masse retenue entre chaque tamis sont utilisés à la section 2.4.4 pour calculer le nombre de particules entre chaque tamis. Les particules ne sont pas sphériques. Elles sont définies plus loin par leurs tailles selon les axes x, y et z. Le volume de chaque particule est déterminé comme celui d'une sphère en prenant comme rayon la moyenne des rayons dans les trois directions :

$$v = \frac{4}{3}\pi\left(\frac{dx + dy + dz}{3}\right)^3 \tag{2.2}$$

où d_x, d_y et d_z représentent la taille des particules respectivement suivant les axe x, y et z. Le tableau 2.3 présente des exemples de masses de particules en fonction du tamis sur lequel les particules sont retenues. La masse est calculée en supposant que la taille correspond à la moyenne de l'ouverture du tamis sur lequel la particule est retenue et celle du tamis précédent.

Tableau 2.3 Tamis et masse moyenne des particules

Taille du tamis (µm)	***m* (g)**
75	$5{,}52\times10^{-7}$
106	$1{,}56\times10^{-6}$
150	$4{,}42\times10^{-6}$
250	$2{,}04\times10^{-5}$
425	$1{,}01\times10^{-4}$
710	$4{,}68\times10^{-4}$
1180	$2{,}15\times10^{-3}$

2.2 Création des spécimens dans Unity

2.2.1 Choix des particules

Le choix des particules doit se faire de façon minutieuse. Il doit satisfaire les conditions suivantes :

- avoir des particules très réalistes;
- éviter les sphères parfaites comme dans les travaux de Pirnia et al. (2018) et Ferrari et al. (2008);
- avoir une diversité au niveau des formes de particules;

- avoir un contrôle sur la taille suivant les trois axes;
- utiliser différentes couleurs et textures pour la surface des particules.

Trois méthodes ont été considérées pour créer les particules : les créer directement avec Unity, les créer avec Blender, un logiciel de design pour Unity, et les importer directement de l'Asset Store. Les deux premières options nécessitent beaucoup plus de temps, car il faut ajouter les détails au niveau du maillage et du collisionneur pour les particules (les détails figurent au niveau de la section 2.3.2). Les éléments proposés dans l'Asset Store sont complets (forme + maillage + collisionneur).

Les particules ont donc été importées de l'Asset Store. Un ensemble de dix particules a été choisi. Ce nombre est arbitraire, mais il permet d'avoir une bonne diversité dans la forme des particules. Les éléments qui ont permis d'obtenir les dix particules sont présentés au tableau 2.4. Ces éléments ont été choisis en fonction de leur maillage, leur collisionneur, leur taille et leur allure. De légères modifications ont été appliquées à certaines particules, par exemple pour changer leur allure, leur maillage ou encore la surface des particules qui était parfois trop lisse.

Tableau 2.4 Noms des éléments choisis et références

Nom	**Références**
5 Rock Package	https://assetstore.unity.com/packages/3d/environments/5-rocks-package-133702#description
Realistic Rock	https://assetstore.unity.com/packages/3d/realistic-rock-51251
Rocks	https://assetstore.unity.com/packages/3d/props/exterior/rocks-604
Sandy Rock Materials	https://assetstore.unity.com/packages/2d/textures-materials/nature/sandy-rock-materials-137089
HQ Rock Pack Free	https://assetstore.unity.com/packages/3d/props/exterior/hq-rock-pack-free-83388

Les particules utilisées sont présentées à la figure 2.5. Dans la suite, les objets seront parfois nommés avec les noms définis dans Unity (Rock1, Rock2, etc.).

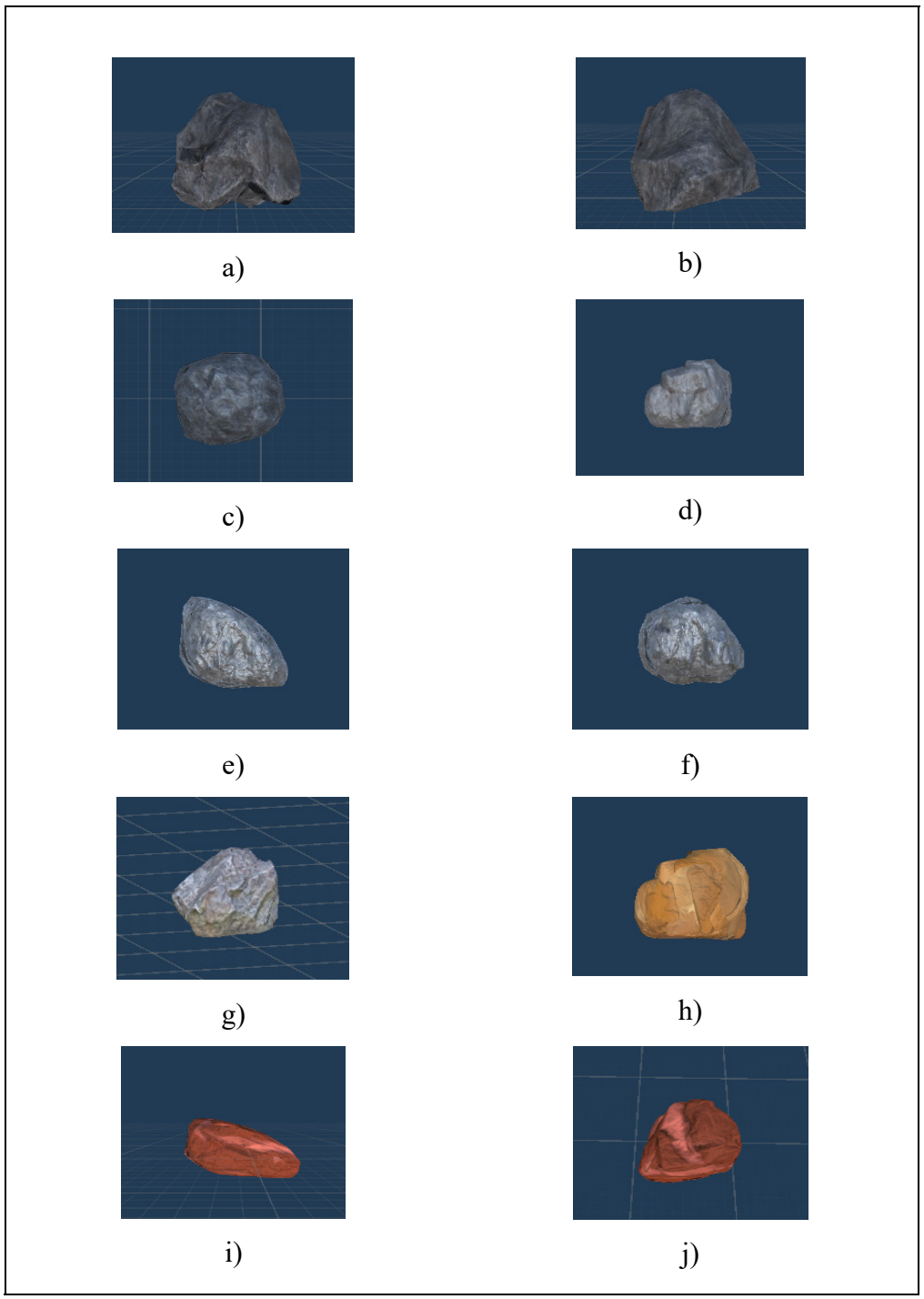


Figure 2.5 a) Rock1, b) Rock2, c) Rock3, d) Rock4, e) Rock5, f) Rock6, g) Rock7, h) Rock8, i) Rock9, j) Rock10

2.2.2 Propriétés des particules

Les dix particules ont le même contenu. L'exemple de la particule Rock1 sera utilisé dans cette section pour éviter les répétitions (figure 2.6 a). Les caractéristiques suivantes ont été ajoutées aux particules issues de l'Asset Store :

Collisionneur : Afin que les particules s'éloignent suite aux collisions avec les autres particules, celles-ci doivent être munies d'un collisionneur ou de détection de collision. Avec Unity, il existe plusieurs collisionneurs tels le collisionneur boîte (*box collider)* (figure 2.6 b), le collisionneur sphère (le *sphere collider)* (figure 2.6 c) ou le maillage collisionneur (*mesh collider)* (figure 2.6 d).

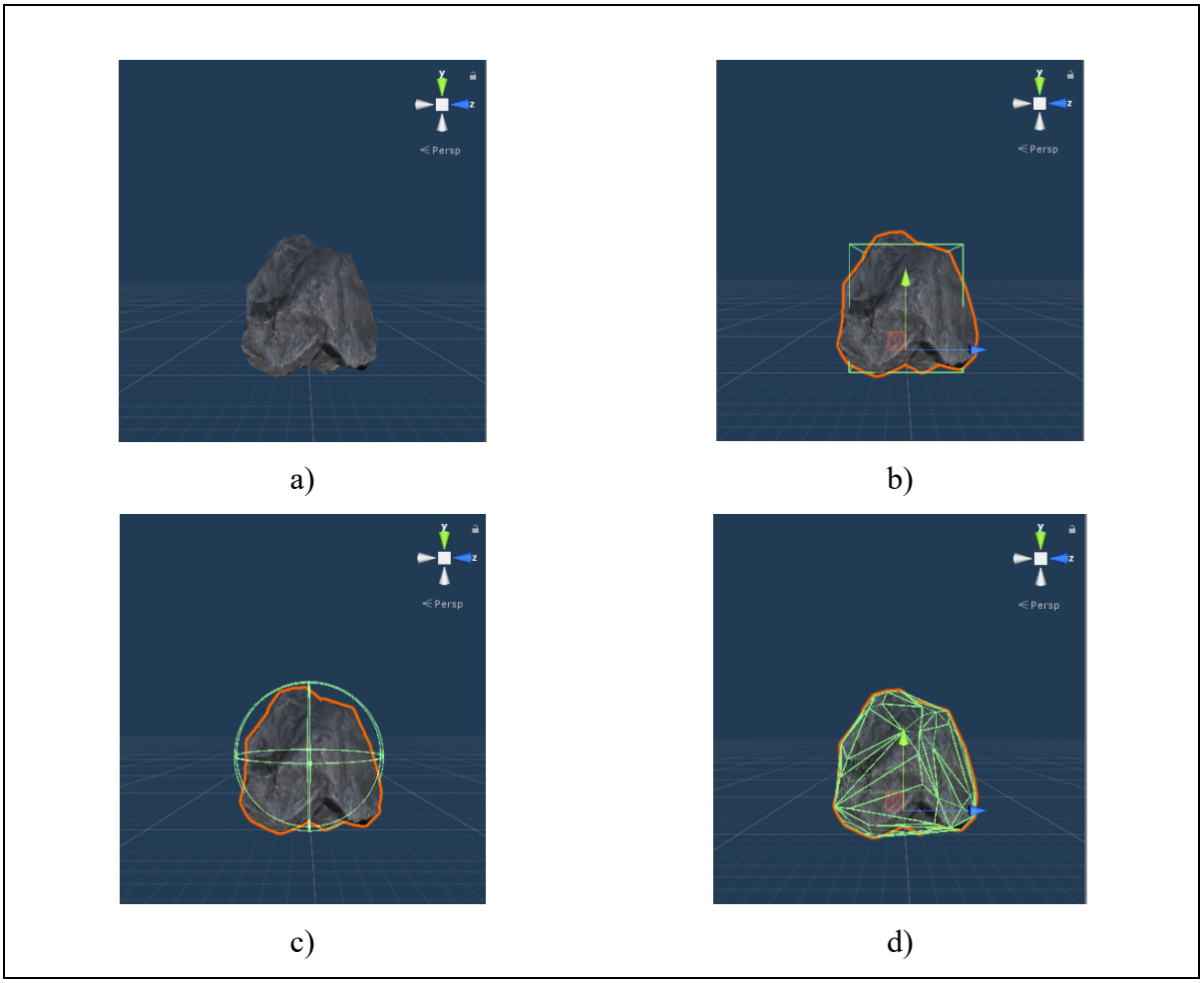


Figure 2.6 a) Rock1, b) Rock1 avec *box collider*, c) Rock1 avec *sphere collider*, d) Rock1 avec *mesh collider*

Il est clair d'après la figure 2.6 qu'un collisionneur avec maillage représente au mieux la particule. Néanmoins, cette méthode augmente le temps de calcul, particulièrement avec un grand nombre de particules. Étant donné que ce projet a pour but de représenter au mieux la réalité, un collisionneur avec maillage a été utilisé.

Maillage polygone ou *mesh* renderer : caractéristique qui donne l'allure de la particule et permet de jouer sur les caractéristiques d'ombre et de lumière (figure 2.7).

Corps rigide ou *rigidbody* : caractéristique très importante permettant d'offrir le côté réel de la chose. Elle permet d'ajouter aux roches la masse, le frottement et de jouer sur le mouvement et les collisions (figure 2.7).

Scripts : élément qu'on rajoute aux particules leur permettant de réaliser les tâches souhaitées (figure 2.7). Cette partie sera plus détaillée par la suite.

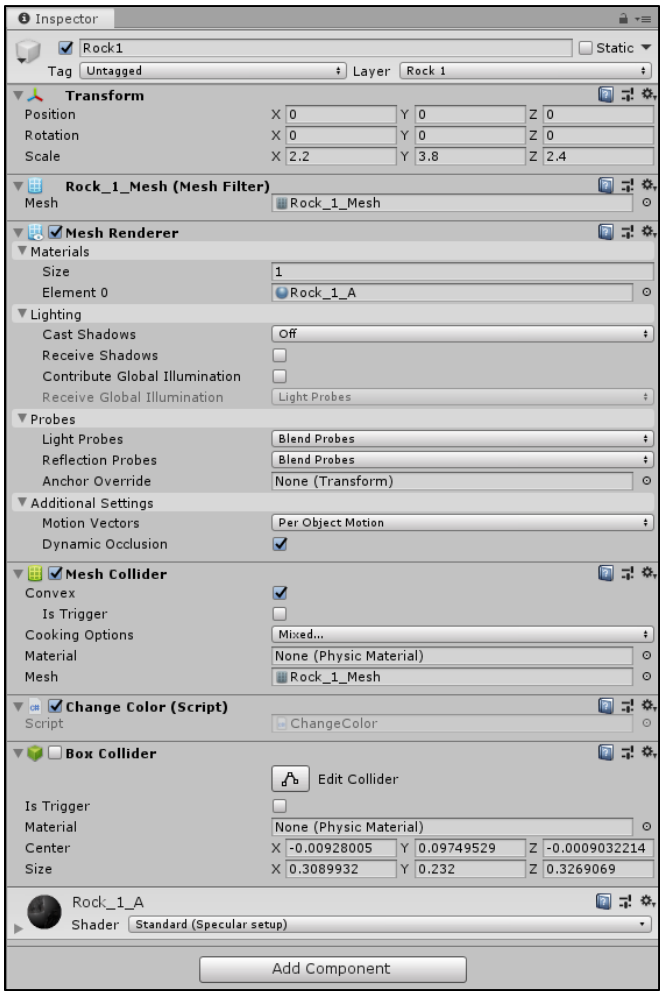


Figure 2.7 Exemple du contenu de Rock1

2.3 Création des images

Une fois les particules choisies, la granulométrie bien définie et la boîte construite, il est important que le tout soit lié et surtout fonctionne de manière à avoir des images des particules du passant aux tamis 75, 106, 150, 250, 425, 710 et 1180 µm ainsi que leur taille. Pour ce faire on a eu recours aux scripts (Annexe II). Dans le cas de ce projet ils sont au nombre de sept. Le script qui génère les particules a été divisé en deux (2.4.1 et 2.4.6).

2.3.1 Générer les particules

Il s'agit ici du script (Annexe II-1) de base pour ce projet. Dans ce but cette étape a été conçue de la façon suivante :

- utiliser aléatoirement les dix particules pour générer un nombre de particules total égal à une valeur choisie. Cette valeur a été choisie en fonction de la granulométrie et sera expliquée en détails dans la partie 2.3.4;
- générer chaque particule aléatoirement au niveau d'une position qui fait en sorte qu'elle tombe parfaitement au niveau de la boîte (figure 2.8). Cette position varie d'une longueur d'un peu moins de 5 mm suivant l'axe x et z (axe horizontal) et de 10 mm sur l'axe y (axe de profondeur);

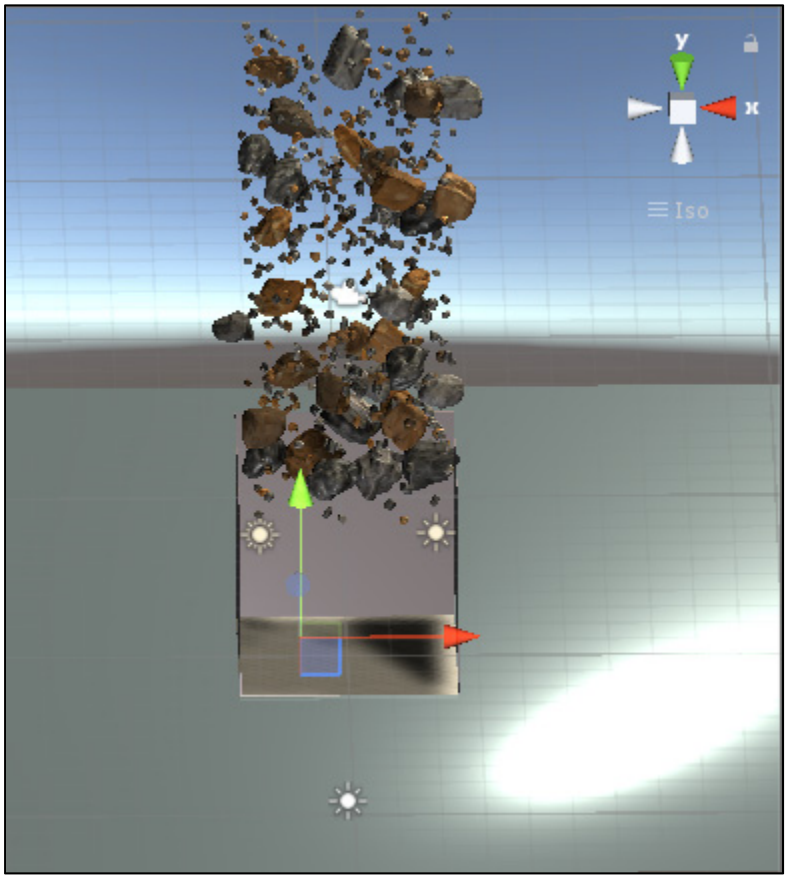


Figure 2.8 Particules générées qui vont dans la boîte

- donner à chaque particule a une rotation aléatoire (visible au niveau de la figure 2.8);
- donner à chaque particule a une taille aléatoire qui varie entre un minimum et un maximum défini par la taille des tamis. Ils prendront donc les valeurs suivantes 75, 106, 150, 250, 425, 710 et 1180 µm;
- refaire la procédure pour un temps ou un nombre d'images (*frame*) donné. Étant donné que le nombre de particules change et que le temps pour les générer aussi on a choisi de jouer sur le nombre d'images (*frame*) pour avoir une donnée intrinsèque.

2.3.2 Accélérer le modèle

Vu que le nombre de particules générées ne va cesser d'augmenter pour atteindre de très grande valeur (voir la partie 2.3.4) il est important que le code écrit (Annexe II-2) soit plus rapide avec des commandes qui ne demandent pas beaucoup en mémoire. Dans ce but l'idée a été de faire les pratiques suivantes :

- éviter d'utiliser les boucles itératives, quand c'était possible, pour générer les particules et on a utilisé des fonctions propres à Unity;
- utiliser la fonction pool : Les fonctions classiques de création et de destruction des objets nécessitent de la place en mémoire et de l'effort de la part du microprocesseur. Afin d'éviter cela la fonction pool a été utilisée comme alternative qui permet de pré créer les objets souhaités juste avant de les utiliser. Cette technique permet de gagner de la mémoire ce qui est très utile dans le cas où le nombre de particules générées est très grand;
- utiliser les préfabriqués : ce système qui est présent dans Unity permet de créer, de configurer et d'enregistrer les objets d'une manière plus facile.

Ainsi pour accélérer le modèle, l'idée s'est axée sur des fonctions et des systèmes moins demandant en mémoire permettant d'augmenter la vitesse de la simulation.

2.3.3 Changer la masse

Comme mentionné dans la partie 2.3.1, chaque particule a une masse qui se calcule de façon bien spécifique. Un script a été fait dans ce but et afin de lier la masse aux particules (Annexe II-3). De ce fait à chaque fois que des particules ont été générées la masse change automatiquement et ce, quel que soit le nombre.

2.3.4 Affecter la granulométrie et nombre de particules

Dans la continuité de ce qui a été écrit dans la partie 2.3.3, ce script lie les 3003 différentes granulométries aux particules solides afin qu'à chaque fois que des roches soient générées la granulométrie change.

D'ailleurs ce script va même un peu plus loin et donne le nombre de particules à générer pour avoir la granulométrie qu'il faut (Annexe II-4).

Il est à rappeler que la granulométrie représente à la base une distribution statistique des différentes classes de grains, il est donc possible d'en déduire le nombre de particule comme le montre l'équation suivante :

$$n = \frac{\%passant * Masse\_tot}{100 * \rho * Volume\_part} \tag{2.3}$$

où :

- n : le nombre de particules;
- *%passant* : Le pourcentage passant de la granulométrie;
- *Masse_tot* =50 mg masse totale de l'échantillon.

Dans le but d'avoir une idée du nombre de particules, une estimation a été faite en prenant la moyenne de deux tamis successifs. Ainsi et compte tenu des tailles suivantes 75, 106, 150, 250, 425, 710 et 1180 µm on obtient des valeurs du nombre de particules n qui varient de 5 à

51533 par tamis. Il est à noter qu'on a pris deux tamis successifs dans le but d'avoir une estimation du nombre n. Les diamètres des particules qui ont été générées ont, quant à eux, tous des tailles intermédiaires par rapport aux tamis choisis.

Les valeurs que peuvent prendre le nombre de particules n varie en fonction de la taille des particules et le pourcentage passant et on peut émettre les remarques suivantes :

- la première image contient 100 % de passant aux tamis 710-1180 µm pour un nombre de particules égal à 45. Cette valeur de n est le minimum que peut avoir le nombre de particules total;
- la dernière image n (image 3003) contient 100% de passant aux tamis 75-106 µm pour un nombre de particules égal à 51533. Cette valeur ne constitue pas le nombre maximal;
- les valeurs du pourcentage passant varient de 0 à 100 % par incrément de 10;
- le reste des images est une combinaison de tous ces pourcentages. Le nombre de particules pour ces images est de loin plus élevé que 51533. Sur certaines images il peut atteindre le triple.

2.3.5 Donner une couleur réaliste

Au début du projet chaque particule avait une couleur différente. Le rendu était le suivant (figure 2.10). Par la suite il s'est avéré que cette idée était un bon moyen de repérer les particules mais ne reflétait en aucun cas la réalité. On a donc d'une part enlevé les couleurs et d'une autre rajouté des lignes de code permettant d'avoir des couleurs plus réalistes (Annexe II-5).

