

Xingyuan XUE



ENSAE 1^{ère}année
Stage d'ouverture au monde professionnel
Année scolaire 2015-2016

Découpage intelligent dans Blender

« confidentiel »

Qarnot Computing
Montrouge, France

Maître de stage : Yanik NGOKO
01/06/2016 – 31/08/2016

Sommaire

Introduction	1
1 Présentation générale de l'organisme d'accueil	2
1.1 Histoire de Qarnot Computing	2
1.2 Objectifs, structure et moyens financiers et humains de Qarnot Computing	2
1.2.1 Objectifs de Qarnot Computing	2
1.2.2 Structure de Qarnot Computing	3
1.2.3 Moyens financiers et humains de Qarnot Computing	3
2 Environnement de travail, organisation de travail et relations professionnelles	4
2.1 Environnement de travail	4
2.1.1 Présentation générale de l'environnement de travail	4
2.1.2 Environnement physique de travail	4
2.1.3 Environnement culturel de travail	5
2.2 Organisation de travail	6
2.3 Relations professionnelles	6
2.3.1 Relations professionnelles dans une même équipe	6
2.3.2 Relations professionnelles dans les différentes équipes	7
2.3.3 Relations professionnelle avec les clients	7
3 Description de la mission et analyse des résultats obtenus	8
3.1 Description de la mission	8
3.1.1 Objectif de la mission	8
3.1.2 Résolution préconisée	9
3.1.3 Différentes étapes conçues pour atteindre l'objectif	9
3.2 Analyse des résultats	10
3.2.1 Description des résultats	10
3.2.2 Observations et impressions	14
Bilan de stage	16
Conclusion	17
A. Résultat de l'analyse de la statistique descriptive	20
B. Code de l'algorithme Rectangle Stacking	23

C. Related Works	40
-------------------------	-----------

Introduction

Cet été, j'ai effectué un stage de trois mois chez Qarnot Computing, une startup qui fabrique les radiateurs intelligents et offre le service de Cloud Computing grâce aux processeurs de haute performance intégrés dans les radiateurs. Dans ce rapport, je raconterai cette expérience en présentant les différents aspects de Qarnot Computing et le déroulement de ma mission.

Le corps de ce rapport est divisé en trois chapitres. Le premier chapitre porte sur la présentation générale de Qarnot Computing, son histoire, ses objectifs, sa structure et ses moyens financiers et humains. Qarnot Computing a vécu un développement rapide avec une idée derrière innovante. Elle offre deux sortes de services avec une structure affinée et flexible et des équipes majoritairement jeunes. Dans le deuxième chapitre, on présentera cette entreprise de façon plus profonde. On décrira son environnement physique et culturel, l'organisation de travail et les relations professionnelles des membres dans une même équipe, dans les différentes équipes ou avec les clients. Généralement, Qarnot Computing a un environnement idéal pour un travail en informatique et pour la communication et la collaboration des membres. De plus, on constate une hiérarchie et en même temps une grande liberté dans l'organisation de travail chez Qarnot Computing. En termes de relations professionnelles, j'entendais bien avec les autres et l'entreprise a une bonne relation avec ses clients. Dans le troisième chapitre, je présenterai la description de ma mission qui s'agit de l'amélioration du service du rendu offert par Qarnot Computing ainsi que les résultats obtenus. Je décrirai aussi mes observations et impressions dans ce chapitre. La plupart de ces éléments sont écrits selon les expériences vécues. À la fin, je présenterai mon bilan de stage et la conclusion.

Je profite de ces lignes pour remercier mon maître de stage, Yanik. Il m'a beaucoup soutenue pendant toute la période de mon stage.

— Chapitre 1 —

Présentation générale de l'organisme d'accueil

1.1 Histoire de Qarnot Computing

Fondée par Paul Benoît, Miroslav Sviezeny et Grégoire Sirou en 2010, Qarnot Computing est une PME d'une vingtaine personnels maintenant. Qarnot Computing est née de l'idée d'utiliser la chaleur émise par les processeurs dans les data centres. Cette idée vient de Paul Benoît, le président-directeur général de Qarnot Computing, un X-Télécom et qui a 7 ans d'expérience en informatique bancaire, surtout en calcul intensif (High Performance Computing). Cette conception a été concrétisée par le Q.rad, un radiateur électronique qui utilise la chaleur émise par les processeurs hautement performants et dont la vente est l'une des ressources principales du revenu de Qarnot Computing. En 2011, Qarnot Computing a développé les premiers prototypes de Q.rad. L'année suivante, la première série de Q.rads est née et Qarnot Computing a cherché et a exploité les premiers sites de chauffage ainsi que leur capacité de calcul. Entre 2013 et 2014, Qarnot Computing a déployé 300 Q.rads dans un bâtiment de logements sociaux à Paris. En 2015, Qarnot Computing a obtenu des clients du secteur bancaire, de l'animation 3D et de l'industrie. De plus, elle a établi des collaborations avec des centres de recherche français en informatique et en biotechnologie. Grâce à sa contribution à la réduction de l'impact négatif sur l'environnement du calcul intensif, à la vie écologique et intelligente et à la excellente conception de Q.rad, Qarnot Computing a gagné une dizaine de prix en France et à l'étranger. En 2016, Qarnot a établi un partenariat avec un acteur majeur des centres de données en Europe et elle a déployé ses bureaux à l'étranger, par exemple, au Danemark. [1]

1.2 Objectifs, structure et moyens financiers et humains de Qarnot Computing

1.2.1 Objectifs de Qarnot Computing

Qarnot Computing est bâtie autour de deux objectifs : la création d'une plateforme distribuée de chauffage et la mise en place d'un nouveau type de cloud décentralisé. Ces deux objectifs se déclinent par la mise en place de deux produits : le Q.rad et le Q.ware. Les Q.rads sont déployés dans des habitatiions (logements, bureaux etc.) et chauffent en effectuant des calculs. Le Q.ware est un intergiciel cloud qui sert à distribuer et moduler les calculs sur les Q.rads.

D'un coté, les équipes de Qarnot Computing s'efforcent à améliorer les Q.rads pour son API (Application Programming Interface), ses usages intelligents etc. De l'autre côté, l'entreprise développe et maintient les plateformes de Cloud Computing et fournit les calculs de haute performance en utilisant l'énergie efficacement.

1.2.2 Structure de Qarnot Computing

La structure de Qarnot Computing est affinée et flexible. Paul Benoît est le président-directeur général de Qarnot Computing. Il s'occupe de toutes les affaires de l'entreprise et il est au courant de tous les projets effectués chez Qarnot Computing, incluant le projet effectué dans mon stage. Il y a en général deux grandes équipes chez Qarnot Computing, l'équipe d'ingénieurs et l'équipe de commerce. L'équipe d'ingénieurs contient environ une dizaine de personnels et cette équipe se divise en plusieurs petites équipes incluant l'équipe de développement du système de Cloud computing (Q.ware), l'équipe de machine learning, l'équipe de l'amélioration de Q.rad etc. Chaque équipe est petite et comprend au plus 4 personnes. Mon stage s'est effectué dans l'équipe de Machine Learning. La structure est aussi flexible car une personne peut appartenir à plusieurs équipes. Par exemple, il y a une personne qui appartient à la fois à l'équipe de machine learning et à l'équipe de développement du système de Cloud Computing.

1.2.3 Moyens financiers et humains de Qarnot Computing

Qarnot Computing a deux principales sources de revenus. L'une est la vente de Q.rads et l'autre est la vente du service de Cloud computing aux banques, aux studios d'animation 3D, aux centres de recherche etc.

Le personnel à Qarnot Computing est majoritairement jeune. La plupart des personnes sont de jeunes diplômés des grandes écoles dont certains ont fait les études en informatique et dont les autres ont appris les divers aspects de commerce. Le reste du personnel est assez expérimenté. Certaines d'entre ces personnes ont l'expérience de fonder une startup ou de travailler dans un grand groupe.