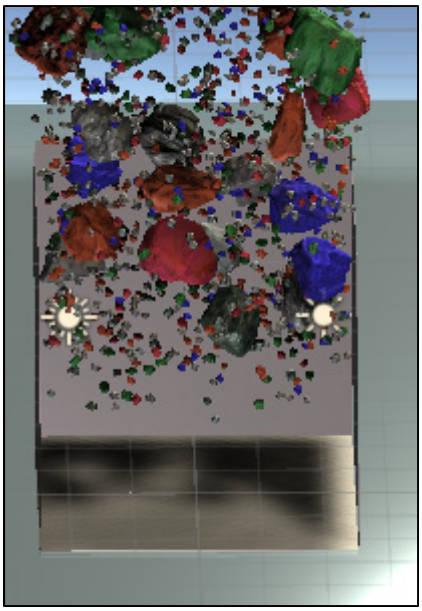


Figure 2.9 Exemple de particules colorées

Pour que cette opération se fasse un travail a été fait sur les caractéristiques des couleurs. D'ordre général les couleurs sont un mélange des trois couleurs primaires qui sont le rouge le vert et le bleu et sont codés de 0 à 255 dans l'espace RVB (pour Rouge Vert Bleu). Pour avoir des roches avec une couleur plus réaliste, on a simplement fait la moyenne de ces trois couleurs suivant l'équation suivante :

$$Couleur\_Uniforme = (Couleur\_Rouge + Couleur\_Verte + Couleur\_Bleue)/3 \qquad (2.4)$$

En réalité la moyenne des couleurs permet de donner différents niveaux de gris. Plus cette valeur est élevée plus le gris sera foncé et plus cette valeur est faible plus le gris sera clair. En appliquant cette formule le résultat attendu était que toutes les particules aient différents niveaux de gris, mais ça n'a pas était le cas pour les roches 8,9 et 10. Ceci s'explique par le fait que ces roches ont une propriété permettant de leur donner des couleurs proches de celles du sable. Ces couleurs sont plus marrons, et vont dominer les couleurs grises par superposition. On a donc laissé cette configuration dans la suite.

Ainsi le but de cette étape est d'uniformiser la couleur des particules et avoir des particules qui ressemblent à la réalité.

2.3.6 Prendre une photo du haut

Il s'agit ici du but du projet: prendre des photos des particules granulaires. La partie du script se trouve en continuité de l'Annexe II-1. Le principe a été simple et a les caractéristiques suivantes (figure 2.10) :

- la caméra est éloignée de presque 9 mm de façon à avoir une vue d'ensemble;
- la caméra est dirigée pleinement vers le bas soit avec un angle de 90 ° avec le plan horizontal (x,z);
- l'image est prise par vue orthographique et non de perspective;
- l'image est prise avec une dimension de 1853×1023;
- deux sources lumineuses permettent d'éclairer l'intérieur de la boîte pour une meilleure visibilité.

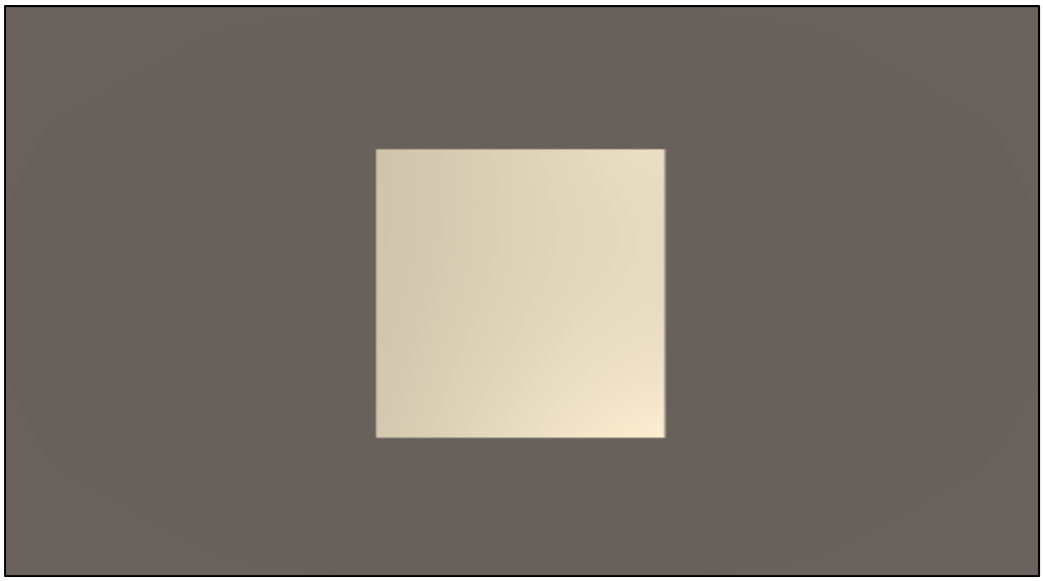


Figure 2.10 Caractéristique de la vue

Par la suite :

- une photo est prise à chaque fois que les particules sont générées et sont au fond de la boîte;
- un nombre d'images (*frame*) à partir duquel en prend les photos est défini. Le nombre de frame choisi prend en compte le temps qu'il faut pour générer les différentes granulométries et la puissance de l'ordinateur.

Il est à noter que le choix du champ de vision a été fait dans le but d'avoir un petit carré vers la fin contenant toutes les particules afin de les exploiter plus facilement dans un réseau de neurones (On expliquera les détails au niveau de la partie 3.3).

Une particularité à la caméra a même été rajoutée en donnant une image de profondeur. Ainsi la caméra va prendre deux images simultanément : une normale et une de profondeur. La caméra de profondeur se rajoute en important des assets et a été configurée dans le cadre de ce projet de façon à ce que :

- la profondeur 0 soit celle de la caméra;
- la profondeur maximale soit celle du fond de la boîte;
- plus on se rapproche de la caméra plus les objets apparaissent avec un teint gris plus foncé;
- plus on se rapproche de la base de la boîte plus les objets apparaissent avec un teint gris plus clair.

Il est à noter que d'autres particularités à la caméra auraient pu être rajoutées comme par exemple, que chaque objet apparaisse d'une certaine couleur, mais on a juste gardé la caméra de profondeur en plus de la caméra de base afin d'éviter de ralentir la simulation.

De cette manière en répétant le processus à chaque fois que les particules sont générées, une image normale et une de profondeur sont obtenues comme le montre la figure 2.11. Dans ce cas il s'agit de l'image 6 qui a la granulométrie suivante (Tableau 2.5).

On parlera plus amplement des autres images ainsi que leur granulométrie dans le chapitre 3 : Résultats et discussion des résultats.

Tableau 2.5 Granulométrie de l'image 6

Taille Tamis	1180 µm	710 µm	425 µm	250 µm	150 µm	106 µm	75 µm
%passant	100	10	10	10	10	10	0

a)

b)

Figure 2.11 Exemples d'images obtenues avec a) la caméra du haut,
b) la caméra de profondeur

2.3.7 Prendre une image du bas

Il s'agit ici d'un autre objectif du projet, prendre, en même temps que l'image du haut, une image du dessous de la boîte contenant les matériaux granulaires. Le script se trouve au niveau de l'Annexe II-6. Le principe est le suivant :

- la caméra est éloignée de presque 6,5 mm de façon à avoir une vue d'ensemble;
- la caméra est dirigée pleinement vers le haut soit avec un angle de -90 ^{o} avec le plan horizontal (x,z);
- l'image est prise par vue orthographique et non de perspective;
- l'image est prise avec une dimension de 1853×1023;
- des masques ont été rajoutés de façon à voir à travers la boîte.

Par la suite :

- une photo est prise à chaque fois que les particules sont générées et sont au fond de la boîte;
- un nombre de frame à partir duquel en prend les photos a été défini. Le nombre de *frame* choisi prend en compte le temps qu'il faut pour générer les différentes granulométries et la puissance de l'ordinateur.

Il est à noter qu'on a placé la deuxième caméra à cette distance et non à une distance symétrique de la première parce que la caméra n'a pas les mêmes propriétés et qu'avec cette distance-là, il y a plus ou moins la même vue. Par ailleurs il est à noter que le fond n'est pas le même pour la vue de dessous et dessus (figure 2.12). Ceci est dû au fait que la caméra du bas est dirigée vers le haut et donc le ciel, d'où cette couleur bleue. On n'a en aucun cas changé l'arrière-plan. La figure 2.12 illustre l'image du bas qui représente la même image et granulométrie que celle de l'image d'en haut.

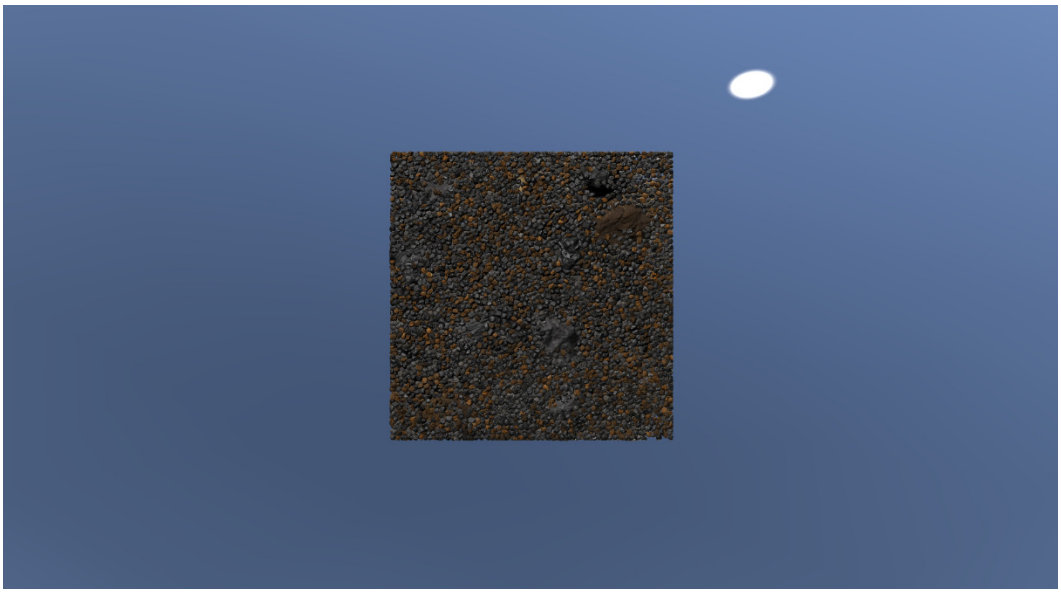


Figure 2.12 Image obtenue avec la caméra du bas

2.3.8 Enregistrer la taille des particules

Un script important pour enregistrer la taille des particules a été mis au point (Annexe II-7). Celui-ci est attaché à la particule solide et permet de :

- enregistrer la taille de chaque particule suivant les trois axes x,y et z;
- enregistrer le nom de chaque particule relative à la taille;
- enregistrer le nom de l'image auquel appartient cette particule;
- l'enregistrement se fait au moment où les particules sont au fond de la boîte, une frame avant de prendre le tout en photo;
- la forme est telle présentée au niveau de la figure 2.13 et montre une partie de l'exemple de l'image numéro 6.

```
1  image6;Rock4(Clone);0,0858579;0,100755;0,1007102
2  image6;Rock7(Clone);0,08890752;0,1038617;0,07837037
3  image6;Rock3(Clone);0,08481233;0,08260159;0,08849102
4  image6;Rock4(Clone);0,08888714;0,1007408;0,07970039
5  image6;Rock10(Clone);0,1014913;0,08406566;0,08658684
6  image6;Rock7(Clone);0,1007736;0,07567786;0,07538076
7  image6;Rock8(Clone);0,07824792;0,0824006;0,1028895
8  image6;Rock7(Clone);0,09717067;0,08049594;0,08216009
9  image6;Rock6(Clone);0,07929879;0,09202909;0,09604588
10 image6;Rock8(Clone);0,09648065;0,07869654;0,09391975
```

Figure 2.13 Exemple d'enregistrement de la taille des particules

L'idée d'introduire un code qui enregistre la taille des particules a deux objectifs : D'abord de vérifier si la granulométrie générée est conforme au résultat final. En effet ceci nous a permis d'éviter certains problèmes rencontrés comme : Un dysfonctionnement du code ce qui donne que les particules sont mal générées, où que les particules ne rentrent pas tous dans la boîte ou bien la traversent, ou encore de voir si la taille des objets est correcte. Ensuite ce code est utile pour les prochains travaux car il permet d'avoir une liste de toutes les particules qui figurent sur toutes les images et ainsi d'avoir toutes les données qu'il faut s'il fallait continuer sur ce travail.

CHAPITRE 3

TRAITEMENT D'IMAGES ET INTELLIGENCE ARTIFICIELLE

L'utilisation de la plateforme de jeux Unity a permis de créer une base de données contenant 3003 images. Néanmoins afin que ces images soient pleinement recevables par le réseau de neurones, il leur manque un traitement d'images et une extraction de caractéristiques.

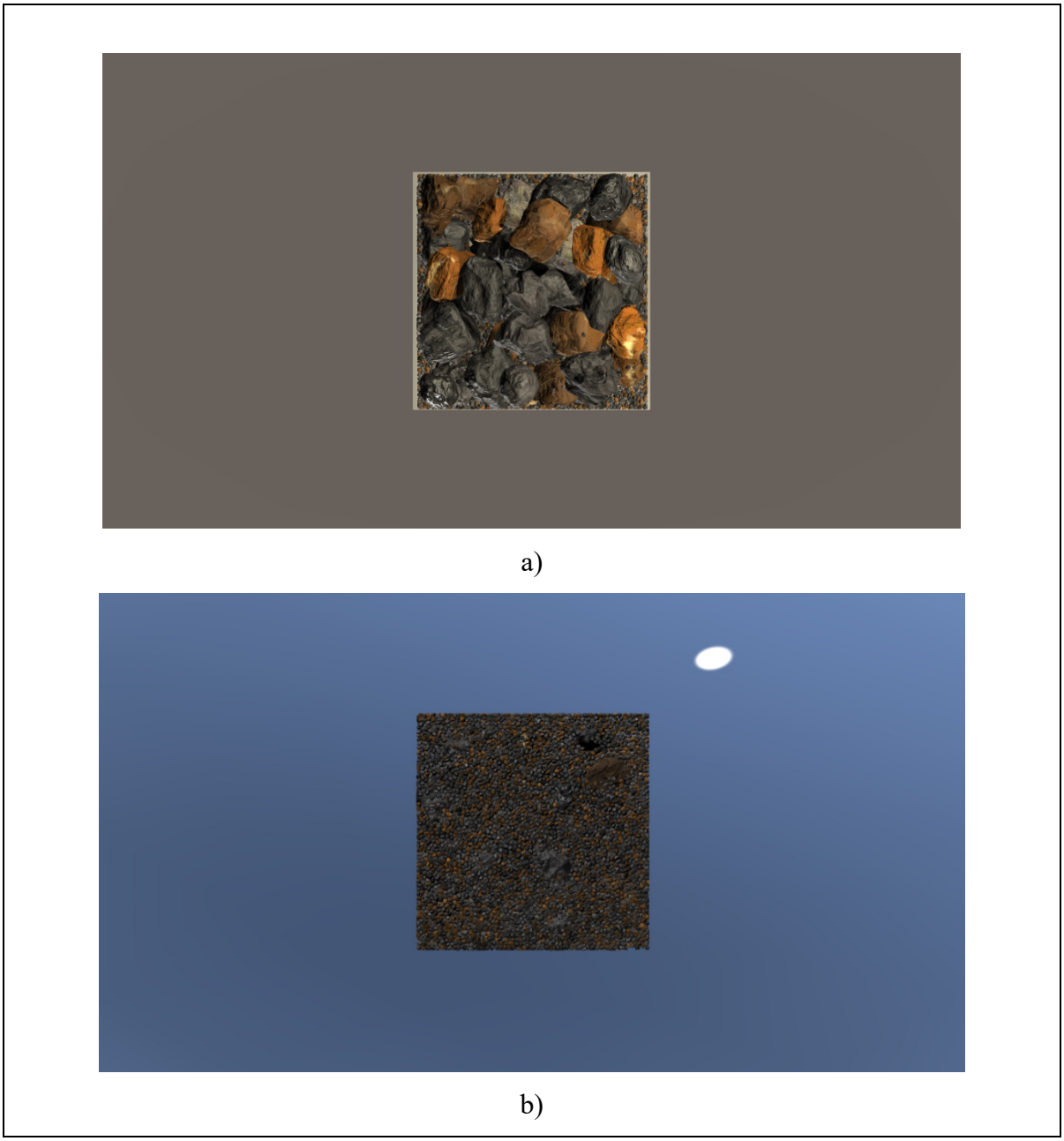
Cette section est constituée de trois parties. La première partie aborde le traitement des images. La deuxième partie explique l'extraction des caractéristiques des images liées aux textures : Il s'agit de l'entropie locale, les ondelettes de Haar et les textures d'Haralick. La troisième partie présente les paramètres des réseaux de neurones utilisés pour prédire la granulométrie et pour comparer les résultats des bases de données de Yade et Unity.

Au cours de ce chapitre, et dans le but de mieux comprendre les résultats obtenus, on a pris l'exemple de l'image 6. Ce choix s'explique purement par le fait que cette image a déjà été traitée dans la partie précédente et on a déjà décrit sa granulométrie.

3.1 Traitement d'images

Les images obtenues avec Unity doivent être préparées avant le calcul des caractéristiques. Comme mentionné précédemment, les images montrent des vues de dessus (figure 3.1a) et de dessous (figure 3.1b). Le travail du traitement d'images a les objectifs suivants :

- enlever le contour gris pour l'image du dessus et le contour bleu pour l'image du bas. Le but est de garder seulement une image de la boîte;
- convertir l'image en niveaux de gris;
- fusionner les deux images rognées pour qu'elles forment une seule image de 128×256 pixels.



a)

b)

Figure 3.1 Exemples d'images obtenues avec Unity : a) vue du haut et b) vue du bas

3.1.1 Redimensionnement des images

Un script a été écrit avec Matlab (Annexe III-1) afin de redimensionner les images de Unity de la taille initiale de 1853×1025 à un carré de 128×128. Cette étape a pour but de :

- préparer les images pour un réseau de neurones;

- ne laisse que la partie qui contient l'information utile soit les particules dans la boîte;
- permettre au réseau de neurones d'accéder à un maximum d'information sur plus petite base : La taille 128×128 rognée d'une image de 1853×1025 est la meilleure option possible;
- comparer les images de Unity avec les images de Pirnia et al (2018) utilisées par Manashti et al. (2020) en ayant la même dimension d'images.

Le code utilisé a été fait de la manière suivante :

- accéder au répertoire contenant les images;
- parcourir les images du haut et du bas une par une;
- convertir les deux images du haut et du bas en niveaux de gris en utilisant la fonction rgb2gray;
- redimensionner ces deux images avec la fonction imresize : Cette fonction permet de choisir la taille avec laquelle on souhaite redimensionner l'image ;
- combiner ces deux images;
- enregistrer le résultat dans un autre répertoire.

Il est à noter que la fonction rgb2gray convertit les valeurs RVB (Rouge Vert Bleu) en niveau de gris N en utilisant la somme pondérée des composantes R, G et B tel que :

$$N= 0.2989 \times R + 0.5870 \times G + 0.1140 \times B \quad (3.1)$$

Avec R, G et B les valeurs respectives des couleurs rouges, vertes et bleues. N le niveau de gris associé à ces couleurs.

3.1.2 Résultats intermédiaires et égalisateur d'histogramme

Une fois les traitements d'images faits, les résultats de la figure 3.2 ont été obtenus. Cette figure représente le résultat des figures 3.1a et 3.1b de la partie qui précède.

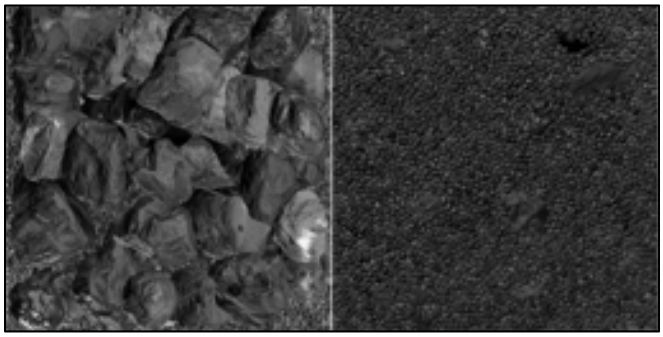


Figure 3.2 Exemple de résultat du traitement pour
les images de la figure 3.1

Le résultat de la figure 3.2 montre une image plutôt sombre. Ceci s'explique par le fait qu'on a choisi dans Unity un éclairage qui va vers le bas et que par conséquent l'image du bas est prise à contre-jour. Elle parait donc plus foncée.

Dans le but de mieux visualiser cette différence un égalisateur d'histogramme a été fait sur l'ensemble des images (Figure 3.3) où l'on voit très bien que les images du bas sont beaucoup plus sombres que les images du haut.

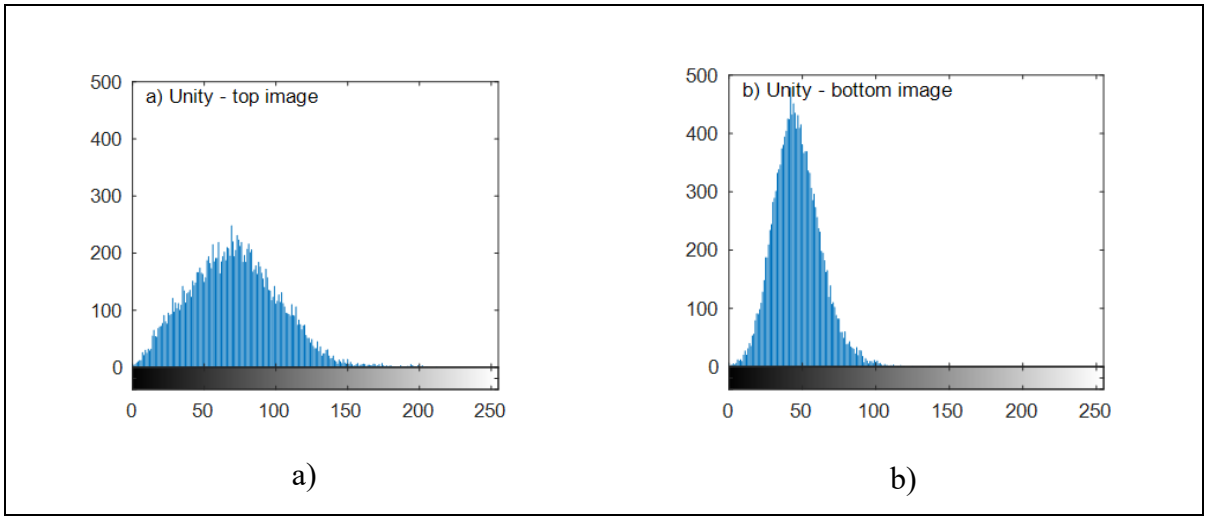


Figure 3.3 Égalisateur d'histogramme pour a) les images du haut b) les images du bas

3.1.3 Résultat du traitement final

Afin d'avoir des images moins sombres, une correction a été appliquée par l'intermédiaire de la fonction Matlab imadjust et on obtient le résultat de la figure 3.4. Cette fonction permet de jouer avec le contraste de l'image et dans ce cas on a choisi de l'éclaircir. Dans ce qui suit le travail se fera sur ces images, l'extraction des caractéristiques et les résultats obtenus avec le réseau de neurones ont été fait sur ces images.

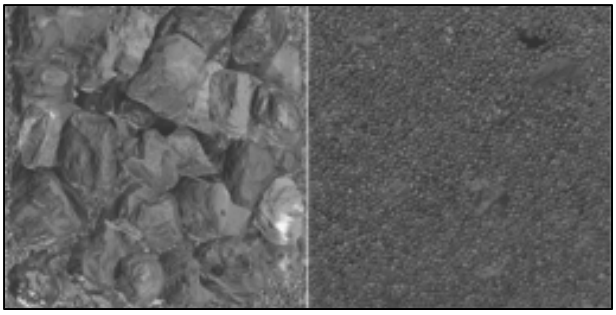


Figure 3.4 Résultats du traitement d'images
après correction

3.2 Extraction des caractéristiques

Les trois séries suivantes de paramètres texturaux ont été calculées :

- moyenne et écart type de l'entropie pour neuf filtres de rayons différents (18 caractéristiques);
- transformation par ondelette de Haar (49 caractéristiques);
- textures d'Haralick (14 caractéristiques).

Les mêmes paramètres ont été utilisés par Manashti et al. (2020) avec la base de données de Pirnia et al. (2018) tout en gardant la même échelle.

Il est à noter que dans tout ce qui suit le terme entropie fait office d'entropie locale calculée par moyenne et écart type, et l'ondelette pour dire d'ondelette de Haar.

Les paramètres obtenus avec les filtres d'entropie et la transformation par ondelette ont été déterminés avec les scripts MATLAB présentés au niveau de l'Annexe III. Les textures d'Haralick ont été obtenues avec le script Python qui est fourni à l'Annexe III. Les trois scripts réalisent les mêmes tâches principales :

- parcourir toutes les images;
- extraire les caractéristiques (filtres d'entropie, transformation par ondelette ou caractéristiques d'Haralick);
- enregistrer les paramètres afin de les utiliser dans les réseaux de neurones.

En ce qui concerne la partie du programme qui change, celles où on extrait les caractéristiques, on trouve que pour :

L'entropie

Comme mentionné dans la littérature, l'entropie est déterminée à travers un filtre : celui-ci vient parcourir tous les pixels, calcule l'entropie puis fait la moyenne et l'écart type de ces valeurs. La valeur du filtre dépend du voisinage choisi. Ce dernier peut varier par rapport à un pixel choisi, dans les quatre directions, d'un pixel voisin allant jusqu'à neuf pixels voisins. On parle ici d'un disque de rayon r=1 pixel allant jusqu'à r =9 pixel. Pour mieux comprendre la notion de disque, les disques r=1(figure 3.5a) et r=3 (figure 3.5b) ont été représentés. En ce qui concerne les pixels aux périmètres, le filtre opère en complétant les pixels manquants par symétrie.

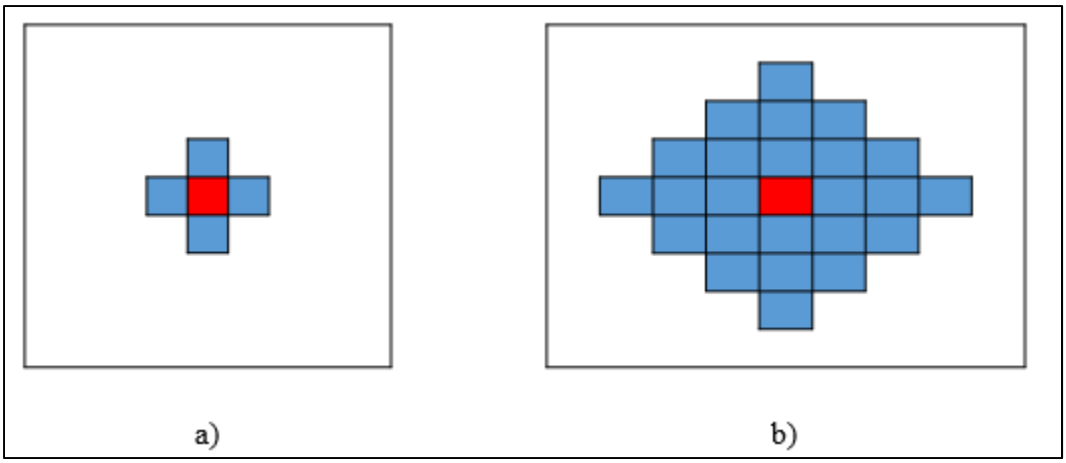


Figure 3.5 Exemple des disques a) r=1 et b) r=3

Ainsi pour le calcul de l'entropie, dans le script le choix s'est fait sur les disques allant de r=1 à r=9, en faisant une moyenne et l'écart type. On a représenté, pour la même image de la figure 3.4 les résultats intermédiaires obtenus pour les disques r=3 (figure 3.6a) et r=9 (figure 3.6b). Il est possible de remarquer que le disque r=3 accède plus facilement aux plus petites particules, alors que le disque r=9 accède plus facilement aux plus grandes particules. Ainsi l'avantage de prendre la moyenne de toutes ces valeurs est de pouvoir accéder aux petites et aux grandes tailles à la fois.

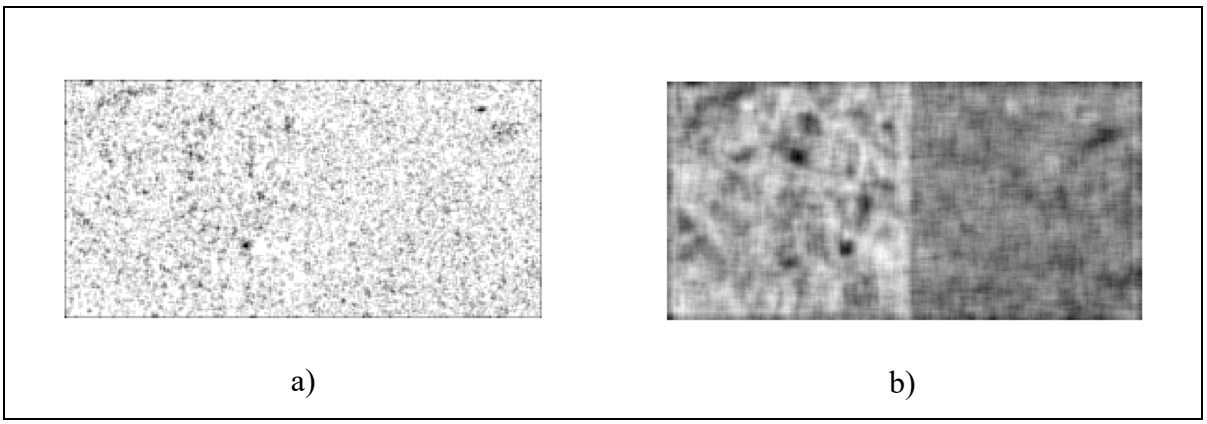


Figure 3.6 Résultats de l'entropie pour un filtrage avec un disque a) r=3 et b) r=9

Les ondelettes

Comme mentionné précédemment, la transformation en ondelette est utilisée pour la réduction d'échelle et la compression d'images.

Pour mieux comprendre comment les ondelettes ont été déterminées pour les images obtenues, il faut être conscient du principe de sous échantillonnage. Ce principe est à la base de la réduction d'échelle et permet d'aller chercher plus d'informations sur l'image en diminuant le nombre de pixels d'un facteur 2. Les images sur lesquelles on travaille font 256×128, et peuvent être sous échantillonner en sept niveau jusqu'à l'obtention d'une image qui contient 2×1 pixels. On va prendre l'exemple de la figure 3.4 pour laquelle l'image a été sous échantillonnée au niveau deux et quatre. Le niveau deux représente 64×32 pixels (figure 3.7a) et le niveau quatre 16×8 pixels (figure 3.7b). L'axe des abscisses et des ordonnées représente le nombre de pixels. Plus la couleur est bleue plus l'objet sur l'image est foncé. Plus la couleur est jaune plus l'objet est clair sur l'image de base.

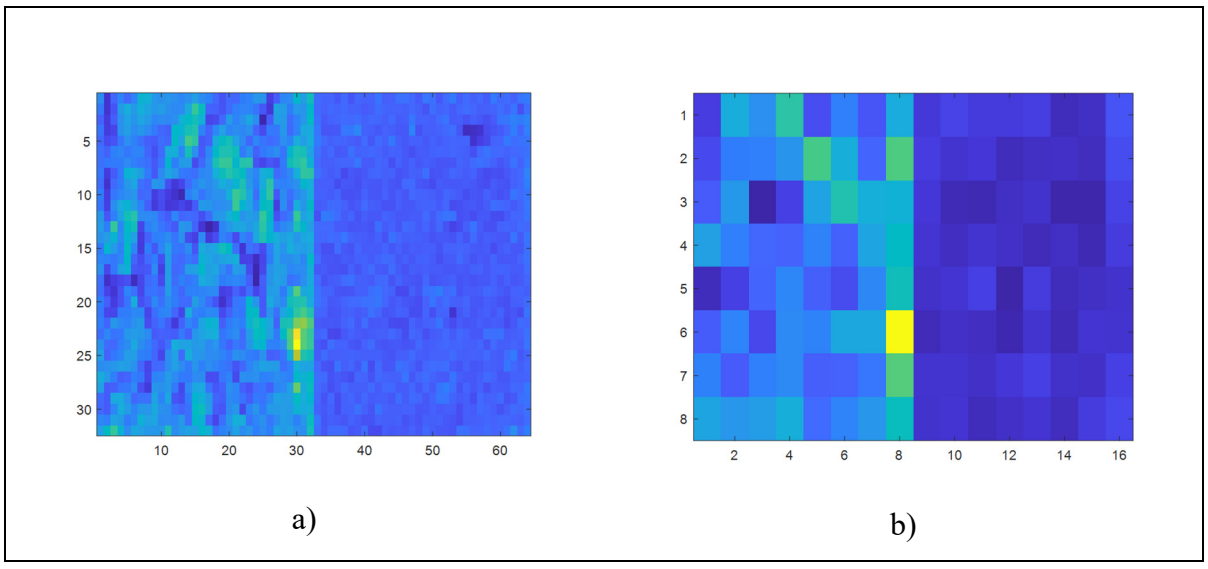


Figure 3.7 Sous échantillonnage a) niveau 2 et b) niveau 4

Il est clair que plus le niveau de l'échantillonnage augmente moins on arrive à voir l'image de base. Dans le code utilisé on a extrait les paramètres de la transformation de Haar au niveau

sept. Ce niveau contient le plus d'informations par rapport au nombre de pixels, et a été choisi en conséquence.

Les textures d'Haralick

Comme cité précédemment, les textures d'Haralick sont obtenues en parcourant tous les pixels en niveau de gris, puis en calculant l'ensemble des formules statistiques qui se trouvent au niveau de l'Annexe I. Durant ces calculs deux paramètres sont à prendre en compte : La GLCM ou matrice de co-occurrence, et la position du pixel voisin le plus proche de ce pixel. Pour la GLCM il existe des fonctions qui permettent de la déterminer directement et c'est ce qu'on a fait au niveau du code. En ce qui concerne la position du pixel voisin, il faut savoir qu'il est défini avec un angle par rapport à l'horizontal (figure 3.8). Cette figure montre que pour un pixel donné X, qui se trouve au milieu, on a la possibilité de choisir un pixel parmi huit en fonction de l'angle qu'on veut. Le choix de cet angle va influencer le calcul de la GLCM et les fonctions statistiques. Pour le code utilisé, on a fait les calculs avec les valeurs par défaut, soit avec un angle nul.

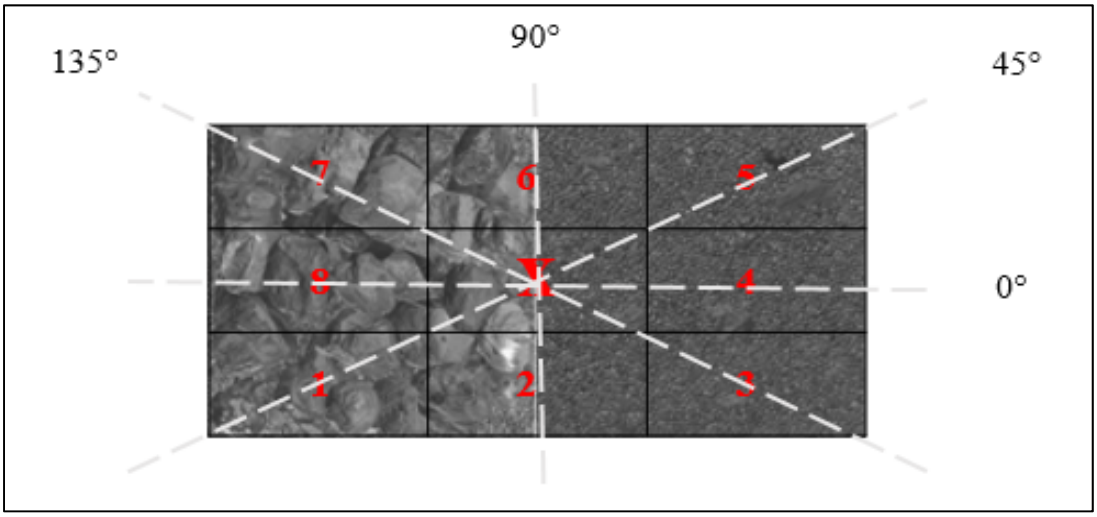


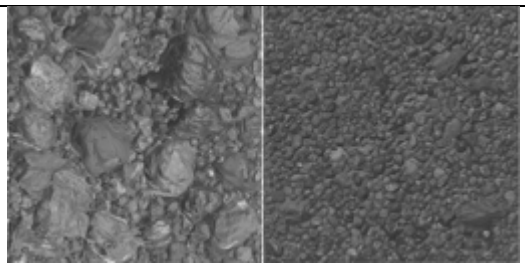
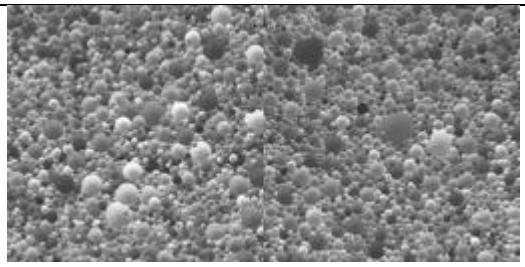
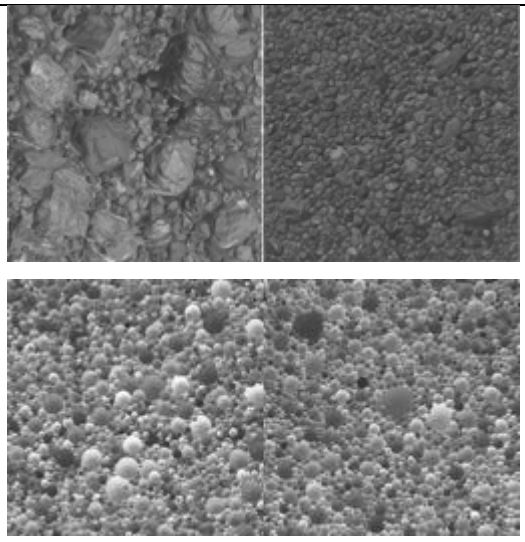
Figure 3.8 Choix du pixel voisin et de l'angle par rapport à l'horizontal

3.3 Création des réseaux de neurones artificiels

3.3.1 Bases de données

Les paramètres texturaux présentés à la section précédente ont été calculés pour trois bases de données présentées au tableau 3.1. Les images de la base de données *N2* ont été choisies aléatoirement parmi les 53130 images de la base de données de Prinia et al. (2018). Rappelons que ces images ont été obtenues avec le code d'éléments discrets Yade. Les mêmes 3003 images ont été utilisées pour chaque type de paramètres texturaux. La base de données N3 combine les images des bases de données *N1* et *N2* pour un total de 6006 images. L'objectif derrière l'utilisation de la base de données *N3* était de vérifier si les réseaux de neurones peuvent généraliser les prédictions de granulométries pour deux séries d'images assez différentes.

Tableau 3.1 Base de données utilisées pour l'entraînement des réseaux de neurones

Base de données	**Exemples d'images**	**Nombre d'images**
N1		3003
N2		3003
N3		6006

3.3.2 Réseaux de neurones

Des réseaux de neurones artificiels ont été utilisés pour prédire la granulométrie à partir des paramètres texturaux. Neuf réseaux de neurones ont été utilisés : trois par base de données, un pour chaque type de paramètres texturaux (entropie, ondelette et textures d'Haralick).

Un exemple de script MATLAB utilisé pour l'entraînement des réseaux de neurones est présenté à l'Annexe IV. L'ensemble des réseaux de neurones ont les caractéristiques suivantes :

- une couche cachée avec 10 nœuds;
- un nombre maximal d'époques égal à 10000;
- un gradient stochastique de 10^{-6};
- un taux d'apprentissage par défaut égal à 0,01;
- une performance souhaitée de 10^{-7};
- un ratio par défaut de 70 % pour l'entraînement, 15 % pour le test et 15 % pour la validation;
- les paramètres texturaux en entrée;
- les pourcentages passants liés à chaque tamis à la sortie.

La performance du modèle a été évaluée à partir de l'erreur quadratique moyenne (MSE) entre le pourcentage passant réel et le pourcentage passant prédit par le réseau de neurones. La racine carrée de l'erreur quadratique moyenne (RMSE) a été calculée pour chaque tamis. Ces valeurs de RMSE seront utilisées dans le prochain chapitre pour comparer les performances des réseaux de neurones pour les trois bases de données et les trois paramètres texturaux.

CHAPITRE 4

ANALYSE DES RÉSULTATS ET DISCUSSIONS

Le chapitre 2 a montré comment Unity a été utilisé pour créer une base de données d'images de matériaux granulaires. Le chapitre 3 a montré comment les paramètres texturaux ont été calculés pour ces images et comment ces paramètres ont été utilisés pour l'entrainement de réseaux de neurones pour prédire la granulométrie.

Ce chapitre présente l'ensemble des résultats et une discussion. Il est constitué de trois parties. La première partie présente les images de Unity. La deuxième partie compare les images de la nouvelle base données avec celles de Pirnia et al. (2018). Finalement, la dernière section montre les résultats obtenus par les réseaux de neurones pour les différents paramètres texturaux (entropie, ondelettes et textures d'Haralick) et pour chaque base de données.

4.1 Images de Unity

La création de la base de données avec Unity représente la base du projet. Pour 3003 courbes granulométriques différentes, ce travail a permis de créer :

- 3003 images avec une vue de dessus de la boîte contenant le spécimen;
- 3003 images avec une vue de dessous de la boîte contenant le spécimen;
- 3003 images présentant le relief de la surface du spécimen;
- une base de données de la taille de chaque particule pour chaque image.

Les 3003 courbes granulométriques correspondent à l'ensemble des combinaisons de pourcentages passants possibles par incrément de 10 % pour les tamis de 75, 106, 150, 250, 425 et 710 et 1180 µm.

Il est à noter que les images représentant le relief de la surface des spécimens n'ont pas été utilisées dans ce mémoire et ne seront pas présentées par la suite. Thurley (2010) a travaillé

sur des images de profondeur en utilisant un laser 3D, et a montré que ces images mettent en valeur les erreurs de chevauchement et de ségrégation qui faussaient les résultats de la granulométrie.

La base de données des tailles de particules pour chaque image a été utilisée pour vérifier si les granulométries réelles des images correspondaient au granulométries visées. L'erreur a été calculée pour chaque image en prenant compte le RMSE des pourcentages passants. Par apport à chaque tamis, la seule certitude est que l'erreur est quasi nulle pour le tamis le plus grossier. Pour chaque image l'erreur varie de 0 % (exemple image1, image 2900) à ± 1,6 % (image 1500). La moyenne tourne autour de ± 0,4 %.

La figure 4.1 donne des exemple d'images pour les granulométries 1, 500, 1000, 1500, 2000 et 3003. Les granulométries sont données au tableau 4.1. Le nom de chaque image sauvegardée par Unity est formé en combinant le numéro et le suffixe « _img ». Il est à noter que les images ont été rognées pour mettre l'accent sur les matériaux granulaires. Des exemples d'images brutes ont été montrés à la figure 4.1. On remarque que plus on avance dans les numéros d'images, plus la granulométrie devient fine. L'image du dessous parait systématiquement plus fine que l'image du dessus en raison de la ségrégation.

Tableau 4.1 Pourcentages passants pour les exemples d'images choisis

		Taille tamis (µm)						
		1180	**710**	**425**	**250**	**150**	**106**	**75**
Images	**1_img**	100	0	0	0	0	0	0
	500_img	100	70	40	10	10	0	0
	1000_img	100	80	60	60	60	40	0
	1500_img	100	90	70	10	10	0	0
	2000_img	100	90	90	90	90	70	0
	2500_img	100	100	90	10	10	0	0
	3003_img	100	100	100	100	100	100	0

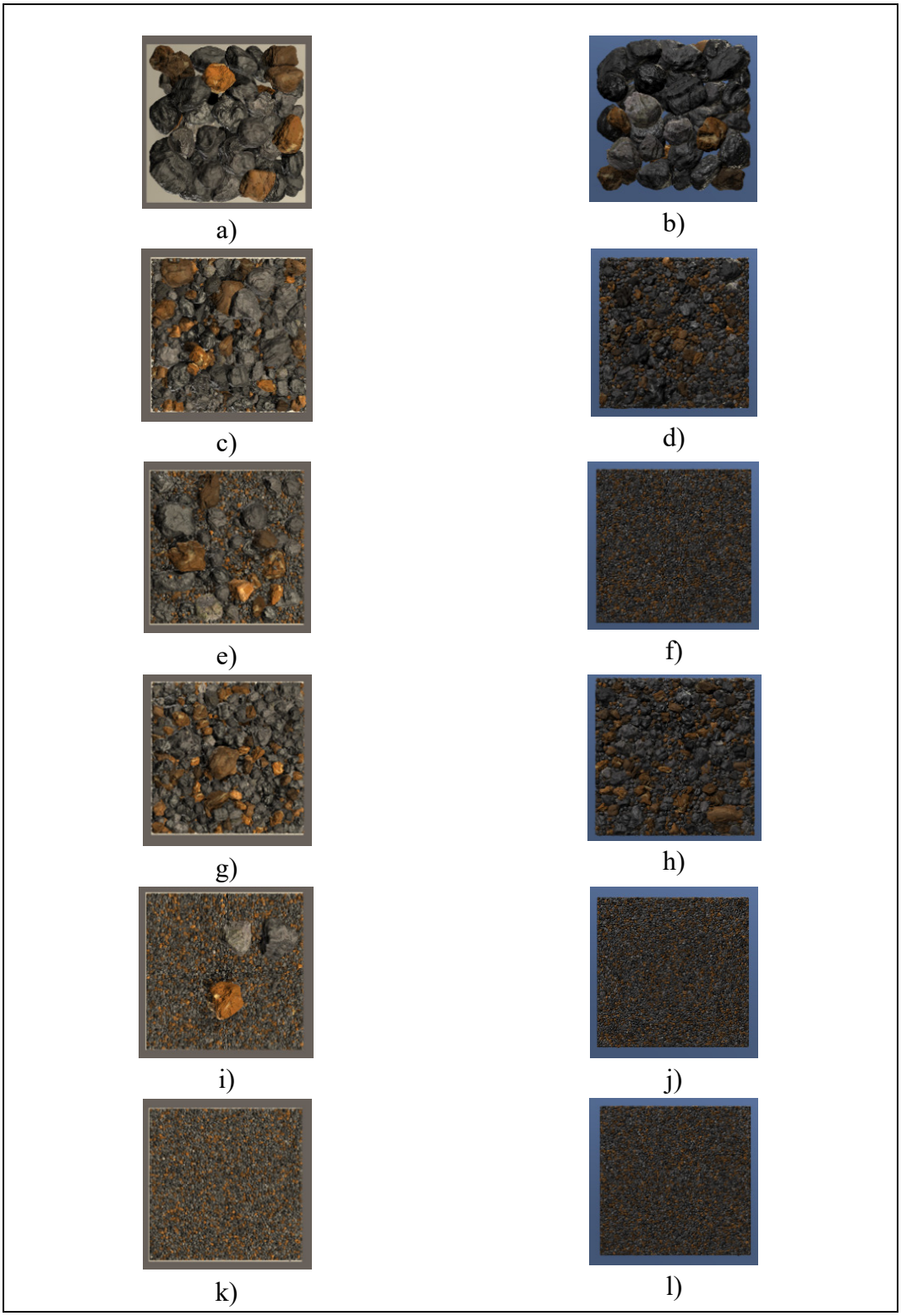


Figure 4.1 Vue du haut (colonne de gauche) et du bas (colonne de droite) pour les granulométries 1 (a et b), 500 (c et d), 1000 (e et f), 1500 (g et h), 2000 (i et j) et 3003 (k et l)

Il est à noter que le nombre de particules par image peut varier de 45 à quelques centaines de milliers. Le nombre total de particules générées pour les 3003 images est supérieur à 37,9 millions, ce qui donne une moyenne d'environ 12 500 particules par image.

La figure 4.2 présente les images combinées, redimensionnées et converties en niveaux de gris pour les mêmes granulométries (1, 500, 1000, 1500, 2000 et 3003). Dans chaque pair d'images, l'image du haut est à gauche et celle du bas est à droite. Les images ont une taille de 256×128 pixels. Cette taille correspond à celle qui a été utilisée par Pirnia et al. (2018) et Manashti et al. (2020). La taille des images a entre autres été réduite pour l'utilisation de réseaux de neurones de convolution qui nécessitent de plus petites images (Pirnia et al. 2018). Dans ce projet, la taille a été réduite pour faciliter la comparaison avec les résultats de Manashti et al. (2020).