— Chapitre 2 —

Environnement de travail, organisation de travail et relations professionnelles

2.1 Environnement de travail

2.1.1 Présentation générale de l'environnement de travail

À mon arrivée, Qarnot Computing occupait le quatrième étage entier et à la fin d'août, elle s'est agrandie en deux étages entiers. Il y a cinq bureaux, deux salles de réunion, une salle à manger qui sont alignés aux deux côtés d'un couloir au quatrième étage. Un balcon se trouve dehors. Il n'y a pas de bureaux individuels chez Qarnot Computing. On travaille toujours avec les autres.

2.1.2 Environnement physique de travail

D'abord, comme Qarnot Computing est une startup de l'informatique, l'environnement physique chez Qarnot Computing est du type de l'environnement pour les geeks à mon avis. À mon arrivée chez cette entreprise, j'étais surprise par les nombreux grands écrans, chaque bureau avec deux ou trois écrans. Ces écrans étaient rangés en parallèle et sont connectés entre eux. Ils nous permettaient de travailler plus facilement. Chez Qarnot Computing, on utilise le système Linux, qui est beaucoup plus rapide que le système Windows.

En second lieu, l'environnement physique chez Qarnot Computing est favorable pour la communication entre les collègues. En général une même équipe travaille dans le même bureau, ce qui facilite la communication entre les membres de l'équipe. J'ai travaillé avec l'équipe de développement du système de Cloud Computing et comme mon projet de stage concernait beaucoup sur le système de Cloud Computing, cela me permettait de poser la question directement à un membre de l'équipe de Cloud Computing ou montrer le problème avec l'écran directement. De plus, quand je suis arrivée, tous mes collègues travaillaient dans le même étage et par ailleurs les bureaux sont alignés aux côtés d'un couloir. Donc, les membres de différentes équipes posaient les questions en quelques pas. En outre, un collègue a créé un groupe de communication en ligne sur Slack, une application pour la communication entre les membres d'une équipe, ce qui a facilité la communication de toute l'entreprise. De temps en temps, il y avait des conversations des



Graphique 2.1 – Environnement physique de travail

problèmes techniques sur le site. Finalement, Qarnot Computing a un système de mail basé sur Google Mail qui permet la transmission des documents entre les collègues.

Enfin, l'environnement physique de Qarnot Computing est aussi favorable pour discuter des problèmes. J'ai souvent fait le point ou discuté des problèmes avec mon maître de stage ou avec des autres collègues. Il y a des lieux pour le faire, par exemple, les salles de réunion, la salle à manger ou le balcon où il y a des chaises et des tables. Également, il y a un tableau blanc sur le mur pour qu'on puisse montrer un problème et le discuter avec plusieurs collègues.

2.1.3 Environnement culturel de travail

Qarnot Computing a un environnement culturel cohésif, dynamique et plein de passion.

En termes de cohésion, l'entreprise organise une réunion qui s'appelle Q.note chaque semaine. Une des équipes présente ce qu'elle est en train de faire et les autres peuvent poser les questions sur les points qu'ils ne comprennent pas pendant la réunion. Dans cette réunion, j'ai pris connaissance de ce que mes collègues étaient en train de concevoir et j'ai mieux compris comment l'entreprise fonctionnait. De plus, les présentateurs exposaient leur sujet de façon simple pour que les gens qui n'avaient connu rien sur ce sujet avant la présentation puissent comprendre. Donc, ces réunions permettaient à tout le monde dans l'entreprise de comprendre ce que les autres faisaient et d'être au courant du développement de l'entreprise. À mon avis, cela permet à chaque membre de l'entreprise voire les stagiaires d'avoir un sens de participation pour le développement de l'entreprise et de faire se connaître les membres de sorte que la compagnie soit mieux unie. En outre, le personnel de l'entreprise déjeune ensemble dans la salle à manger. On partageait des blagues, des actualités et des jeux. Comme dans les autres entreprises, Qarnot Computing organise un afterwork (appelé Qfterwork dans l'entreprise) presque toutes les deux semaines pour que les personnels se bavardent et se détendent. Qarnot Computing organise aussi d'autres activités. On a fait des barbecues dans le balcon et on a joué aux jeux de cartes après le travail.