4.2 Comparaison avec les images de la base de données de Pirnia et al. (2018)

Cette section présente une comparaison qualitative des images de Unity avec celles de Pirnia et al. (2018). On peut rappeler que les images de Pirnia et al. (2018) sont utilisées comme point de comparaison parce qu'elles ont le même format et parce que les mêmes tamis ont été utilisés pour définir les granulométries. Contrairement, aux images produites avec Unity, les images de Pirnia et al. (2018) présentent des sphères parfaites. Elles présentent une combinaison des images vue du dessus à gauche et vue du bas à droite. Les granulométries des quatre images utilisées pour la comparaison sont présentées au tableau 4.2. La figure 4.3 présente les images. La numérotation correspond à celle de la base de données de Unity.

On remarque qu'il y a une certaine similarité visuelle entre les images de Unity et Yade, à la différence que les images de Unity sont plus réalistes. D'autre part, les images de Unity montrent une ségrégation plus importante que les images de Yade. Cette ségrégation plus importante implique que les petites particules passent plus facilement entre les grandes particules dans Unity. Cette différence pourrait être due à un frottement moins important entre

les particules dans Yade. Il est à noter qu'on peut plus manipuler le paramètre du frottement des particules avec Yade qu'avec Unity.

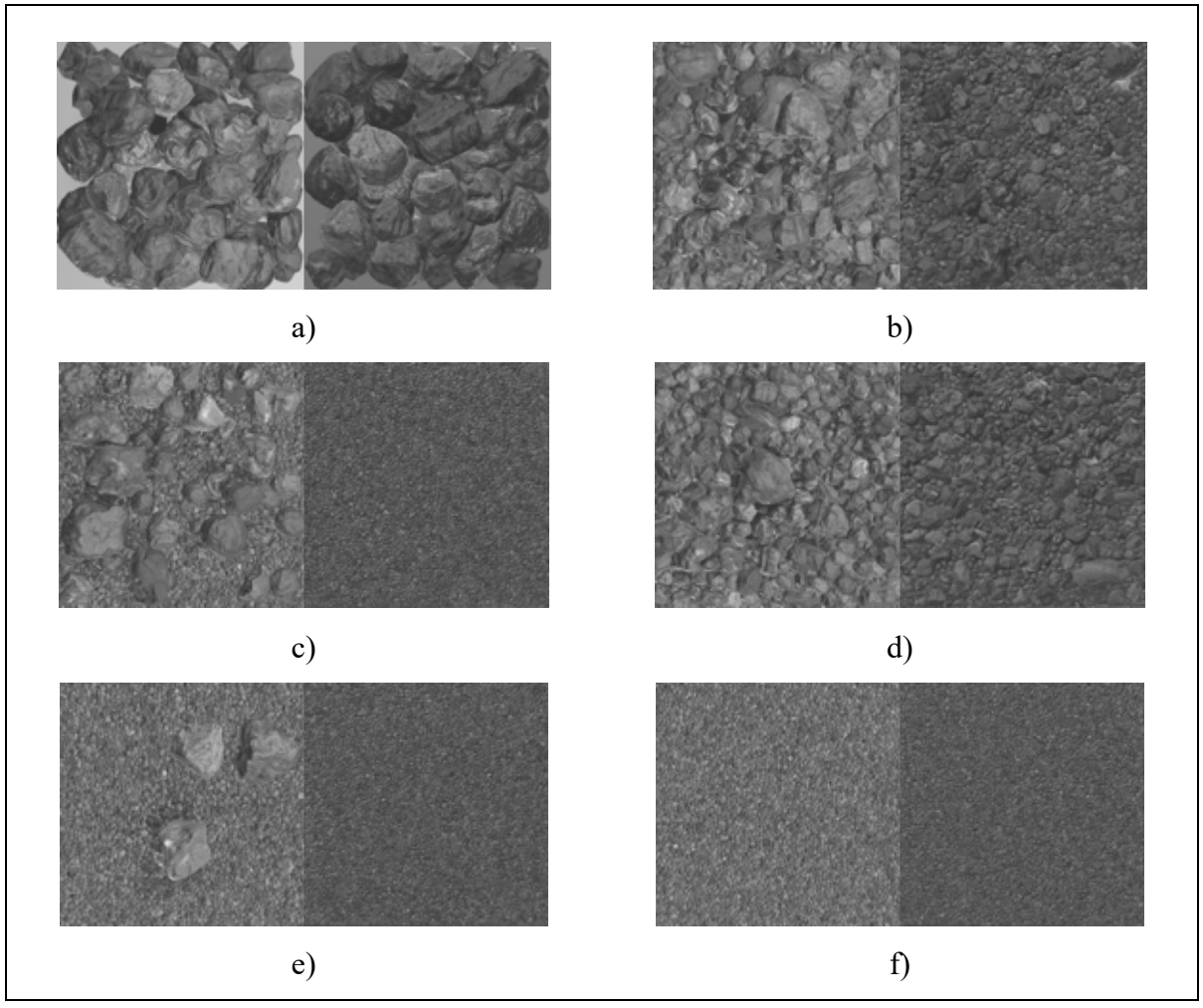


Figure 4.2 Images combinées pour les granulométries a) 1, b) 500, c) 1000, d) 1500, e) 2000 et f) 3003

Tableau 4.2 Granulométrie des quatre images pour la comparaison entre Unity et Yade

Passant à µm	**1180**	**710**	**425**	**250**	**150**	**106**	**75**
1_img	100	0	0	0	0	0	0
3003_img	100	60	20	20	20	20	0
2401_img	100	100	80	50	50	10	0
268_img	100	100	100	100	100	100	0

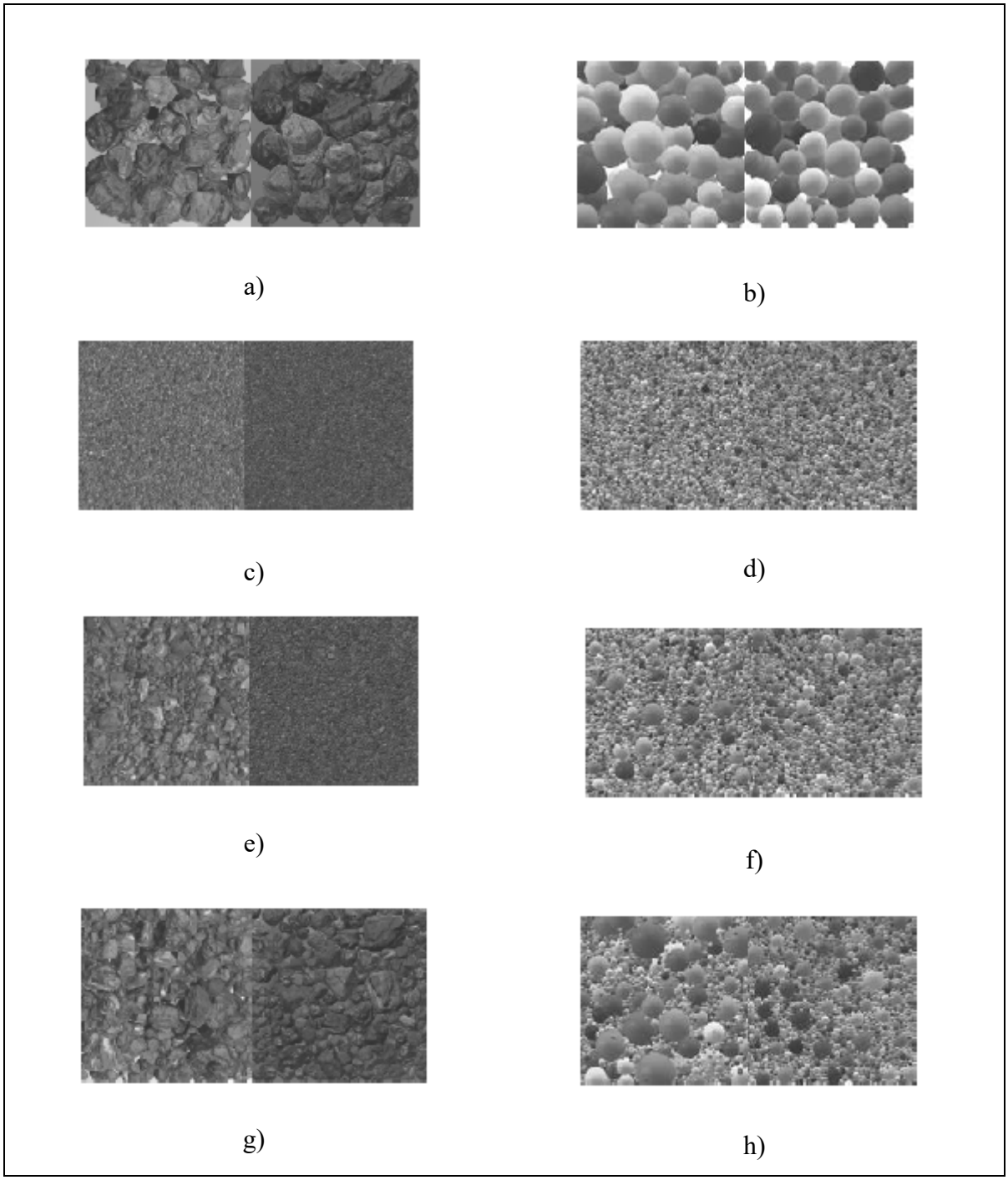


Figure 4.3 Comparaison des images de Unity (à gauche) et Yade (à droite) pour la même granulométrie a) 1 c) 3003 e) 2401 g) 268 a) et leurs équivalents respectifs avec Yade, b), d), f) et h)

4.3 Prédiction de la granulométrie à partir de réseaux de neurones

Comme mentionné au chapitre 3, trois types de paramètres texturaux ont été extraits des images et utilisés comme intrants pour la prédiction des pourcentages passants avec des réseaux de neurones :

- moyenne et écart type de l'entropie locale;
- l'ondelette par transformée de Haar;
- les textures d'Haralick.

Trois bases de données différentes ont été utilisées : les images de Unity (3003 images), celles de Yade (3003 images) et les deux séries d'images combinées (6006 images).

La racine carrée de l'erreur quadratique moyenne (*RMSE*, de *Root Mean Squared Error*) a été utilisée pour définir la performance des réseaux de neurones. Le paramètre RMSE décrit l'erreur des pourcentages passants prédits par rapport aux pourcentages passants réels des images. Le paramètre RMSE peut être calculé pour un tamis spécifique ou pour l'ensemble des tamis. L'équation utilisée pour calculer le RMSE est la suivante :

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (\%P_i - \%O_i)^2}{n_t}} \tag{4.1}$$

où :

- n_t : Le nombre d'éléments qu'on souhaite vérifier (produit du nombre de tamis et du nombre d'images);
- $\%P_i$: Le pourcentage prédit de l'élément i;
- $\%O_i$: Le pourcentage observé ou réel de l'élément i.

4.3.1 Images de Unity

Cette partie présente les résultats obtenus avec les 3003 images de Unity pour lesquels on a extrait l'entropie locale, l'ondelette de Haar et les textures d'Haralick.

Dans ce qui suit les figures représentent les pourcentages passants réels et prédits par le réseau de neurones pour les images de Unity. Les cinq premières images correspondent aux tamis de 710, 425, 250, 150 et 106 µm. La sixième image combine l'ensemble des points pour les cinq tamis.

Le tamis 710 µm présente moins de points pour les faibles valeurs du pourcentage passant et le tamis 106 µm moins de points pour les valeurs hautes du pourcentage passant. Ceci s'explique par le fait qu'il s'agit du pourcentage passant : Les petites particules vont passer à travers les grands tamis, et les grandes particules vont rester sur les grands tamis.

La figure 4.4 donne les résultats obtenus avec la moyenne et l'écart type de l'entropie comme intrants. Pour des prédictions parfaites, l'ensemble des points seraient sur la diagonale. On remarque une corrélation entre les valeurs réelles et prédites, mais cette corrélation n'est pas parfaite. La valeur de RMSE pour l'ensemble des tamis est de 7,5 %. La valeur de RMSE augmente avec la taille des mailles des tamis. Elle passe de 3,3 % pour le tamis de 106 µm (plus petit tamis) à 9,6 % pour le tamis de 710 µm (plus grand tamis). Plus la taille des mailles des tamis augmente, moins le réseau est capable de prédire le pourcentage passant avec précision.

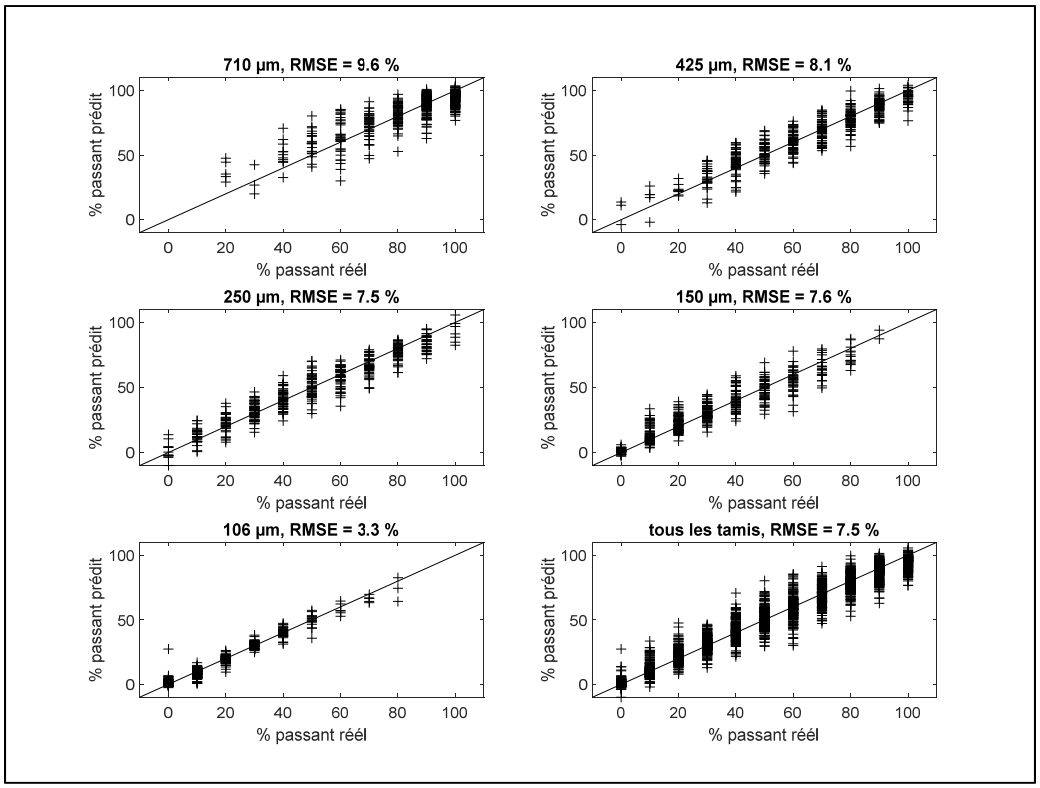


Figure 4.4 Résultats de l'entropie pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis, pour les images de Unity

La figure 4.5 donne les résultats avec l'ondelette de Haar comme paramètre. La valeur de RMSE pour l'ensemble des tamis est un peu plus grande que pour l'entropie (8,1 %). Comme c'était le cas avec l'entropie, les performances diminuent quand la taille des particules augmente. La valeur de RMSE est maximale pour le tamis le plus grossier (10,2 % pour le tamis 710 µm) et minimale pour le tamis le plus fin (5,4 % pour le tamis 106 µm).

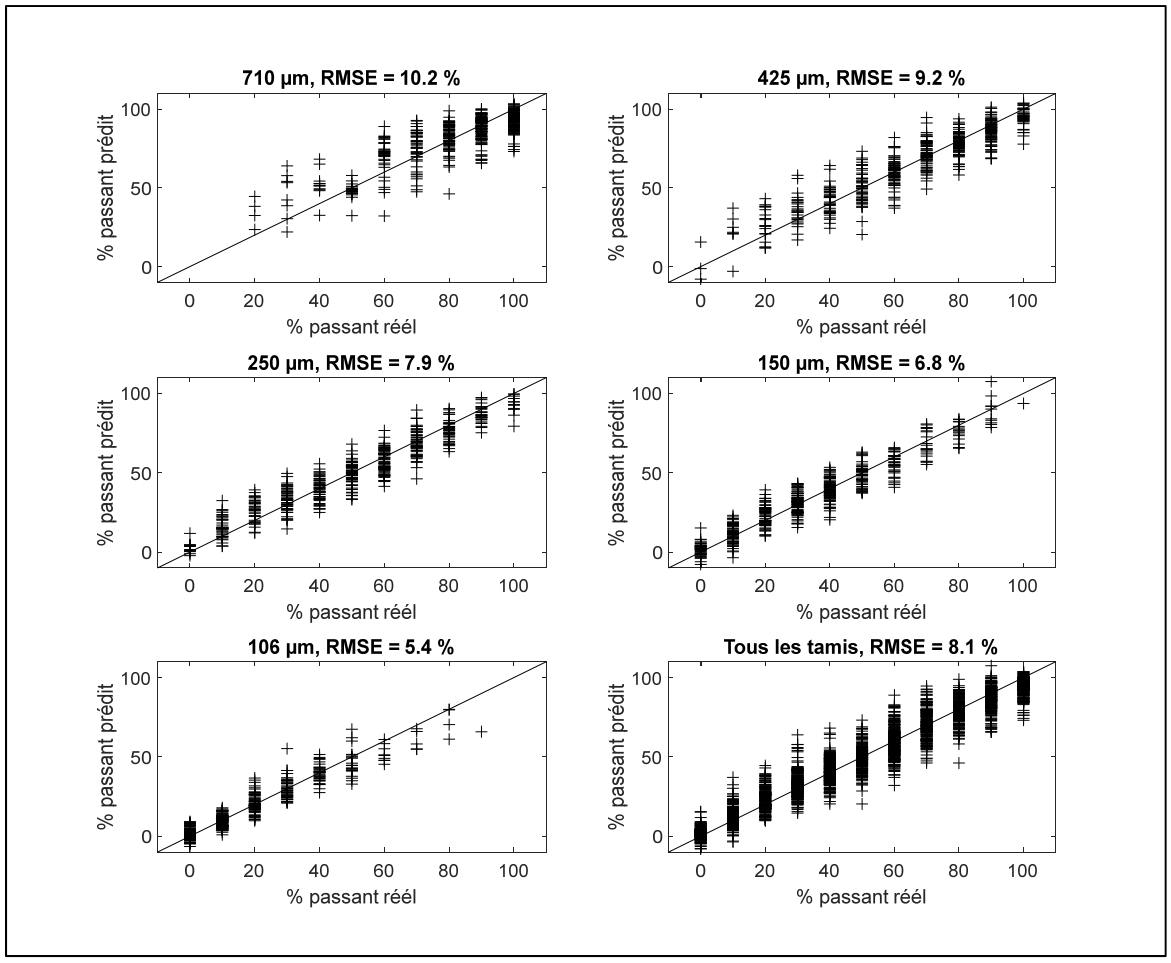


Figure 4.5 Résultats de l'ondelette pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis, pour les images de Unity

La figure 4.6 donne les résultats pour les textures d'Haralick. Comme c'était le cas avec l'entropie locale et l'ondelette de Haar, les performances diminuent quand la taille des particules augmente. La valeur de RMSE est maximale pour le tamis 425 µm (12,6 % pour ce tamis) et minimale pour le tamis le plus fin (5,4 % pour le tamis 106 µm).

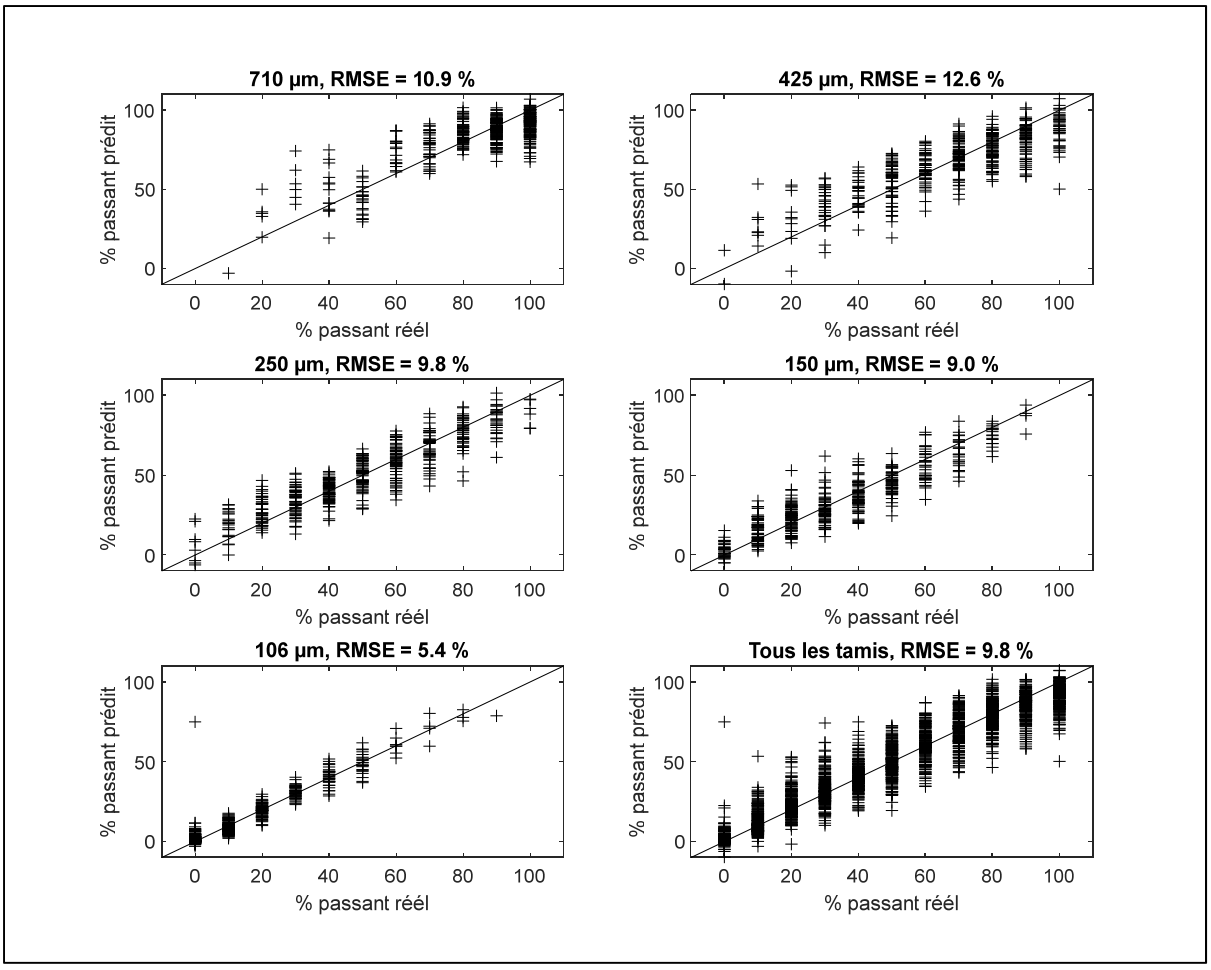


Figure 4.6 Résultats des textures d'Haralick pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis, pour les images de Unity

4.3.2 Images de Yade

La figure 4.7 présente les résultats pour les images de Yade en prenant l'entropie comme paramètre. La valeur de RMSE pour l'ensemble des tamis est un peu plus petite que pour les images de Unity (7,2 % versus 7,5 %), ce qui indique de meilleures performances dans l'ensemble. Comme c'était le cas avec l'entropie pour les images de Unity, les performances varient avec la taille des particules. Par contre, la relation entre les valeurs de RMSE et la taille des particules est différente. La valeur de RMSE est maximale pour les tamis intermédiaires (p. ex. 8,4 % avec le tamis 425 µm) et minimale pour les tamis le plus fin (5,9 % pour le tamis 106 µm) et le plus grossier (7,0 % pour le tamis 710 µm). La valeur de RMSE pour l'ensemble

des tamis est de 7,2 %. On peut noter que les performances sont moins bonnes avec Yade pour le plus petit tamis (RMSE de 5,8 % avec Yade versus 3,3 % avec Unity).

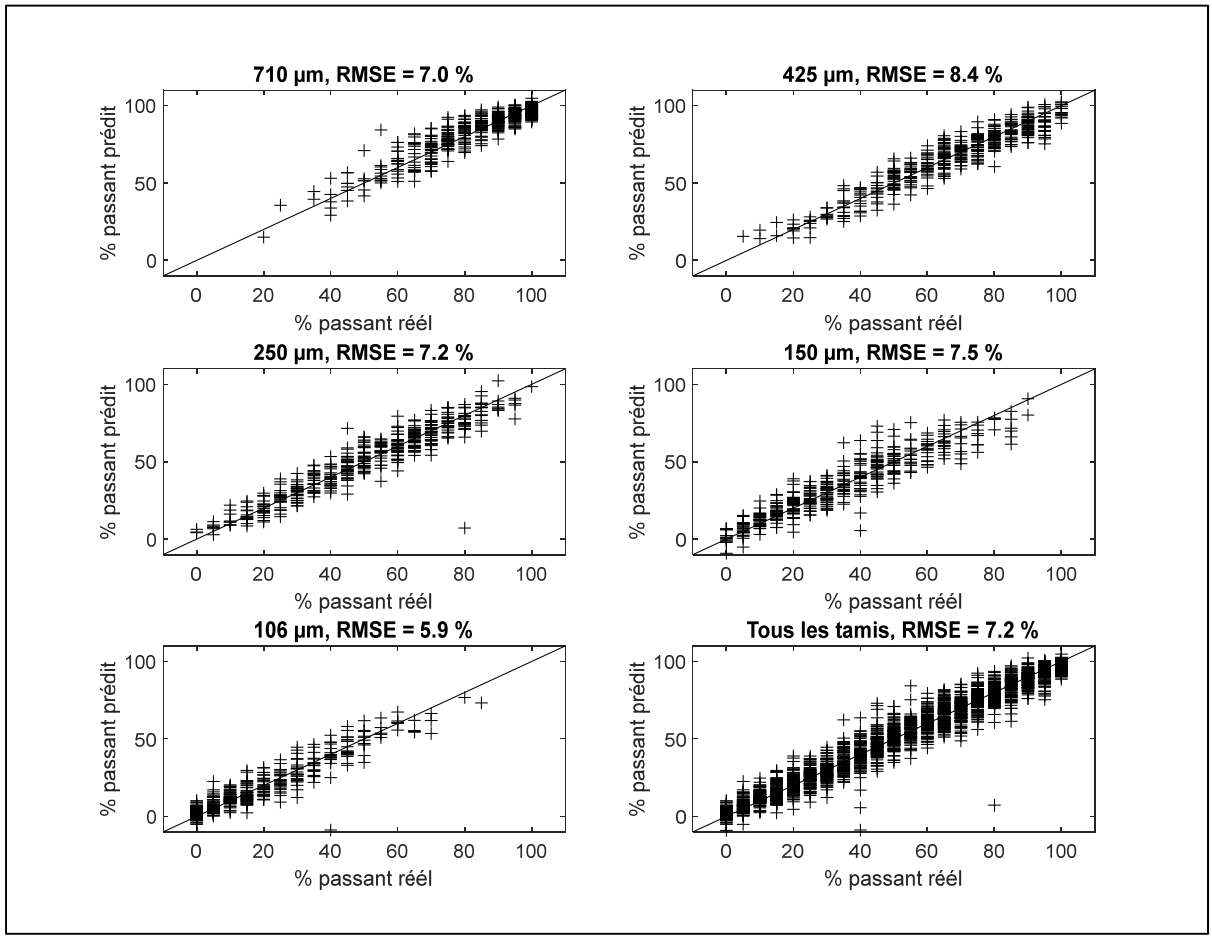


Figure 4.7 Résultats de l'entropie avec les images de Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis

La figure 4.8 donne les résultats en prenant l'ondelette de Haar comme paramètre. La valeur de RMSE pour l'ensemble des tamis (7,4 %) est un peu plus grande que pour l'entropie (7,2 %) et un peu plus faible que celle qui est obtenue avec les images de Unity pour les ondelettes de Haar (8,1 %). Les performances diminuent quand la taille des particules augmente. La valeur de RMSE est maximale pour le tamis le plus grossier (9,8 % pour le tamis 710 µm) et minimale pour le tamis le plus fin (4,5 % pour le tamis 106 µm).

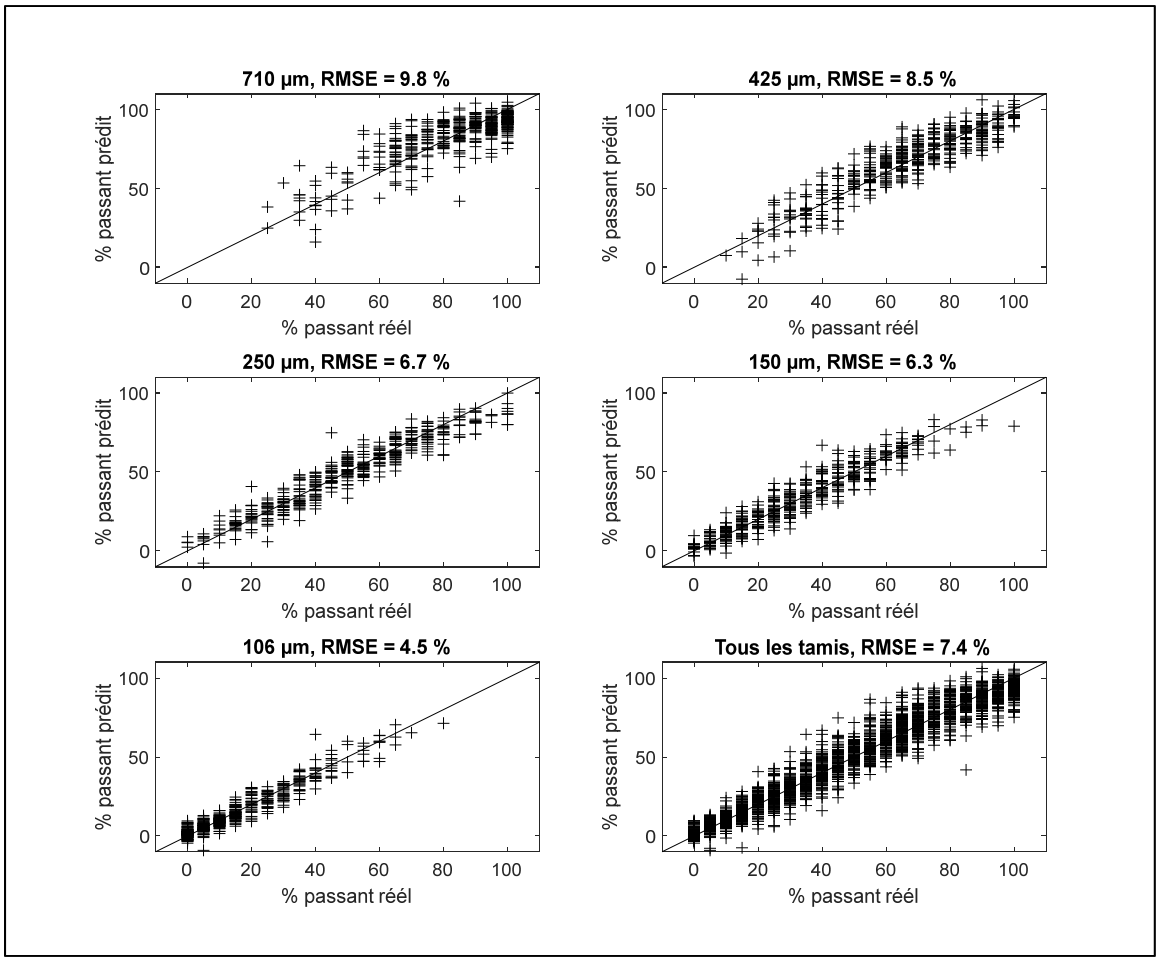


Figure 4.8 Résultats de l'ondelette avec les images de Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis

La figure 4.9 donne les résultats pour les textures d'Haralick. Comme c'était le cas précédemment, les performances sont meilleures pour le tamis le plus fin (RMSE de 3,6 % pour le tamis de 106 µm). La valeur de RMSE est maximale pour le tamis 425 µm (7,4 %). Des valeurs de RMSE de l'ordre de 7 % ont été obtenues pour les autres tamis. Parmi les trois types de paramètres texturaux, les textures d'Haralick produisent la plus grande différence entre les performances pour les images de Unity et de Yade. Entre les deux séries d'images, la valeur de RMSE passe de 9,8 % pour Unity à 6,5 % pour Yade pour l'ensemble des tamis. Les images de Yade et les textures d'Haralick ont produit la plus faible valeur de RMSE parmi les neuf combinaisons de méthodes et de bases de données.

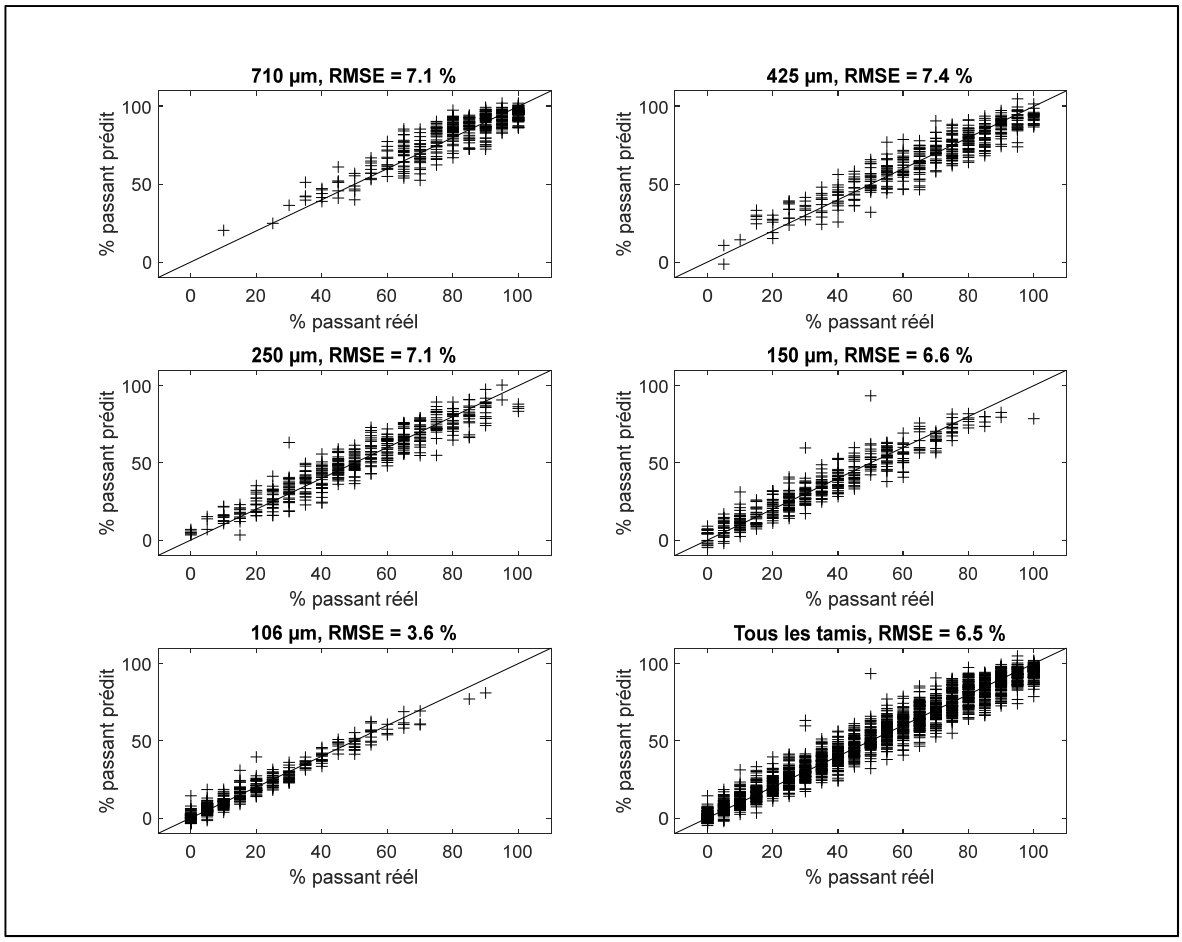


Figure 4.9 Résultats des textures d'Haralick avec les images de Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis

4.3.3 Les images de Unity et Yade combinées

Même principe qu'avec les images de Unity où le réseau de neurones va afficher à la fin le RMSE entre le pourcentage passant prédit et réel, à la seule différence près ou dans ce cas on va traiter une base de données qui contient le double du nombre d'images soit 6006 au lieu de 3003. Au niveau des résultats, on a gardé la même structure avec les cinq premières images qui correspondent aux tamis 710, 425 µm, 250, 150 et 106 µm. La sixième image combine l'ensemble des points pour les cinq tamis.

La figure 4.10 présente les pourcentages passants réels et prédits par le réseau de neurones pour les images de Unity +Yade en prenant l'entropie comme paramètre. La valeur de RMSE pour l'ensemble des tamis (7,1 %) est comparable aux images de Yade (7,2 %) et est légèrement plus petite que pour les images de Unity (7,5 %). Pour cette base de données la valeur maximale du RMSE est de 7,9 % pour les tamis 710 et 150 µm à la différence des images de Unity (9,6 % avec le tamis 710 µm) et Yade (8,4 % avec le tamis 425 µm). La valeur minimale du RMSE est de 4,6 % pour le tamis 106 µm qui est le tamis le plus petit comme ça a été le cas avec les images de Unity et Yade.

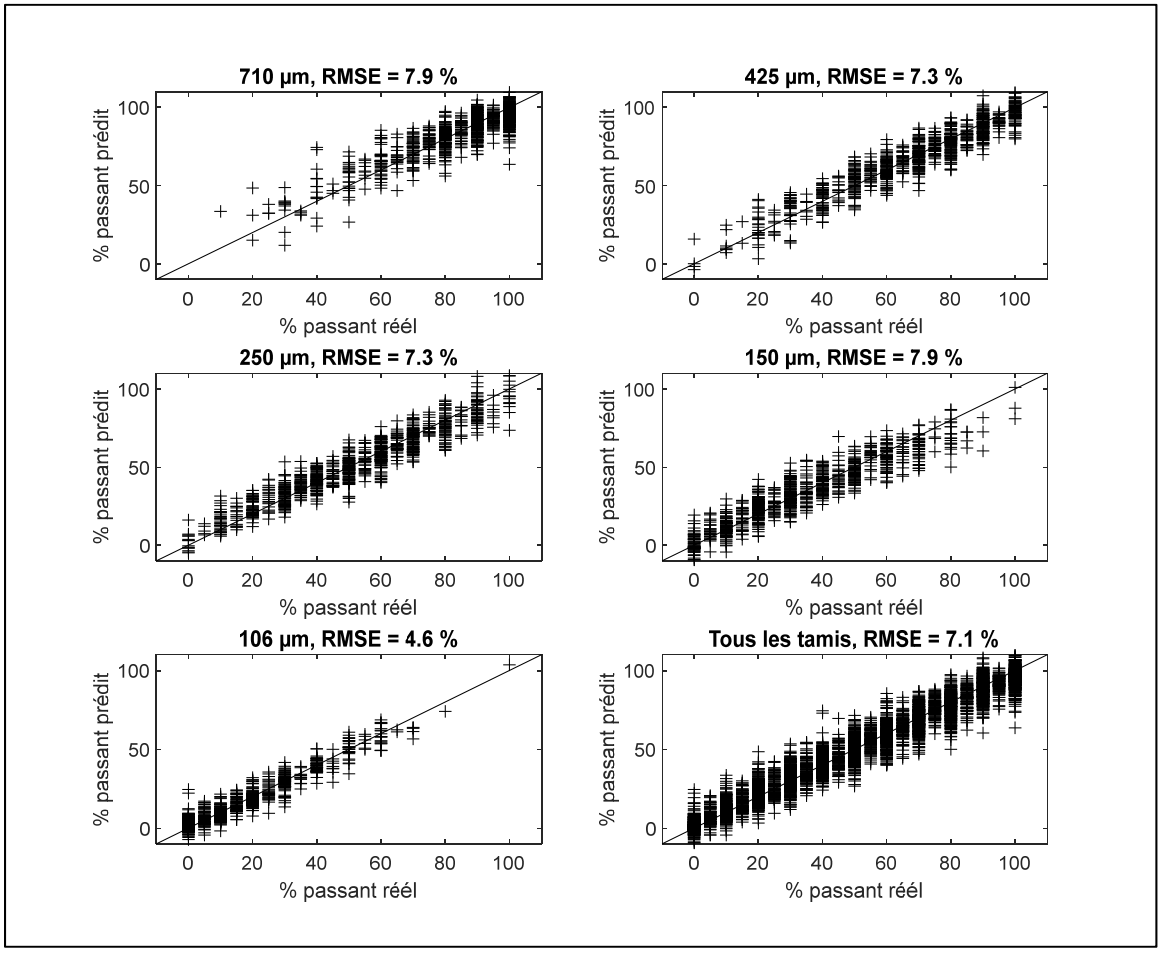


Figure 4.10 Résultats de l'entropie avec les images combinées de Unity et Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis

La figure 4.11 présente les pourcentages passants réels et prédits par le réseau de neurones pour les images de Unity + Yade en prenant l'ondelette comme paramètre. Tout comme les images de Unity et Yade la valeur maximale du RMSE est atteinte pour de pour le tamis 710 µm qui est le tamis le plus grand avec des valeurs proches (9,8 % pour les images de Yade, 10,2 % avec les images de Unity, contre 10,1 % pour les images combinées). Même chose pour la valeur minimale du RMSE où elle est aussi atteinte pour le tamis 106 µm qui est le tamis le plus petit avec 5,4%. Le RMSE global est de 8,2 % plus élevé que les images de Yade (7,4 %) et celles de Unity (8,1 %).

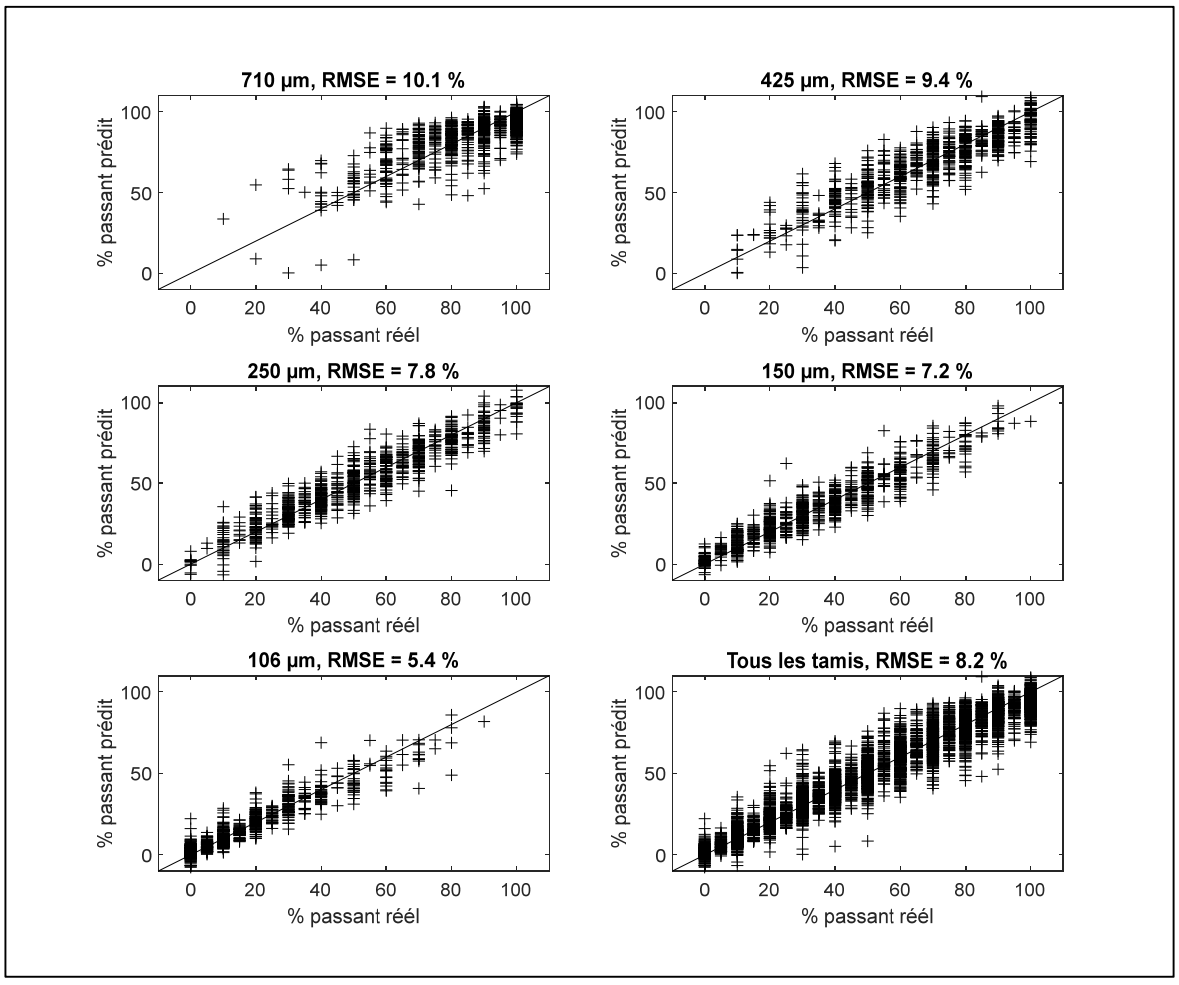


Figure 4.11 Résultats de l'ondelette avec les images combinées de Unity et Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis

La figure 4.12 présente les pourcentages passants réels et prédits par le réseau de neurones pour les images de Unity +Yade en prenant les textures d'Haralick comme paramètre. La valeur du RMSE global est de 9,9 % ce qui est plus élevé que l'entropie (7,1 %) et l'ondelette de Haar (8,2 %) pour les images de Unity + Yade. La valeur maximale du RMSE est de 12,7 % pour le tamis 425 µm. La valeur minimale du RMSE est de 4,8 % pour le tamis 106 µm et est du même ordre que l'entropie (4,6 %).