En second lieu, l'entreprise est très dynamique. Selon mes observations, chaque membre de l'entreprise avait toujours du travail à effectuer. J'étais aussi toujours occupée par quelque tâche.

De temps en temps, même Paul est venu pour me demander si j'avais le travail à faire et je crois qu'il voulait assurer que tout le monde avait le travail à faire.

Enfin, l'environnement culturel de Qarnot Computing est plein de passion. Il était normal de constater que mes collègues travaillaient après 7 heures le soir. Quand on faisait la pause le midi, cela normalement ne dépassait pas une heure et tout le monde revenait au travail volontairement. Pour la plupart de temps, le temps de travail par jour des employés chez Qarnot Computing a bien dépassé 7 heures et tout le monde travaillaient volontairement.

2.2 Organisation de travail

Généralement, le président-directeur général, Paul Benoît s'occupe de toutes les choses et décide ce que chaque équipe à faire. Pour chaque équipe, il y avait un chef qui surveille le travail des membres de cette équipe et attaque à la tâche concrètement que Paul distribue. Par exemple, je faisais une équipe avec mon maître de stage sur un sujet qui a été défini par Paul. Mon maître de stage a conçu les démarches pour résoudre le problème et j'ai réalisé les conceptions concrètement sur l'ordinateur. On a mentionné Q.note, la présentation de travail. C'était non seulement une occasion de faire connaître le travail aux autres, mais également une occasion de présenter le travail à Paul.

Par ailleurs, chez Qarnot Computing, on a une grande liberté pour son travail. Paul ne définit que le sujet de travail pour chaque équipe. La résolution est à choisir et à décider par chaque membre. Dans mon stage, j'ai conçu un algorithme pour résoudre mon problème. Mon maître de stage ne m'a qu'expliqué le problème et c'était à moi de choisir comment résoudre le problème.

En termes de règles, il n'était pas obligatoire d'arriver au bureau avant un certain moment. Mais tout le monde assurait qu'il avait passé au moins 7 heures dans le bureau. Il y avait un collègue qui partait toujours à 17 heures. Mais il arrivait à 8 heures le matin. Par contre, deux collègues arrivaient au bureau à environ 10 heures et ils partaient à environ 19 heures. Par contre, il y avait la date limite dans le travail. Par exemple, parfois, mon maître de stage m'a demandé de finir une tâche cette semaine et je n'ai jamais dépassé la date limite. On ne discutait pas normalement dans le bureau comme cela perturberait les autres. Quand on discutait dans le bureau, on baissait la voix afin de ne pas perturber les autres.

2.3 Relations professionnelles

2.3.1 Relations professionnelles dans une même équipe

Selon mes observations, les relations professionnelles dans une même équipe étaient harmonieuses. À cause de la limite de place, j'ai travaillé dans le même bureau avec trois différentes équipes. Il y avait toujours les blagues dans les conversations entre les membres et le partage du goûter.

Pour moi, j'ai fait une équipe avec mon maître de stage, Yanik Ngoko. Selon le sujet de mon stage défini par Paul, Yanik concevait les démarches et il me distribuait les tâches. Je lui montrais les résultats quand j'ai fini mon travail et il m'indiquait comment avancer le projet après. Dans mon stage, quand les démarches conçues par lui ne marchaient pas en pratique, on en discutait

ensemble. Quand j'ai rencontré les problèmes en informatique et je lui ai demandé un coup de main, il venait toujours m'aider.

Dans mon stage, d'une part, Yanik m'a beaucoup aidée et m'a donné une grande liberté pour finir le sujet. De l'autre part, je ne sentais pas la hiérarchie de maître de stage et stagiaire avec lui. En premier lieu, quand j'ai eu des questions sur mon travail, il m'a répondu en détail et patiemment. De plus, il m'a parlé des connaissances qu'il avait sur cette question sans réserve. Par exemple, quand je rédigeais la partie de "related works" pour l'article sur le projet de mon stage, il m'a donné des conseils pour rédiger la partie de "related works" pour une thèse. En outre, il posait peu de contraintes à la réalisation de mon travail tant que j'ai obtenu des résultats. Par exemple, quand il m'a demandé de dessiner un graphique de 3 dimensions, il ne limitait pas le langage ou les packages que j'utilisais. En second lieu, quand je n'étais pas d'accord avec les résolutions qu'il a proposées, je pouvais discuter avec lui et donnais mes avis et parfois, il acceptait ma proposition. Quand sa conception ne marchait pas en pratique et je le lui montrais, il corrigeait sa conception et modifiait les étapes suivantes. Je parlais avec lui comme avec mes camarades de classe et je sentais l'égalité dans notre relation professionnelle.