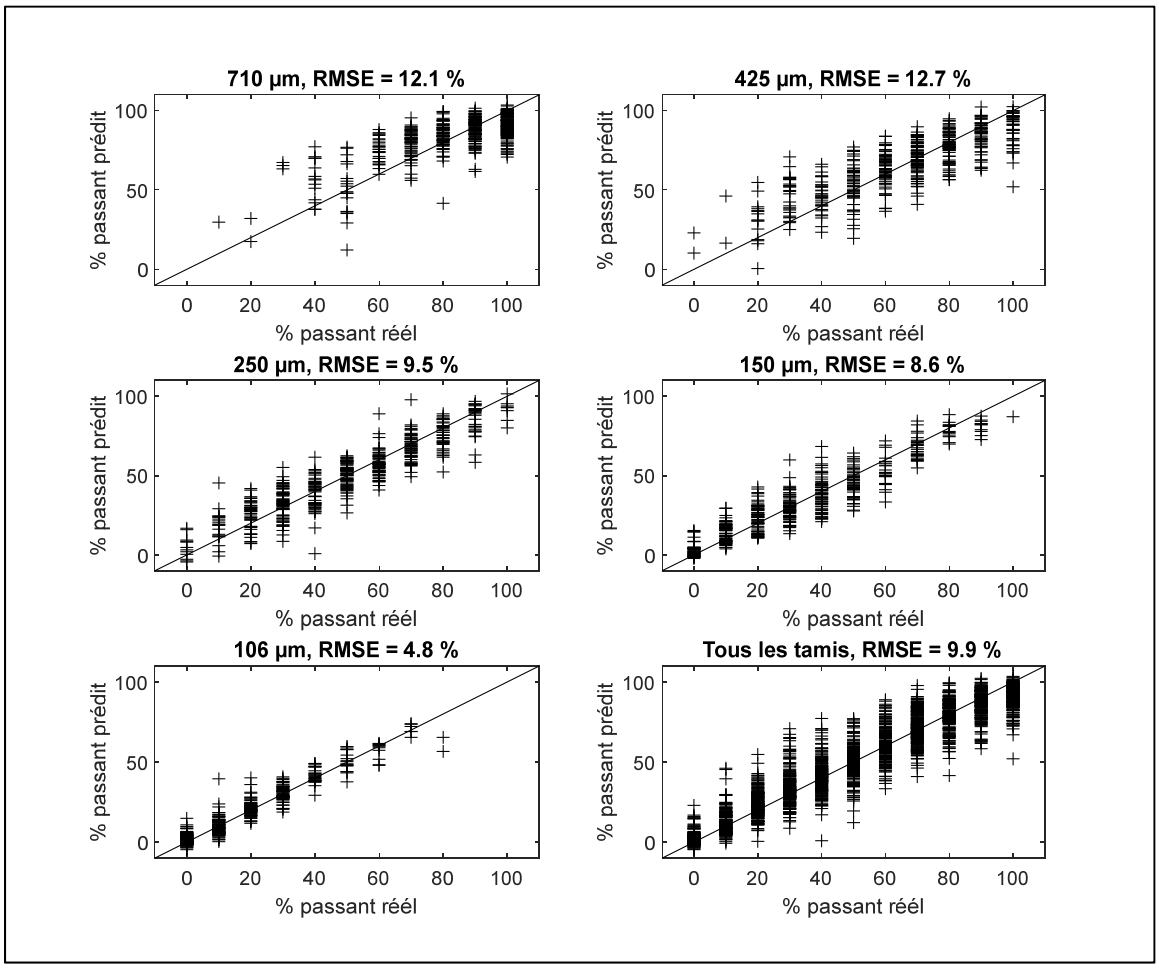


Figure 4.12 Résultats des textures d'Haralick avec les images combinées de Unity et Yade pour les tamis 710, 425 µm, 250, 150 et 106 µm et l'ensemble des tamis

4.3.4 Discussion

Le tableau 4.3 présente une synthèse des valeurs de RMSE pour l'ensemble des tamis pour les neuf combinaisons de paramètres texturaux et de bases de données. Parmi les trois caractéristiques utilisées, l'entropie donne de meilleurs résultats avec une valeur moyenne de RMSE de 7,3 % devant l'ondelette (7,9 %) et les textures d'Haralick (8,7 %). La comparaison des résultats montre que la valeur moyenne de RMSE varie relativement peu en fonction des caractéristiques texturales et de la base de données utilisées (entre 6,5 % et 9,9 %). Comme les paramètres des différentes méthodes n'ont pas été optimisés (par exemple, nombre de niveaux de gris pour l'entropie, relation *R* entre les pixels utilisés pour calculer la matrice de cooccurrence avec Haralick, ou nombre de couches pour les réseaux de neurones), il est probable que les valeurs de RMSE pour les différentes méthodes puissent être améliorées. Il est possible que l'optimisation de ces paramètres réduise encore plus l'écart entre les méthodes.

Tableau 4.3 Valeurs du RMSE pour l'ensemble des tamis pour les différents cas obtenus

Base de données	Entropie	Ondelette	Haralick
Unity	7,5	8,1	9,8
Yade	7,2	7,4	6,5
Unity + Yade	7,1	8,2	9,9

Le RMSE obtenu pour les textures d'Haralick est plus faible pour les images obtenues avec Yade que celles obtenues avec Unity. Ceci pourrait s'expliquer par la différence de luminosité des deux différentes bases de données. En effet les textures d'Haralick sont dépendantes de l'éclairage et de la luminosité des images (Haralick et al., 1973) et les images de Yade sont plus claires que celles de Unity.

Parmi les trois bases de données, les images de Yade donnent un plus faible RMSE avec une moyenne de 7,0 % devant les images de Unity+Yade (8,4 %) et celles de Unity (8,5 %). Ceci s'explique par le fait que les images de Yade représentent des images parfaites avec des particules sphériques moins réalistes que celles de Unity.

Les performances obtenues pour la base de données Unity+Yade sont surprenantes. Le fait de combiner les deux types d'images n'augmente pas la valeur de RMSE. Cette observation implique que l'utilisation de paramètres texturaux et de réseaux de neurones peut être généralisée à différents types d'images. Néanmoins il est à noter que la base de données de Unity+Yade contient 6006 images au lieu de 3003. Un nombre plus élevé d'images affecte l'apprentissage de manière positive.

Nous avons eu recours à un réseau de neurones avec peu de couches soit un réseau peu profond. Ceci s'explique par le fait que le réseau de neurones a été entraîné sur des textures, et vu que c'est des motifs qui se répètent un réseau peu profond suffit amplement (Liu et al., 2019).

Utiliser les images de YADE dans le réseau de neurones de Unity et vice-versa ne fonctionne pas. La figure 4.13 représente les estimations du pourcentage passant par rapport au pourcentage réel en utilisant les images de Unity dans le réseau de neurones utilisé pour les images de Yade (représentés au niveau de la première colonne soit les figures 4.13a, 4.13c et 4.13e) et inversement (représentés au niveau de la deuxième colonne soit les figures 4.13b, 4.13d et 4.13f). Ces figures ont été représentées de façon à ce que l'entropie apparaisse au niveau de la première ligne (figure 4.13a, 4.13b), l'ondelette la deuxième ligne (figure 4.13c, 4.13d) et les textures d'Haralick la dernière ligne (figure 4.13e, 4.13f). Ces figurent confirment bien qu'en changeant la base de données d'images, le réseau de neurones est incapable de faire de donnes prédictions. Le RMSE varie de 43,0 % pour l'image c) à 750,6 % pour l'image 4.13f. Dans ce cas-là le RMSE est de respectivement 361,8 % et 94,2 %.

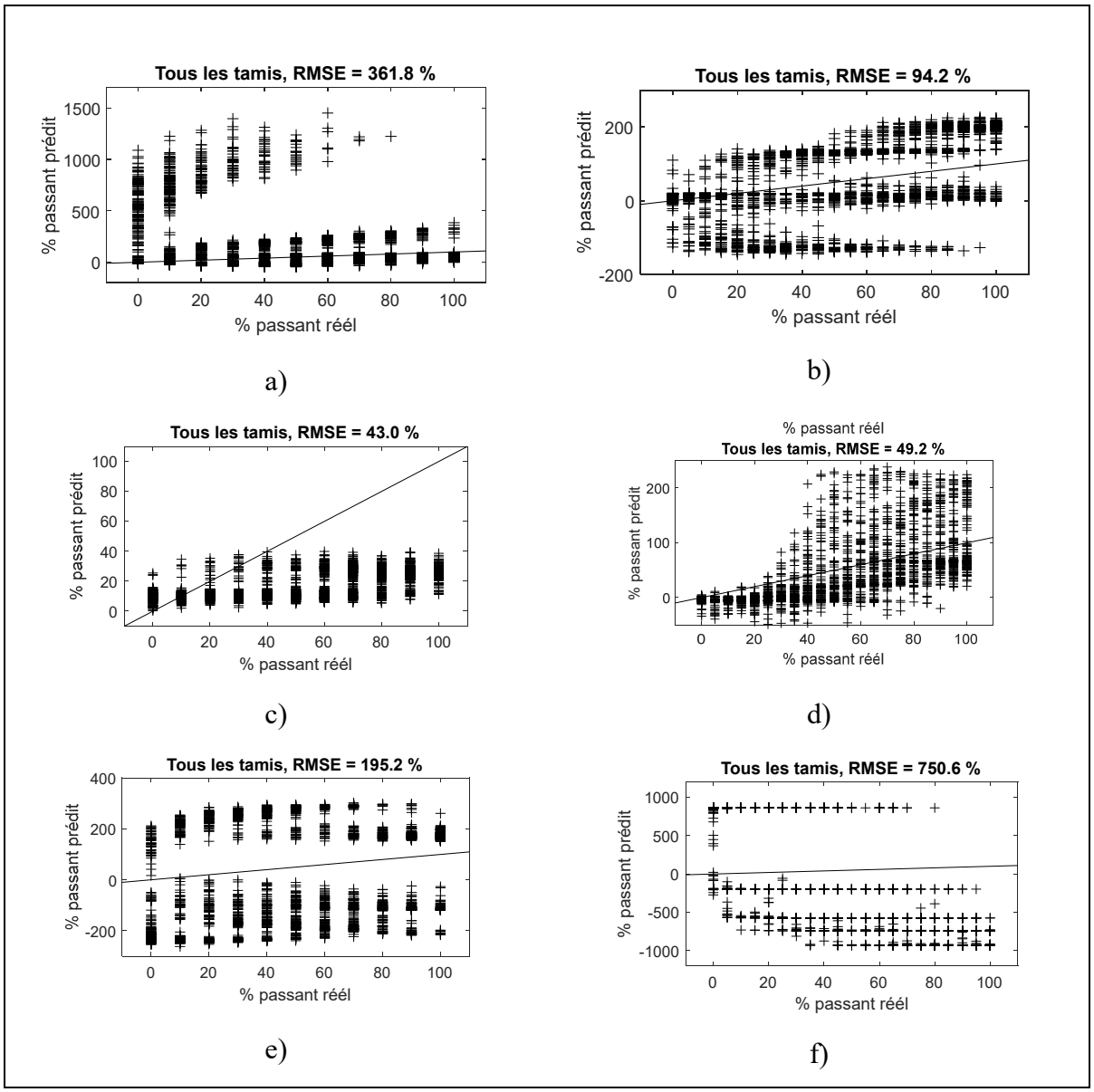


Figure 4.13 Résultats des images de Yade dans le réseau de Unity pour a) l'entropie, c) l'ondelette, e) les textures d'Haralick, et des images de Unity dans le réseau de Yade pour b) l'entropie, d) l'ondelette, f) les textures d'Haralick

Pour les images de Unity+Yade, il était attendu que les résultats soient de moins bonnes performances alors qu'elles sont proches de celles de Unity. Ceci montre qu'on perd peu de précision en généralisant.

Pirnia et al. (2018) ont travaillé sur une base de données contenant 53130 images de Yade. Dans ces travaux les images ont été utilisée directement dans un réseau de neurones à convolution. Le réseau de neurones utilisé a pour but de prédire le pourcentage passant pour les mêmes tamis soient les tamis 710, 425 µm, 250, 150 et 106 µm. Les résultats qu'ils ont obtenus se trouvent au niveau du tableau 4.4 :

Tableau 4.4 Résultats obtenus par Pirnia et al. (2018)

Taille tamis (µm)	710	425	250	150	106	Tous les tamis
RMSE (%)	9,15	8,4	6,3	5,2	4,2	6,9

Manashti et al. (2020) ont eux aussi utilisé la même base de données que Pirnia et al. (2018) en poussant les résultats plus loin : Ils ont utilisé les mêmes tamis, les mêmes images mais ont extrait un total de 14 paramètres texturaux avec lesquels ils ont entrainé un réseau de neurones. Parmi ces paramètres se trouve l'entropie, l'ondelette et les textures d'Haralick dont on a représenté les résultats au niveau du Tableau 4.5.

Tableau 4.5 Résultats obtenus par Manashti et al. (2020)

Texture	**Entropie**	**Ondelette**	**Haralick**
RMSE max	6,8	8,8	6,8
RMSE moyen	6,8	6,6	5,8
RMSE min	5,5	3,5	3,0

En comparant les résultats obtenus avec les travaux de Pirnia et al. (2018) on trouve dans l'ensemble que l'ordre de grandeur est le même pour les images de Unity + Yade (cas des textures d'Haralick avec RMSE max de 7,4 %, un RMSE pour tous les tamis de 6,5 %, et un RMSE min de 3,6 %), un RMSE plus bas pour les images de Yade (cas de l'entropie avec RMSE max de 7,9 %, un RMSE pour tous les tamis de 7,1 %, et un RMSE min de 4,6 %), et un RMSE plus élevé pour les images de Unity (cas de l'entropie RMSE max de 9,6 %, un

RMSE pour tous les tamis de 7,5 %, et un RMSE min de 3,3 %). Ceci prouve bien que le choix de la base de données et des paramètres texturaux peut bien améliorer les valeurs du RMSE. Il y a des textures qui marchent mieux avec certaines images et d'autres moins que le fait d'utiliser des images directement dans un réseau de neurones.

Toujours en comparant avec les résultats de Pirnia et al. (2018) on remarque qu'il est possible d'obtenir de meilleurs résultats en prenant une texture comme entrée du réseau de neurones au lieu des images directement. Les résultats des textures d'Haralick obtenus avec les images de Yade en témoignent (RMSE max de 7,4 %, un RMSE pour tous les tamis de 6,5 %, et un RMSE min de 3,6 %). Ce constat s'affirme en comparant les résultats de Manashti et al. (2020) avec celles de Pirnia et al. (2018) et même celles obtenues dans ces travaux. Les résultats de Manashti et al. (2020) ont donné un RMSE moyen de 5,8 % pour les textures d'Haralick avec les images de Yade.

Les résultats obtenus par Manashti et al. (2020) montrent que pour l'entropie et l'ondelette les RMSE moyen étaient proches (6,8 % et 6,6 %) et que le RMSE moyen des textures d'Haralick était plus faible (5,8 %). Ceci est aussi le cas des résultats obtenus avec la base de données de Yade. L'entropie et les ondelettes avaient des RMSE moyens proches (7,2 % et 7,4 %) et les textures d'Haralick un RMSE moyen plus faible (6,5 %). Néanmoins les résultats obtenus durant ces travaux avec Yade sont moins bons que ceux de Manashti et al. (2020). Ceci s'explique par le fait que ces derniers ont utilisé une base de données de 53130 images de Yade au lieu des 3003 qu'on a utilisés. Le nombre plus élevé d'images dans la base de données a permis d'améliorer l'apprentissage dans ce cas-ci. En revanche il est à noter qu'avoir plus d'images ne signifie pas forcément avoir un RMSE plus faible.

Une comparaison d'ordre générale avec ces travaux permet de situer les images créées avec Unity. On trouve que les résultats obtenus suivent la même allure : le RMSE moyen est du même ordre et plus la taille du tamis augmente moins la prédiction est précise et inversement. Les valeurs maximales et minimales sont du même ordre que celles obtenues. Petite mention aux textures d'Haralick avec les images qui donnent même des meilleurs RMSE. Finalement

les images construites avec Unity sont bonnes et significatives : Elles montrent qu'elles ont bien été créé en respectant le critère de granulométrie imposé, et que le fait qu'elles soient plus réelles avec plus de ségrégation peut expliquer leur RMSE qui est un peu plus élevé.

Les résultats obtenus par Manashti et al. (2020) sont légèrement meilleurs que ceux qu'on a obtenus avec Yade et présentent la même allure que ceux obtenus avec Unity et Unity + Yade. Ces résultats ont été obtenus sur une base de 53130 images au lieu de 3003 et 6006 et avec des réseaux de neurones plus élaborés. Ainsi il est possible de mentionner que la base de données créée avec Unity respecte les critères de granulométrie imposés et peut être utilisée pour prédire le pourcentage passant de sols à partir d'images.

CONCLUSION

Ce projet de maitrise traite la création d'une base de données de sols réalistes avec la plateforme de jeux Unity puis l'utilisation de ces images dans un réseau de neurones artificiels dans le but de vérifier l'efficacité de ces images. Cette section présente les conclusions générales et certaines limitations concernant les résultats obtenus pour chaque objectif.

1- Création de la base de données avec Unity

Les résultats ont montré qu'il est possible de créer, avec Unity, une base de données d'images de matériaux granulaires dont la granulométrie est imposée. Cette base de données contient les vues de dessus, de dessous et les images de profondeur pour 3003 granulométries. Les images de profondeur sont utiles pour mettre en valeur le chevauchement et la ségrégation comme l'a montré Thurley (2010), chose qui ne fait pas l'objet de ce mémoire. Les granulométries utilisées ont été choisies en fonction des tamis 75, 106, 150, 250, 425, 710 et 1180 µm. Un fichier texte contenant la taille de toutes les particules a été créé dans le but de vérifier que le programme fonctionnait bien.

Il est fort de constater que la plateforme Unity a un degré de réalisme assez élevé. Les roches ont été sélectionnées de l'asset store et modifiées de manière à ce qu'elles représentent au mieux la réalité. Les caractéristiques ont été choisies en fonction de la forme, la couleur, l'ombre et les caractéristiques physiques. Ensuite la dualité interface-script que présente Unity a été exploitée dans le but de mettre en place des scripts permettant de réaliser au mieux les tâches nécessaires : que chaque image soit générée en respectant une granulométrie donnée, que les particules tombent correctement dans la boîte, et que les images soient prises une fois que les particules sont tombées. Finalement on s'est assuré que le programme fonctionne correctement et que pour chaque image il y avait toujours autant de détails.

L'idée de vouloir chercher le détail à chaque fois n'est pas sans conséquences en ce qui concerne le temps de calcul. A cet égard nous avons essayé de changer certaines fonctions en

s'orientant vers une approche plus dynamique (fonction *Pool* et *Instantiate*) au lieu de l'approche statique. Ces changements ont amélioré les performances, notamment en mémoire allouée pendant le traitement des opérations et en temps de calcul. Malgré ces modifications, le temps de calcul a atteint deux heures par courbe granulométrique pour les spécimens les plus fins (100 000 particules et plus). Le nombre d'images a dû être limité à 3003. Parmi les éléments qui étaient très demandant en mémoire, on cite le maillage convexe qui a été ajouté à chaque particule dans but d'avoir des empilements de particules réalistes.

Les dimensions utilisées pour faire la simulation avec Unity sont petites et ont été choisies afin qu'elles donnent de bonnes images dans le réseau de neurones. La boîte de 5 mm × 5 mm × 5 mm devait contenir pleinement des particules dont la taille varie de 75 à 1180 µm, la masse totale est de 50 mg. Là aussi même si la possibilité d'améliorer la conception a été étudiée, la simulation ne pouvait être plus rapide à cause des calculs très demandant en mémoire.

2- Extraction des caractéristiques et utilisation dans un réseau de neurones

Un traitement a été fait sur ces images. Elles sont passées de 1853×1025 pixels à un carré de 128×128 en niveaux de gris. Ensuite les images du haut et du bas ont été combinées en une seule entité de 256×128. Finalement les niveaux de gris ont été atténués. Par la suite ces images ont été exploitées pour en extraire l'entropie (18 paramètres), les ondelettes (49 paramètres) et les textures d'Haralick (14 paramètres). Ces caractéristiques ont été choisies pour faciliter les comparaisons avec la littérature.

Des réseaux de neurones ont été créés pour prédire les pourcentages passants pour les tamis 106, 150, 250, 425, et 710 µm à partir des caractéristiques mentionnées précédemment. Du fait que le travail s'est focalisé sur l'entropie, les ondelettes et les textures d'Haralick qui sont des textures, il n'a pas été nécessaire d'utiliser un réseau de neurones avec plusieurs couches, un réseau peu profond a suffi amplement. Afin que les résultats soient pertinents trois bases de données différentes ont été utilisées : celle qui a été conçue avec Unity dans ce projet, celle de Pirnia et al. (2018) avec Yade, et une dernière qui constitue un mélange des deux. Les deux

premières ont un total de 3003 images alors que la dernière contient 6006 images. Le RMSE a été calculé à la fin pour comparer le pourcentage passant prédit au pourcentage passant réel.

L'entraînement des réseaux de neurones avec ces trois bases de données a mis en lumière plusieurs points. D'abord les résultats montrent que les trois bases de données donnaient de bonnes prédictions. La courbe de prédiction était linéaire avec un RMSE allant de 6,50 % à 9,90 %. Les images de Yade donnaient un plus faible RMSE avec une moyenne de 7,03 % devant les images de Unity+Yade (8,40 %) et celles de Unity (8,46 %). Cette différence s'explique par le fait les images de Yade sont synthétiques et donc non réel comme celle de Unity, ainsi que la présence de plus de ségrégation dans les images de Unity. En termes de textures l'entropie a donné des résultats légèrement meilleurs que les ondelettes et les textures d'Haralick. Ceci s'explique par le fait que les textures d'Haralick prennent en compte la luminosité ce qui n'est pas le cas des deux autres. Ensuite, en comparant les résultats à ceux de Pirnia et al. (2018) et Manashti et al. (2020) dont les travaux ont des points communs, on comprend que la base de données construite par Unity est fiable et peut être utilisée pour d'autres travaux. Finalement, on se référant à Manashti et al. (2020), dont l'apparition de l'article à coïncider avec la rédaction de ce mémoire, on pense qu'il aurait été possible d'utiliser d'autres textures afin d'améliorer les pourcentages prédits par le réseau de neurones.

Pour conclure, ce projet se démarque essentiellement par deux grandes idées. D'une part par le fait d'avoir eu recours à une plateforme de jeux pour créer les particules, ceci rend le projet innovant. D'une autre part en créant une base de données importante et réaliste qu'on peut exploiter à la fois pour les approches déterministes et statistiques. Ce projet constitue aussi une étape supplémentaire pour la granulométrie par analyse d'images. Avec cette idée, nous avons apportés une solution quant aux bases de données qui existaient déjà : soit elles contenaient des images de vrais sols mais avec un très petit nombre, soit elles contenaient beaucoup d'images mais les particules étaient générées et avaient des formes sphériques parfaites. Maintenant il ne reste plus qu'à faire un pas de plus en améliorant le rendu afin qu'il soit le plus réel possible.

RECOMMANDATIONS

Cette section propose des recommandations concernant ce qui a été traité en ouvrant la porte à des axes de recherche qui pourraient être développés dans les projets futurs.

1- Accélérer la simulation avec Unity

Comme mentionné précédemment, il est vrai que les images avec Unity sont réalistes et représentent au mieux un sol naturel, en revanche elles nécessitent de l'espace en mémoire. Ce problème s'explique par 3 raisons : 1- La présence de plusieurs scripts qui utilisent des fonctions dites statiques, 2- La présence d'un nombre très grand de particules atteignant l'ordre des 100 000 par image, 3- La présence du maillage convexe qui consomme énormément de mémoire. Même si la première et la deuxième raison a vu des modifications en mettant plusieurs fonctions dites dynamiques permettant d'alléger le code et de générer des particules instantanément, il n'en n'a pas été de même avec le troisième cas qui n'a pas pu être amélioré. Il serait donc fort intéressant de trouver une configuration qui permet d'éviter au maximum le maillage convexe comme par exemple ajouter le maillage sur les particules à travers un code avant que celles-ci ne rentrent dans la boîte.

2- Étendre jusqu'au tamis 4,75 mm

Ce mémoire se focalise sur les matériaux granulaires dont la taille varie de 75 à 1180 µm. D'un point de vue granulométrique, le tamisage se fait en deux parties : de 75 µm à 4,75 mm et de 4,75 mm à 75 mm. Le tamis 75 µm correspond à la limite inférieure du sable alors que la limite supérieure, est fixée à 4,75 mm et représente la frontière entre le sable et le gravier. Le tamis 1180 µm a été utilisé pour restreindre le nombre de particules. En effet comme le nombre de particules est proportionnel au volume, augmenter la limite supérieure de 1180 à 4750 µm, c'est-à-dire quatre fois, augmenterait le nombre de particules fines par un facteur $4^3 = 65$ pour la courbe granulométrique la plus fine. Ainsi augmenter le tamis supérieur et élargir la granulométrie jusqu'à 4,75 mm pourrait être un axe de recherche. Ceci pourrait améliorer la

base de données des matériaux granulaires et la rendre plus cohérente avec le tamisage qui est réalisé en pratique.

3- Générer plus d'images avec le GAN

Le GAN (*Generative Adversarial Networks*), est une approche récente qui est apparue dans les travaux de Goodfellow et al. (2014) et qui permet de générer des images très réalistes. Le principe consiste à utiliser sur des images deux réseaux de neurones artificiels : un générateur et un discriminateur. Le premier comme son nom l'indique va générer des images à partir d'une base de données. Le second va recevoir ces images générées ainsi que les vraies images de la base de données, et a pour but de détecter le vrai du faux. Par la suite un retour est fait de la manière suivante : le générateur reçoit les images que le discriminateur a démasquées ou qu'il a trouvé fausses, le discriminateur quant à lui reçoit les images sur lesquelles il s'est trompé. Et le processus continue dans ce sens de façon à ce que l'apprentissage s'améliore étape par étape. L'avantage du GAN est qu'il permet de générer des images très réalistes à partir d'une base de données. Cette technique, même si elle est récente, commence à faire ces preuves (Huang, Yu, & Wang, 2018) comme par exemple avec les images de visages. Il serait fort intéressant de voir ce que pourrait apporter le GAN aux images de sols granulaires en général, et celles obtenues avec Unity en particulier, surtout que comme on l'a déjà mentionné il y a un manque dans les jeux de données de sols réalistes.

4- Améliorer le réseau de neurones

Améliorer les réseaux de neurones serait un autre point sur lequel pourrait porter les futurs travaux. En effet dans ce mémoire les réseaux de neurones ont été construits dans le but de vérifier les résultats obtenus avec les images de Unity en utilisant l'entropie, l'ondelette et les textures d'Haralick. Le réseau de neurones qui avait pour objectif de prédire le pourcentage passant en fonction de ces caractéristiques a donné un RMSE global aux alentours de 7 %. Il serait possible d'envisager d'améliorer les résultats en utilisant d'autres hyper paramètres; changer le nombre de couches, la fonction d'activation, le gradient stochastique, la rétro

propagation etc. On serait curieux de voir un réseau utiliser ces images en entrée pour voir ce qu'il serait possible d'extraire en plus, surtout qu'il reste à la portée un fichier texte contenant l'ensemble des particules qu'on a générées.

5- Utiliser les images de profondeur

Quand la base de données avec Unity a été générée, des images de profondeur ont été créées. Ces images, même si elles n'ont pas été utilisées dans le réseau de neurones servent à détecter les erreurs de chevauchement. Cette erreur s'explique par le fait que les grosses particules vont cacher les petites, créant ainsi un chevauchement. Thurley (2011) a montré qu'il était possible d'utiliser les images de profondeur sur des piles de roches pour repérer les défauts sur ces roches lors d'un contrôle qualité. Ces travaux ont même permis de proposer quelques solutions pour les erreurs de chevauchements sans pour autant résoudre totalement le problème. Il serait de bonne augure d'utiliser les images de profondeur de Unity pour améliorer les erreurs de chevauchement ce qui pourrait améliorer la qualité de la base de données construite.

6- Utiliser d'autres textures

Dans le but de vérifier les images de Unity à travers le réseau de neurones les textures suivantes ont été utilisées : l'entropie, les ondelettes et les textures d'Haralick. Comme ce qui a été déjà mentionné, ces textures ont été utilisées parce qu'elles sont fréquemment utilisées et donnent de bons résultats. Par la suite en cherchant à donner un avis critique sur la chose, les résultats obtenus ont été comparés à ceux de Manashti et al. (2020), article qui est apparu au moment de la rédaction de ce mémoire, qui a utilisé plusieurs textures sur un réseau de neurones et qui a prédit le pourcentage passant de sols granulaires. Manashti et al. (2020) ont utilisé d'autres paramètres texturaux comme le LCP (*Local Configuration Pattern*) et le CLBP (Completed modeling of Local Binary Pattern) qui ont donné un RMSE plus faible. Utiliser ces caractéristiques sur les images de Unity pourrait donner des résultats prometteurs.

ANNEXE I

LES TEXTURES D'HARALICK

Les textures d'Haralick sont au nombre de 13. Il existe cependant une quatorzième texture dont la détermination se fait par déduction des treize autres. Elle sera ajoutée à titre informatif. Les textures sont :

Le moment angulaire $f_1 = \sum_i \sum_j p(i,j)^2$

Le contraste $f_2 = \sum_{n=0}^{Ng-1} n^2 \left\{ \sum_{i=1}^{Ng} \sum_{j=1}^{Ng} p(i,j) / \left| i-j=n \right| \right\}$

La corrélation $f_3 = \dfrac{\sum_i \sum_j (ij)\, p(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y}$

La variance $f_4 = \sum_i \sum_j (i-\mu)^2 \, p(i,j)$

Le moment de la différence inversée $f_5 = \sum_i \sum_j \dfrac{1}{1+(i-j)^2} \, p(i,j)$

La somme des moyennes $f_6 = \sum_{i=2}^{2Ng} i p_{x+y}(i)$

La somme des $\mathrm{var}\,iances$ $f_7 = \sum_{i=2}^{2Ng} (i-f_8)^2 \, p_{x+y}(i)$

La somme des entropies $f_8 = -\sum_{i=2}^{2Ng} p_{x+y}(i) \log\left\{ p_{x+y}(i) \right\}$

L'entropie $f_9 = -\sum_i \sum_j p(i,j) \log(p(i,j))$

La différence des $\mathrm{var}\,iances$ $f_{10} = \mathrm{var}\,iance \;\; de \;\; p_{x-y}$

La différences des entropies $f_{11} = -\sum_{i=0}^{Ng-1} p_{x-y}(i) \log\left\{ p_{x-y}(i) \right\}$

La mesure des corrélations $f_{12} = \dfrac{HXY - HXY1}{\max\{HX, HY\}}$

et $f_{13} = (1 - \exp[-2(HXY2 - HXY)])^{\frac{1}{2}}$

Le coefficient $\max imal$ *de corrélation* $f_{14} = 2ème$ *plus grande valeur propre de* $Q^{\frac{1}{2}}$

avec $Q(i,j) = \sum_k \dfrac{p(i,k)p(j,k)}{p_x(i)p_y(k)}$

sachant que :

- $p(i,j)$ est la matrice normalisée de la GLCM;
- μ_x μ_y σ_x σ_y sont respectivement les moyennes et les déviations standards de p_x et p_y
- $p_{x+y}(k) = \sum_{i=1}^{Ng}\sum_{j=1}^{Ng} p(i,j) \mid i+j=k, \quad k = 2,3,..,2N_g$;
- $p_{x-y}(k) = \sum_{i=1}^{Ng}\sum_{j=1}^{Ng} p(i,j) \mid \lvert i-j=k\rvert, \quad k = 0,1,..,N_g - 1$;
- $HXY = -\sum_i \sum_j p(i,j)\log(p(i,j))$;
- $HXY1 = -\sum_i \sum_j p(i,j)\log(p_x(i)p_y(j))$;
- $HXY2 = -\sum_i \sum_j p_x(i)p_y(j)\log(p_x(i)p_y(j))$;
- HX et HY les entropies respectives de p_x et p_y.

ANNEXE II

LES SCRIPTS DE UNITY

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using System.Linq;
4   using UnityEngine;
5
6   public class SceneController : MonoBehaviour
7   {
8       public ImageSynthesis synth;
9       public GameObject[] prefabs;
10      public int minObjects = 100;
11      public int maxObjects = 200;
12      public float tailleMin = 0.075f;
13      public float tailleMax = 0.106f;
14      //public List<int> taille = new List<int>();
15
16      private ShapePool pool;
17      public int frameCount = 0;
18      public int nb = 0;
19      //private int i = 0;
20      private int palier = 20;
21
22
23
24      // Start is called before the first frame update
25      void Start()
26      {
27          pool = ShapePool.Create(prefabs);
28          //taille.Add(0.425)
29
30      }
31
32      // Update is called once per frame
33      void Update()
34      {
35          //if (Enumerable.Range(0, 4).Contains(nb-1))
36          //{
37              //palier = 200;
38          //}
39          //else if (Enumerable.Range(5, 20).Contains(nb-1))
40          //{
41              //palier = 150;
42          //}
43          //else
44          //{
45              //palier = 20;
46          //}
47
48          if (frameCount % palier == 0)
49          {
50              GenerateRandom();
51              nb++;
52          }
53          frameCount++;
54          if (frameCount % palier == 0)
```

```
55          {
56              //string filename = $"image_{frameCount.ToString().PadLeft(5, '0')}";
57              string filename = (nb-1).ToString();
58              synth.Save(filename, 1853, 1025, "captures", 3);
59              Debug.Log(frameCount);
60          }
61
62      }
63
64      void GenerateRandom()
65      {
66          pool.ReclaimAll();
67          int objectsThisTime = Random.Range(minObjects, maxObjects);
68          for (int i = 0; i < objectsThisTime; i++)
69          {
70              //Pick out a prefab
71              int prefabIndx = Random.Range(0, prefabs.Length);
72              GameObject prefab = prefabs[prefabIndx];
73
74              //Position
75              float newX, newY, newZ;
76              newX = Random.Range(-2.4f, 2.4f);
77              newY = Random.Range(4.0f, 15.0f);
78              newZ = Random.Range(-2.4f, 2.4f);
79              Vector3 newPos = new Vector3(newX, newY, newZ);
80
81              //Rotation
82              var newRot = Random.rotation;
83
84              var shape = pool.Get((ShapeLabel)prefabIndx);
85              var newObj = shape.obj;
86              newObj.transform.position = newPos;
87              newObj.transform.rotation = newRot;
88
89              //Scale
90              float sx = Random.Range(tailleMin, tailleMax);
91              float sy = Random.Range(tailleMin, tailleMax);
92              float sz = Random.Range(tailleMin, tailleMax);
93              Vector3 newScale = new Vector3(sx, sy, sz);
94              newObj.transform.localScale = newScale;
95
96              //Color
97              //float newR, newG, newB;
98              //newR = Random.Range(0.0f, 1.0f);
99              //newG = Random.Range(0.0f, 1.0f);
100             //newB = Random.Range(0.0f, 1.0f);
101             //var newColor = new Color(newR, newG, newB);
102             //newObj.GetComponentInChildren<Renderer>().material.color = newColor;
103
104         }
105         synth.OnSceneChange();
106     }
107
108 }
```

Figure-A II-1 Générer les particules et prendre une photo du haut

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public enum ShapeLabel { Rock1, Rock2, Rock3, Rock4, Rock5, Rock6, Rock7, Rock8, Rock9, Rock10};
6
7   public class Shape
8   {
9       public ShapeLabel label;
10      public GameObject obj;
11  }
12
13  public class ShapePool : ScriptableObject
14  {
15      private GameObject[] prefabs;
16      private Dictionary<ShapeLabel, List<Shape>> pools;
17      private List<Shape> active;
18
19      public static ShapePool Create(GameObject[] prefabs)
20      {
21          var p = ScriptableObject.CreateInstance<ShapePool>();
22          p.prefabs = prefabs;
23          p.pools = new Dictionary<ShapeLabel, List<Shape>>();
24          for (int i = 0; i < prefabs.Length; i++)
25          {
26              p.pools[(ShapeLabel)i] = new List<Shape>();
27          }
28          p.active = new List<Shape>();
29          return p;
30      }
31
```

```
32      public Shape Get(ShapeLabel label)
33      {
34          var pool = pools[label];
35          int lastIndex = pool.Count - 1;
36          Shape retShape;
37          if (lastIndex <=0)
38          {
39              var obj = Instantiate(prefabs[(int)label]);
40              retShape = new Shape() { label = label, obj = obj };
41          } else
42          {
43              retShape = pool[lastIndex];
44              retShape.obj.SetActive(true);
45              pool.RemoveAt(lastIndex);
46          }
47          active.Add(retShape);
48          return retShape;
49      }
50
51      public void ReclaimAll()
52      {
53          foreach (var shape in active)
54          {
55              shape.obj.SetActive(false);
56              pools[shape.label].Add(shape);
57          }
58          active.Clear();
59      }
60
61  }
62
```

Figure-A II-2 accélérer le modèle

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Masse : MonoBehaviour
6   {
7       private float Masse_Vol = 2.5e-06f;
8       public float masse;
9       public Vector3 boxScale;
10
11      // Start is called before the first frame update
12      void Start()
13      {
14
15      }
16
17      // Update is called once per frame
18      void Update()
19      {
20          masse = GetComponent<Rigidbody>().mass;
21          boxScale = GetComponent<Transform>().localScale;
22
23          float volume = 4 * Mathf.PI * Mathf.Pow(((boxScale.x + boxScale.y + boxScale.z) / 3) / 2, 3) / 3;
24          masse = Masse_Vol * volume;
25          GetComponent<Rigidbody>().mass = masse;
26      }
27  }
28
```

Figure-A II-3 Changer la masse

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Granulometrie : MonoBehaviour
{
    //List<Quest> quests = new List<Quest>();

    SceneController sc;
    SceneController sc1;
    SceneController sc2;
    SceneController sc3;
    SceneController sc4;
    SceneController sc5;
    private float Masse_tot = 5e-05f; //en kg
    private float Volume_part;
    private float Volume_part1;
    private float Volume_part2;
    private float Volume_part3;
    private float Volume_part4;
    private float Volume_part5;
    private float Masse_Vol = 2.5e-06f; //en kg/mm3
    private int palier1 = 19;



    // Use this for initialization
    void Start()
    {
        //volume en mm3
        Volume_part = 4 * Mathf.PI * Mathf.Pow((1.18f+0.71f)/4, 3) / 3;
        //Debug.Log(Volume_part*0.000001f);
        Volume_part1 = 4 * Mathf.PI * Mathf.Pow((0.71f + 0.425f) / 4 , 3) / 3;
        Volume_part2 = 4 * Mathf.PI * Mathf.Pow((0.425f + 0.25f) / 4 , 3) / 3;
        Volume_part3 = 4 * Mathf.PI * Mathf.Pow((0.25f + 0.15f) / 4, 3) / 3;
        Volume_part4 = 4 * Mathf.PI * Mathf.Pow((0.15f + 0.106f) / 4 , 3) / 3;
        Volume_part5 = 4 * Mathf.PI * Mathf.Pow((0.106f + 0.075f) / 4, 3) / 3;

        //Debug.Log(data[1]);
        //Debug.Log(data[1].IndexOf(','));
        //Debug.Log(data[1].Substring(0, (data[1].IndexOf(','))));

    }

    // Update is called once per frame
    void Update()
    {
        sc = GameObject.Find("SceneControl").GetComponent<SceneController>();
        sc1 = GameObject.Find("SceneControl (1)").GetComponent<SceneController>();
        sc2 = GameObject.Find("SceneControl (2)").GetComponent<SceneController>();
        sc3 = GameObject.Find("SceneControl (3)").GetComponent<SceneController>();
        sc4 = GameObject.Find("SceneControl (4)").GetComponent<SceneController>();
        sc5 = GameObject.Find("SceneControl (5)").GetComponent<SceneController>();

```

```
55
56
57          if (sc.frameCount % palier1 == 0)
58          {
59              TextAsset Granulometrie = Resources.Load<TextAsset>("Granulometrie");
60              string[] data = Granulometrie.text.Split(new char[] { '\n' });
61
62              string ch = string.Empty;
63              string ch1 = string.Empty;
64              string ch2 = string.Empty;
65              string ch3 = string.Empty;
66              string ch4 = string.Empty;
67              string ch5 = string.Empty;
68
69
70
71
72              //sc.nb = sc.nb + 50;
73              Debug.Log(data[sc.nb]);
74
75              ch = data[sc.nb].Substring(0, (data[sc.nb].IndexOf(';')));
76              data[sc.nb] = data[sc.nb].Substring((data[sc.nb].IndexOf(';') + 1), (data[sc.nb].Length - data[sc.nb].IndexOf(';') - 1));
77              int pourc_particule = System.Convert.ToInt32(ch);
78              Debug.Log(pourc_particule);
79
80              ch1 = data[sc.nb].Substring(0, (data[sc.nb].IndexOf(';')));
81              data[sc.nb] = data[sc.nb].Substring((data[sc.nb].IndexOf(';') + 1), (data[sc.nb].Length - data[sc.nb].IndexOf(';') - 1));
82              Debug.Log(data[sc.nb]);
83              int pourc_particule1 = System.Convert.ToInt32(ch1);
84              Debug.Log(pourc_particule1);
85
86              ch2 = data[sc.nb].Substring(0, (data[sc.nb].IndexOf(';')));
87              data[sc.nb] = data[sc.nb].Substring((data[sc.nb].IndexOf(';') + 1), (data[sc.nb].Length - data[sc.nb].IndexOf(';') - 1));
88              int pourc_particule2 = System.Convert.ToInt32(ch2);
89              Debug.Log(data[sc.nb]);
90              Debug.Log(pourc_particule2);
91
92              ch3 = data[sc.nb].Substring(0, (data[sc.nb].IndexOf(';')));
93              data[sc.nb] = data[sc.nb].Substring((data[sc.nb].IndexOf(';') + 1), (data[sc.nb].Length - data[sc.nb].IndexOf(';') - 1));
94              int pourc_particule3 = System.Convert.ToInt32(ch3);
95              Debug.Log(pourc_particule3);
96
97              ch4 = data[sc.nb].Substring(0, (data[sc.nb].IndexOf(';')));
98              data[sc.nb] = data[sc.nb].Substring((data[sc.nb].IndexOf(';') + 1), (data[sc.nb].Length - data[sc.nb].IndexOf(';') - 1));
99              int pourc_particule4 = System.Convert.ToInt32(ch4);
100             Debug.Log(pourc_particule4);
101
102             ch5 = data[sc.nb].Substring(0, (data[sc.nb].IndexOf(';')));
103             data[sc.nb] = data[sc.nb].Substring((data[sc.nb].IndexOf(';') + 1), (data[sc.nb].Length - data[sc.nb].IndexOf(';') - 1));
104             int pourc_particule5 = System.Convert.ToInt32(ch5);
105             Debug.Log(pourc_particule5);
106
107
```

```
108
109             //Debug.Log(ch1 + ch2);
110
111
112             float var = (pourc_particule - pourc_particule1) * Masse_tot / (100 * Masse_Vol * Volume_part);
113             sc.minObjects = (int)var ;
114             sc.maxObjects = sc.minObjects;
115
116             float var1 = (pourc_particule1 - pourc_particule2 ) * Masse_tot / (100 * Masse_Vol * Volume_part1);
117             sc1.minObjects = (int)var1 ;
118             sc1.maxObjects = sc1.minObjects;
119
120             float var2 = (pourc_particule2 - pourc_particule3 ) * Masse_tot / (100 * Masse_Vol * Volume_part2);
121             sc2.minObjects = (int)var2 ;
122             sc2.maxObjects = sc2.minObjects;
123
124             float var3 = (pourc_particule3 - pourc_particule4 ) * Masse_tot / (100 * Masse_Vol * Volume_part3);
125             sc3.minObjects = (int)var3 ;
126             sc3.maxObjects = sc3.minObjects;
127
128             float var4 = (pourc_particule4 - pourc_particule5) * Masse_tot / (100 * Masse_Vol * Volume_part4);
129             sc4.minObjects = (int)var4 ;
130             sc4.maxObjects = sc4.minObjects;
131
132             float var5 = pourc_particule5 * Masse_tot / (100 * Masse_Vol * Volume_part5);
133             sc5.minObjects = (int)var5 ;
134             sc5.maxObjects = sc5.minObjects;
135
136
137
138
139
140             palier1 = palier1 + 20;
141         }
142
143     }
144 }
145
```

Figure-A II-4 Affecter la granulométrie

```
4
5   public class ChangeColor : MonoBehaviour
6   {
7       Renderer Objectcolor;
8       Renderer Object_Children_color;
9       private Color newColor;
10
11      // Start is called before the first frame update
12      void Start()
13      {
14          float newR, newG, newB, newGrey;
15          newR = Random.Range(0.0f, 1.0f);
16          newG = Random.Range(0.0f, 1.0f);
17          newB = Random.Range(0.0f, 1.0f);
18          newGrey = (newR + newG + newB) / 3;
19          //newColor = new Color(newR, newG, newB);
20          newColor = new Color(newGrey , newGrey, newGrey);
21          Objectcolor = GetComponent<Renderer>();
22          //Object_Children_color = GetComponentInChildren<Renderer>();
23      }
24
25      // Update is called once per frame
26      void Update()
27      {
28          //Objectcolor.material.color = Color.red;
29          //Objectcolor.material.color = Object_Parent_color.material.color;
30          //Objectcolor.material.color = Color.Lerp(Color.white, Color.black, Mathf.PingPong(Time.time, 1));
31          //Objectcolor.material.color = Color.Lerp(Color.white, Color.black, Random.Range(0, 0));
32          //Objectcolor.material.color = Color.Lerp(Color.white, Color.black, 0);
33          Objectcolor.material.SetColor("_Color", newColor);
34
35      }
36  }
37
```

Figure-A II-5 Donner une couleur réaliste

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
 //using System.IO;

public class TakeScreenshot : MonoBehaviour
{
    SceneController sc;
    public Camera MainCamera;
    public Camera LowerCamera;
    //string sourceFile;
    //string destinationFile;
    private int palier1 = 19;

    // Start is called before the first frame update
    void Start()
    {
        MainCamera.enabled = true;
        LowerCamera.enabled = false;
    }


    // Update is called once per frame
    void Update()
    {
        sc = GameObject.Find("SceneControl").GetComponent<SceneController>();
        if (sc.frameCount % palier1 == 0)
        {
            MainCamera.enabled = false; LowerCamera.enabled = true;
            TakeScreenShoot();
            //File.Move(sourceFile, destinationFile);
            palier1 = palier1 + 20;
        }

        void TakeScreenShoot()
        {
            ScreenCapture.CaptureScreenshot("capture" + (sc.nb - 1).ToString() + ".png");

        }



    }
}

```

Figure-A II-6 Prendre une capture du bas

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;

public class SaveData : MonoBehaviour
{

    public Vector3 boxSize;
    public Vector3 boxScale;
    public Vector3 Objectposition;
    //public int nb = 1;
    //private int frameCount = 0;
    SceneController sc;
    //private int palier = 0;
    //List<int> palier = new List<int>() {0 };
    string nom;
    string numero;
    string image;
    private int palier = 20;



    // Start is called before the first frame update
    void Start()
    {
        // Determine les caracteristiques de l'objet

        nom = GetComponent<Transform>().name;

    }



    // Update is called once per frame
    void Update()
    {
        boxScale = GetComponent<Transform>().localScale;
        Objectposition = GetComponent<Transform>().position;
        sc = GameObject.Find("SceneControl").GetComponent<SceneController>();
        //StartCoroutine(pause());
        //Debug.Log("attente terminado");

        // enregistrer en appuyant sur S
        //if (Input.GetKeyDown(KeyCode.S))
        if (sc.frameCount % palier ==  0 )
        {

            //numero = nb.ToString();
            if (//Objectposition.x > -2.5 &&
                    //Objectposition.x < 2.5 &&
                    Objectposition.y > 0 )//&&
                    //Objectposition.z > -2.5 &&
                    //Objectposition.z < 2.5)
```

```
55                  {
56                      Save();
57
58                  }
59
60
61              }
62
63          }
64
65
66
67
68          void Save()
69          {
70
71
72              image = "image" +(sc.nb-1).ToString();
73
74
75              string[] content = new string[]
76              {
77                  image,
78                  nom,
79                  (boxScale.x).ToString(),
80                  (boxScale.y).ToString(),
81                  (boxScale.z).ToString()
82
83
84              };
85
86              //Accede au fichier
87              string fichier = Application.dataPath + "/data.txt";
88
89              //Prepare le texte a ecrire
90              string saveString = string.Join(";", content);
91
92              //Ecris sur le fichier
93              File.AppendAllText(fichier, "\r\n" + saveString);
94              Debug.Log("Sauvegarde effectuée");
95              Debug.Log(image);
96          }
97      }
98
```