2.3.2 Relations professionnelles dans les différentes équipes

Je m'entendais bien avec les membres des autres équipes. Je mangeais avec tout le monde dans l'entreprise chaque midi. Comme l'entreprise n'a pas de cantine, on marchait parfois ensemble à l'enseigne Carrefour Market dans la proche.

Comme mon projet aussi concernait le travail des autres équipes et comme je n'ai pas eu beaucoup de connaissances en informatique, je posais souvent les questions aux personnels dans les autres équipes. Gwenn, Clément, Yoann et Genoud m'ont aidée avec leurs connaissances du système de Cloud Computing et en informatique patiemment.

De plus, le président-directeur général, Paul, est venu souvent pour me demander si j'étais contente avec le stage ou comment s'avancait mon projet. Je mangeais souvent dans la même table avec les filles dans l'entreprise, Marion, Tiphaine, Élöise et Hélène. Toutes ces filles étaient gentilles et on partageait les blagues et nos vies pendant les déjeuners.

2.3.3 Relations professionnelle avec les clients

Qarnot Computing a des bonnes relations avec les clients. Il a un partenaire chinois qui collabore avec Qarnot Computing depuis une dizaine années et leur responsable est devenu ancien ami avec les responsables chez Qarnot Computing.

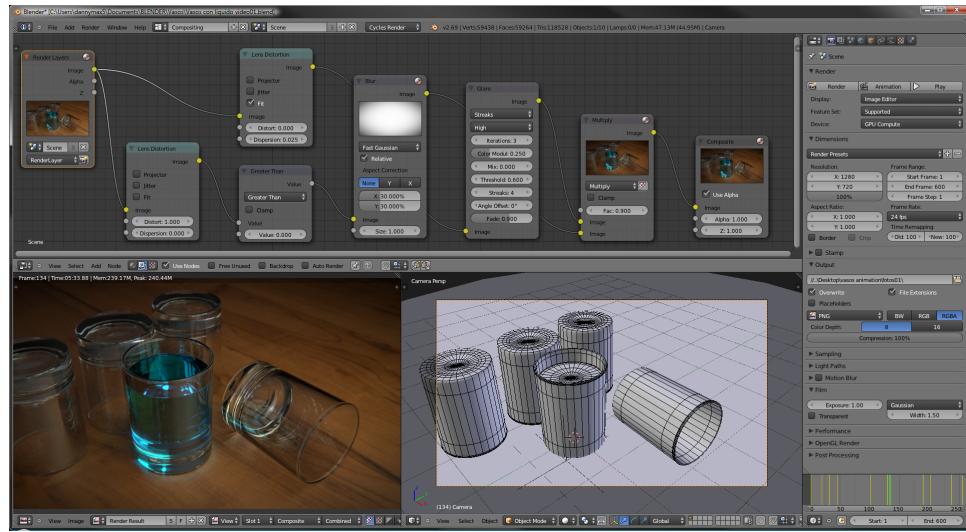
— Chapitre 3 —

Description de la mission et analyse des résultats obtenus

3.1 Description de la mission

3.1.1 Objectif de la mission

Comme je l'ai déjà indiqué, Qarnot Computing propose le service de Cloud Computing, c'est-à-dire que «l'exploitation de la puissance de calcul ou de stockage de serveurs informatiques distants par l'intermédiaire d'un réseau, généralement internet» [2]. Chez Qarnot Computing, les «serveurs informatiques distants» sont processeurs à hautes performances intégrés dans les Q.rads et le Cloud Computing s'applique au rendu dans l'animation. Le rendu est le processus qui génère un graphique d'un modèle de 2 dimensions ou de 3 dimensions [3]. Ce processus s'appuie sur la puissance de calcul des processeurs et le Cloud Computing permet d'obtenir de bons rapports performance-prix. Qarnot Computing utilise Blender, un open-source logiciel qui fait le rendu. L'interface de Blender est présentée ci-dessous. Le graphique à gauche est obtenu