Figure-A II-7 Enregistrer la taille des tailles des particules

ANNEXE III

DETERMINATION DE L'ENTROPIE LOCALE, L'ONDELETTE DE HAAR ET LES TEXTURES D'HARALICK

```
% Get the filenames of the top images. "img" is added to the name to
% exclude the depth images.
Filenames = dir('Unity images/Top/*img.png');

% The script then cycles through the filename vector
for i=1:size(Filenames,1)
    % Progress display
    fprintf('image %u\n',i)

    % Creates four strings for the relative paths and filenames (top,
    % bottom, combined, and combined and remapped images).
    TopFilename = strcat('Unity images/Top/',sprintf('%u_img.png',i)); %xxxx_img.png, without padding
    BottomFilename = strcat('Unity images/Bottom/',sprintf('capture%u.png',i)); %capturexxxx.png, without padding
    CombinedFilename = sprintf('Unity images/Combined/combined%05u.png',i); %combinedxxxxx.png, with padding
    CombinedRemappedFilename = sprintf('Unity images/CombinedRemapped/ccombinedremapped%05u.png',i); %combinedremappedxxxxx.png, with padding
    CombinedOriginalFilename = sprintf('Unity images/CombinedOriginal/ccombined%05u.png',i); %combinedremappedxxxxx.png, with padding

    % Loading the top and bottom images
    img1 = imread(TopFilename);
    img2 = imread(BottomFilename);

    % Convert the images grayscale
    img1 = rgb2gray(img1);
    img2 = rgb2gray(img2);

    % Images are cropped and resized to 128x128
    img1 = img1(266:765,680:1179); %(This results in a 500x500 image)
    img2 = img2(266:765,680:1179);
    img3 = [img1 img2];
    img1 = imresize(img1,[128 128]);
    img2 = imresize(img2,[128 128]);

    % Image are combined
    img1 = [img1 img2];

    % The file is written in the "Combined folder". The number stays the
    % same but padding with "0" is added before the number.
    imwrite(img1,CombinedFilename)
    imwrite(img3,CombinedOriginalFilename)
    img1 = imadjust(img1,[0 1],[0.2 1]);
    imwrite(img1,CombinedRemappedFilename)

end
```

Figure-A III-1 Traitement d'images

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computes the entropy features for the Unity dataset
%
% Author: Mehdi Temimi
% Date: 2020-08-08
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all

% The PSD are loaded from the Excel file. First column with filename is not
% read if present.
PSD_Unity=xlsread('Unity images/Granulométrie.xlsx');

% The script cycles through the filenames
Filenames = dir('Unity images/CombinedOriginal/*.png');
for i=1:size(Filenames,1)
    % Progress display
    fprintf('image %u\n',i)

    % String for filename
    CombinedFilename = sprintf('Unity images/CombinedRemapped/combinedremapped%05u.png',i); %combinedxxxxx.png, with padding

    % Loading the image
    img1 = imread(CombinedFilename);

    % Features are calculated
    EntropyUnity(i,1)=mean(mean(entropyfilt(img1)));
    EntropyUnity(i,2)=std2(entropyfilt(img1));
    for r=[1,2,3,4,5,6,7,8]
        SE_disk = getnhood(strel('disk',r));
        EntropyUnity(i,2+r) = mean(mean(entropyfilt(img1,SE_disk)));
        EntropyUnity(i,10+r) =std2(entropyfilt(img1,SE_disk));
    end

end

% Save Features
save('EntropyUnity.mat','EntropyUnity');
```

Figure-A III-2 Code qui permet de déterminer l'entropie,
utilisé ici sur les images de Unity

```
% Computes the wavelet features for the Unity dataset and uses these
%
% Author: Mehdi Temimi
% Date: 2020-08-08
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all

% The PSD are loaded from the Excel file. First column with filename is not
% read if present.
PSD_Unity=xlsread('Unity images/Granulométrie.xlsx');

% The script cycles through the filenames
Filenames = dir('Unity images/CombinedOriginal/*.png');
for i=1:size(Filenames,1)
    % String for filename
    CombinedFilename = sprintf('Unity images/CombinedRemapped/combinedremapped%05u.png',i); %combinedxxxxx.png, with padding

    % Loading the image
    img1 = imread(CombinedFilename);

    % Features are calculated
    [a2, h2, v2, d2]=haart2(img1,7); % Haar transform Eq.9, 2 levels.
    for j=1:7
        Eh(j) = h2(1,j).^2;
        wavelet_FD.SumEh(i,j) = sum(sum(Eh(1,j)));
        wavelet_FD.StdEh(i,j) = std(Eh(1,j)(:));
        Ev(j) = v2(1,j).^2;
        wavelet_FD.SumEv(i,j) = sum(sum(Ev(1,j)));
        wavelet_FD.StdEv(i,j) = std(Ev(1,j)(:));
        Ed(j) = d2(1,j).^2;
        wavelet_FD.SumEd(i,j) = sum(sum(Ed(1,j)));
        wavelet_FD.StdEd(i,j) = std(Ed(1,j)(:));
        wavelet_FD.SumEt(i,j) = sum(sum(Eh(1,j)))+sum(sum(Ev(1,j)))+sum(sum(Ed(1,j)));
    end
    WaveletUnity=[wavelet_FD.SumEt, wavelet_FD.StdEd, wavelet_FD.SumEd, wavelet_FD.StdEv, wavelet_FD.SumEv, wavelet_FD.StdEh, wavelet_FD.SumEh];


end

% Save Features
save('WaveletUnity.mat','WaveletUnity');
```

Figure-A III-3 Code qui permet de déterminer l'ondelette,
utilisé ici sur les images de Unity

```
Created on Fri Jun  5 09:33:01 2020
@author: Temimi
"""
import cv2
import numpy as np
import os
import glob
import mahotas as mt
from sklearn.svm import LinearSVC
import imageio
import numpy
from matplotlib.pyplot import *
from random import *

#La fonction qui détermine les Haralick features
def extract_features(image):
        textures = mt.features.haralick(image)

        ht_mean = textures.mean(axis=0)
        return ht_mean

#le dossier qui contient les images
folder_path = "C:/Users/AP85150/Desktop/IA/Code/Unity images/CombinedRemapped"

#la liste des éléments du dossier
path=os.listdir("C:/Users/AP85150/Desktop/IA/Code/Unity images/CombinedRemapped")

#génére aléatoirement 100 images du dossier
#path = sample(liste, 100)

matrix=[]

for i in path:

    #Trouver le fichier adéquat
    fichier=""
    fichier= folder_path + "/" + i
    #print(fichier)

    #Charger l'image
    image = cv2.imread(fichier)

    # détermine la gray_tone spatial dependence
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    #détermine les Haralick features
    features = extract_features(gray)
    #print(features)

    #fait une liste de toutes les features
    matrix= matrix+ list(features)


#transforme cette liste de features en matrice
matrix = numpy.reshape(matrix,(3003,13))
matrix
```

Figure-A III-4 Code qui permet de déterminer les textures d'Haralick, utilisé ici sur les images de Unity

ANNEXE IV

RÉSEAU DE NEURONES

```
clear all
close all
clear gpu

%Variables are created for the real PSD and Entropy features
PSD_Unity=xlsread('Unity images/Granulométrie.xlsx');
load('Features/EntropyUnity.mat');

% Network parameters
netEntropy = fitnet(10);

netEntropy.performFcn = 'mse';
netEntropy.trainParam.epochs=10000;
netEntropy.trainParam.min_grad=0.000001;
netEntropy.trainParam.show;
netEntropy.trainParam.max_fail=5000;
netEntropy.trainParam.goal=1e-7;
[netEntropy,trEntropy] = train(netEntropy,EntropyUnity',PSD_Unity','useGPU','no');

test_RealPSD = PSD_Unity(trEntropy.testInd,:);
test_PredictedPSD = netEntropy(EntropyUnity(trEntropy.testInd,:)')';

RMSE(1) = sqrt(perform(netEntropy,test_RealPSD(:,1),test_PredictedPSD(:,1)));
RMSE(2) = sqrt(perform(netEntropy,test_RealPSD(:,2),test_PredictedPSD(:,2)));
RMSE(3) = sqrt(perform(netEntropy,test_RealPSD(:,3),test_PredictedPSD(:,3)));
RMSE(4) = sqrt(perform(netEntropy,test_RealPSD(:,4),test_PredictedPSD(:,4)));
RMSE(5) = sqrt(perform(netEntropy,test_RealPSD(:,5),test_PredictedPSD(:,5)));
RMSE(6) = sqrt(perform(netEntropy,test_RealPSD(:),test_PredictedPSD(:)));

subplot(3,2,1)
plot(test_RealPSD(:,1),test_PredictedPSD(:,1),'+k')
hold on
title(sprintf('710 µm, RMSE = %.1f %%', RMSE(1)))
plot([-10 110],[-10 110],'-k')
xlim([-10 110])
ylim([-10 110])
```

```
subplot(3,2,2)
plot(test_RealPSD(:,2),test_PredictedPSD(:,2),'+k')
hold on
title(sprintf('425 µm, RMSE = %.1f %%', RMSE(2)))
plot([-10 110],[-10 110],'-k')
xlim([-10 110])
ylim([-10 110])

subplot(3,2,3)
plot(test_RealPSD(:,3),test_PredictedPSD(:,3),'+k')
hold on
title(sprintf('250 µm, RMSE = %.1f %%', RMSE(3)))
plot([-10 110],[-10 110],'-k')
xlim([-10 110])
ylim([-10 110])

subplot(3,2,4)
plot(test_RealPSD(:,4),test_PredictedPSD(:,4),'+k')
hold on
title(sprintf('150 µm, RMSE = %.1f %%', RMSE(4)))
plot([-10 110],[-10 110],'-k')
xlim([-10 110])
ylim([-10 110])

subplot(3,2,5)
plot(test_RealPSD(:,5),test_PredictedPSD(:,5),'+k')
hold on
title(sprintf('106 µm, RMSE = %.1f %%', RMSE(5)))
plot([-10 110],[-10 110],'-k')
xlim([-10 110])
ylim([-10 110])

subplot(3,2,6)
plot(test_RealPSD(:),test_PredictedPSD(:),'+k')
hold on
title(sprintf('Tous les tamis, RMSE = %.1f %%', RMSE(6)))
plot([-10 110],[-10 110],'-k')
xlim([-10 110])
ylim([-10 110])
```

Figure-A IV-1 Réseau de neurones utilisé avec les images de Unity avec l'entropie

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Adeli, H. (2001). Neural networks in civil engineering: 1989-2000. *Computer-Aided Civil and Infrastructure Engineering, 16*(2), 126-142. doi:10.1111/0885-9507.00219

Amankwah, A., & Aldrich, C. (2011). Estimation of Particulate Fines on Conveyor Belts by Useof Wavelets and Morphological Image Processing. *International Journal of Machine Learning and Computing, 1*(2), 132-137. doi:10.7763/IJMLC.2011.V1.20

Andersson, T. (2010). *Estimating particle size distributions based on machine vision.* (Thèse de doctorat, Luleå University of Technology, Luleå, Sweden). Repéré à https://www.diva-portal.org/smash/get/diva2:991638/FULLTEXT01.pdf

ASTM International. (2017a). *Standard Practice for Classification of Soils for Engineering Purposes (Unified Soil Classification System)*. Norme D2487-17e1. West Conshohocken, PA: ASTM International .

ASTM International. (2017b). *Standard Test Methods for Particle-Size Distribution (Gradation) of Soils Using Sieve Analysis*. Norme D6913 / D6913M-17. West Conshohocken, PA: ASTM International

ASTM International. (2020). *Standard Specification for Woven Wire Test Sieve Cloth and Test Sieves.* Norme E11-20. West Conshohocken, PA: ASTM International

Basseville, M. (1979). Détection de contours: méthodes et études comparatives. *Annales des Télécommunications, 34*(11), 559-579. doi:10.1007/BF03004242

Beucher, S. (1990). *Segmentation d'Images et Morphologie Mathématique.* (Thèse de doctorat, École des Mines de Paris, Paris, France). Repéré à https://www.researchgate.net/publication/29973646_Segmentation_d'Images_et_Morphologie_Mathematique

Blanco-Pons, S., Carrión-Ruiz, B., & Lerma, J. (2016). *Review of augmented reality and virtual reality techniques in rock art*. Communication présentée à the Arqueológica 2.0 - 8th International Congress on Archaeology, Computer Graphics, Cultural Heritage and Innovation, Valencia (Spain). doi 10.4995/arqueologica8.2016.3561

Bouchard, J. (2016). *Caractérisation de l'enrochement du barrage Romaine-2.* (Mémoire de maîtrise, Université Laval, Québec, Canada). Repéré à http://hdl.handle.net/20.500.11794/27324

Chapuis, R. (2012). Predicting the saturated hydraulic conductivity of soils: A review. *Bulletin of Engineering Geology and the Environment, 71*(3), 401-434. doi:10.1007/s10064-012-0418-7

Chapuis, R. (2017). *Extracting information from grain size distribution curves* (2e éd). Montréal: Geotics.

Cheimanoff, N. M., Chavez, R., & Schleifer, J. (1993). *FRAGSCAN: A scanning tool for fragmentation after blasting.* Article de conférence présenté à the 4th International symposium, Rock fragmentation by blasting, Vienna.

Cocquerez, J.-P., & Philipp, S. (1995). *Analyse d'Images: Filtrage et Segmentation* (2e éd). Paris : Masson.

Combarieu, O. (1996). A propos de la détermination de l'angle de frottement des sols pulvérulents au pressiomètre. *Revue Française de Géotechnique,* 6(77), 51-57. doi:10.1051/geotech/1996077051

Digabel, H., & Lantuéjoul, C. (1978). *Iterative algorithms*. article présenté à the Actes du Second Symposium Européen d'Analyse Quantitative des Microstructures en Sciences des Matériaux, Biologie et Médecine, (p.85-89) Repéré à https://www.bibsonomy.org/bibtex/137ab19e0430de25a4ec820b1467790e7

Durand, J.-M., Royet, P., & Meriaux. (1999). *Technique des petits barrages en Afrique sahélienne et équatoriale* (Cemagref ed.). Paris: Quae.

Esen, S., & Bølgøn, H. A. (2000). Effect of explosive on fragmentation. In *Department of Mining Engineering*. Ankara, Turkey: Middle East Technical University.

Facco, P., Santomaso, A. C., & Barolo, M. (2017). Artificial vision system for particle size characterization from bulk materials. *Chemical Engineering Science, 164*, 246-257. doi:https://doi.org/10.1016/j.ces.2017.01.053

Ferrari, S., Piuri, V., & Scotti, F. (2008). *Image Processing for Granulometry Analysis via Neural Networks*, IEEE International Conference on Computational Intelligence for Measurement Systems and Applications, Istanbul, Turkey, (pp. 28-32), doi: 10.1109/CIMSA.2008.4595827 .

Ghalib, A., Hryciw, R., & Shin, S. (1998). Image textural analysis and neural networks for characterization of uniform soils. *Proc.,Conf. on Computing in Civil Engineering*, 671-682.

Girdner, K. K., Kemeny, J. M., Srikant, A., & McGill, R. (1996). The split system for analyzing the size distribution of fragmented rock. In J. Franklin & T. Katsabanis (Eds.), *Measurement of Blast Fragmentation* (pp. 101–108). Rotterdam: Balkema.

Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing (4th Edition)*. New York, NY: Pearson

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., WardeFarley, D., Ozair, S., Courville, A., & Bengio,Y. (2014). *Generative adversarial nets*. Article présenté à Advances in neural information processing systems (p. 2672–2680), Montréal, Canada.

Haar, A. (1910). Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen, 69*(3), 331-371. doi:10.1007/BF01456326

Hamzeloo, E., Massinaei, M., & Mehrshad, N. (2014). Estimation of particle size distribution on an industrial conveyor belt using image analysis and neural networks. *Powder Technology, 261*, 185–190. doi:10.1016/j.powtec.2014.04.038

Haralick, R., Shanmugam, K., & Dinstein, I. (1973). Textural Features for Image Classification. *IEEE Trans Syst Man Cybern, SMC-3*, 610-621.

Huang, H., Yu, P.S., & Wang, C. (2018). An Introduction to Image Synthesis with Generative Adversarial Nets. Repéré à *ArXiv, abs/1803.04469*.

Holmberg, R. (2003). *Explosives and Blasting Techniques,* Communication présentée à Second World Conference on Explosives and Blasting Technique (EFEE). Prague, Czech Republic.

Holtz, R. D., & Kovacs, W. D. (1981). *An Introduction to Geotechnical Engineering*. Prentice_Hall,Englewood Cliffs, N.J.

Hryciw, R., & Jung, Y. (2009). Three-point imaging test for AASHTO soil classification. *Transportation Research Record, 2101*, 27-33. doi:10.3141/2101-04

Hryciw, R., Ohm, H., & Zhou, J. (2013). Theoretical Basis for Optical Granulometry by Wavelet Transformation. *Journal of Computing in Civil Engineering, 29*, 04014050. doi:10.1061/(ASCE)CP.1943-5487.0000345

Kallergis, D., Chimos, K., Chavakis, T., Papacharalampou, C., & Douligeris, C. (2014). *Earth and groundwater visualisation using 3D modelling techniques in a cloud computing environment.* Article présenté à the IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), Noida, India.

Katsabanis, T. (1999). *Calibration of optical digital fragmentation mesuring systems*. Article présenté à 25th Annual Conference on Explosing and Blasting Techniques, II(ISEE), Cleveland OH, (p. 325-332).

Khalifa, N. H., Nguyen, Q. V., Simoff, S. J., & Catchpoole, D. R. (2015). *A visualization system for analyzing biomedical and genomic data sets using Unity3D platform.* Article présenté à the 8th Australasian Workshop on Health Informatics and Knowledge Management, Sydney, Australia.

Larkin, K. (2016). Reflections on Shannon Information: In search of a natural information-entropy for images. *Image science and optics journal*, 47. Repéré à https://arxiv.org/abs/1609.01117

Lefèvre, S. (2009). Segmentation par Ligne de Partage des Eaux avec Marqueurs Spatiaux et Spectraux. *Colloque GRETSI sur le Traitement du Signal et des Images*, 4. Repéré à https://hal.archives-ouvertes.fr/hal-00516095/document

Leps, T.M. "Flow through rockfill" In Hirschfeld et Poulos (1973). Embankment dam engineering : Casagrande volume. New York : Wiley.

Liu L., Chen J., Fieguth P., Zhao G., Chellappa R., & Pietikäinen M., (2019). From BoW to CNN: Two Decades of Texture. Representation for Texture Classification, *International journal of computer vision*. 127 , 74–109. doi:10.1007/s11263-018-1125-z.

Liu, Q., & Tran, H. (1996). *Comparing systems - Validation of FragScan, WipFrag and Split*. article présenté à Proceedings of the FRAGBLAST 5 Workshop on Measurement of Blast Fragmentation, Montreal, Quebec, Canada, (p 151-156).

Lu, P., Chen, S., & Zheng, Y. (2012). Artificial Intelligence in Civil Engineering. *Mathematical Problems in Engineering, 2012*, 22. doi:10.1155/2012/145974

Maerz, N. (1998). *Aggregate sizing and shape determination using digital image processing.* Article présenté à the Sixth Annual Symposium Proceedings, St. Louis, Missouri.

Maerz, N., Palangio, T., & Franklin, J. (1996). *WipFrag Image Based Granulometry System*. Article présenté à Proceedings of the FRAGBLAST 5 Workshop on Measurement of Blast Fragmentation, Montreal, Quebec, Canada, (p 91-99).

Maerz, N. H., Franklin, J. A., Rothenburg, L., & Linncoursen, D. (1987). *Measurement Of Rock Fragmentation By Digital Photoanalysis*. Article présenté à the 6th ISRM Congress, Montreal, Canada.

Manashti, J., Duhaime, F., Toews, M., & Pirnia, P. (2020). (soumis) Grain Size Analysis of Granular Materials Based on Image Feature Extraction. *Journal of Computing in Civil Engineering.*

Michel, F., & Courard, L. (2006). *Apport de la granulométrie laser dans la caractérisation physique des fillers calcaires*. Article présenté à the Septième édition des Journées scientifiques du Regroupement francophone pour la recherche et la formation sur le béton, Toulouse, France.

Ondercin, M. (2016). *An Exploration of Rockfall Modelling through Game Engines.* (Mémoire de maîtrise, Queen's University, Kingston, Canada).

Ool, A., & Soria, J. (1992). The application of wavelet analysis to extract particle size information from particle-laden flow images. Communication présentée à *11th Australasian Fluid Mechanics Conf*(Australia), (p 187-190).

Orr, T. J., MacDonald, B. D., Iverson, S. R., & Hammond, W. R. (2016). *Development of a Generic Mine Visualization Tool Using Unity.* Article présenté à the International Symposium on the Application of Computers and Operations Research in the Mineral Industry, Fairbanks, Alaska.

Outal, S. (2006). *Quantification par analyse d'images de la granulométrie des roches fragmentées : amélioration de l'extraction morphologique des surfaces, amélioration de la reconstruction stéréologique*. (Thèse de doctorat en Géologie appliquée. Ecole Nationale Supérieure des Mines de Paris, 2006).

Pirnia, P., Duhaime, F., & Manashti, J. (2018). *Machine learning algorithms for applications in geotechnical engineering*. Article présenté à the GeoEdmonton 2018, Edmonton.

Sala, Z. (2018). *Game-Engine Based Rockfall Modelling: Testing and Application of a New Rockfall Simulation Tool.* (Mémoire de maîtrise, Queen's University, Kingston, Canada).

Schleifer, J., & Tessier, B. (1996). *FRAGSCAN: A tool to measure fragmentation of blasted rock.* Article Tirée de the Measurement of Blast Fragmentation, Franklin and Katsabanis (eds), (p 73–78), Balkema, Rotterdam.

Scott, R.-F. (1989). Esaai en centrifugeuse et technique de la modélisation. *Revue Française de Géotechique, 48*, 15-34. Repéré à http://www.geotech-fr.org/sites/default/files/rfg/article/48-2.pdf

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal, 27*(3), 379-423. doi:10.1002/j.1538-7305.1948.tb01338.x

Shin, S., & Hryciw, R. (2004). Wavelet Analysis of Soil Mass Images for Particle Size Determination. *Journal of Computing in Civil Engineering, 18*(1). doi:10.1061/(ASCE)0887-3801(2004)18:1(19)

Singh, P. K., Roy, M. P., Paswan, R. K., Sarim, M., Kumar, S., & Ranjan Jha, R. (2016). Rock fragmentation control in opencast blasting. *Journal of Rock Mechanics and Geotechnical Engineering, 8*(2), 225-237. doi:https://doi.org/10.1016/j.jrmge.2015.10.005

Smid, L. (2017). *Master's Thesis: 3D visualization of geological study areas from high-detail geospatial datasets using Unity*. [Vidéo en ligne] Repéré à https://vimeo.com/234914406

Sobel, I., & Feldman, G. (1973). A 3×3 isotropic gradient operator for image processing. *Pattern Classification and Scene Analysis*, 271-272.

Terzaghi, K. (1960). Discussion of Rockfill dams : Salt Springs and Lower Bear River concrete face dams by I.C. Steele and J. Barry Cooke. *Transactions of the American Society of Civil Engineers. Vol.* 125, no 2, p. 139-148

Thurley, M. J. (2011). Automated online measurement of limestone particle size distributions using 3D range data. *Journal of Process Control, 21*(2), 254-262. doi:10.1016/j.jprocont.2010.11.011

Unity Technologies. (2019). Unity (Version 2019.3.13). [Logiciel] Repéré à https://unity.com/fr

Wipware Inc. (2013). WipFrag. (Version 3). [Logiciel] Repéré à https://wipware.com/

Yaghoobi, H., Mansouri, H., Ebrahimi Farsangi, M. A., & Nezamabadi-Pour, H. (2019). Determining the fragmented rock size distribution using textural feature extraction of images. *Powder Technology, 342*, 630-641. doi:https://doi.org/10.1016/j.powtec.2018.10.006