après le rendu du modèle à droite. C'est une démarche importante dans l'animation 3D. Bien que l'utilisation d'un cloud permet de raccourcir le temps de rendu d'un modèle, il reste des cas où il faut compter plus de dix heures pour la génération de certaines images. Dans l'animation, normalement, il faut 24 images pour générer une seconde d'animation. Donc, Qarnot Computing veut raccourcir encore le temps de rendu dans son système. Le processus de rendu à Qarnot computing est effectué en parallèle. Premièrement, la région à rendre est découpée d'abord de façon régulière, par exemple, 4 par 4, 8 par 8 etc. Ensuite, les sous-régions sont distribuées aux processeurs intégrés dans les Q.rads. Puis, chaque processeur effectue le rendu pour la sous-région distribuée. Enfin, les processeurs retournent les images engendrées correspondant aux sous-régions distribuées. Le temps de rendu observé par l'utilisateur est déterminé par le processeur dont le temps de rendu est le plus long. Normalement, certaines sous-régions sont plus difficiles à rendre que les autres. Il est fréquent de tomber dans des situations où la plupart des processeurs sont disponibles et quelques processeurs seulement travaillent à finir des rendus. Donc, le sujet de mon stage est de trouver une façon de découpage intelligent pour que les temps de rendu des processeurs soient plus équitables et finalement pour que le temps total de rendu soit raccourci.



Graphique 3.1 – L'interface de Blender

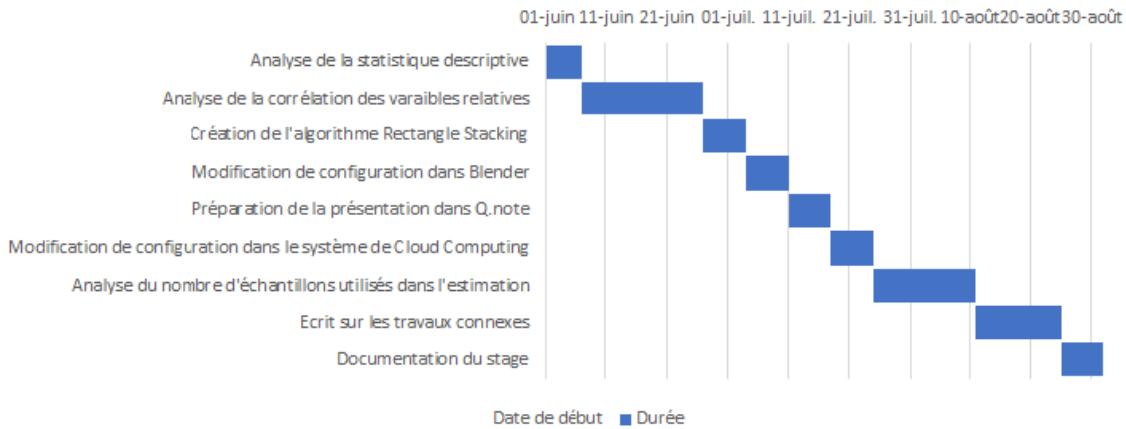
3.1.2 Résolution préconisée

Paul et un autre collègue ont préconisé une approche de résolution. L'idée est de découper la région à rendre de façon très fine, puis d'estimer le temps de rendu en utilisant un beaucoup plus petit nombre d'échantillons que le nombre d'échantillons souhaité ou en mettant la taille de l'image beaucoup plus petite que la taille souhaitée. Ensuite, on regroupe les sous-régions en fonction de l'estimation de leurs temps de rendu et chaque partie dans le regroupement correspond à un processeur qui rend cette partie pour que les temps de rendu des processeurs soient plus équitables. Les sous-régions sont regroupées sous forme de rectangles car dans le système de Qarnot Computing, on ne peut que rendre les régions sous forme de rectangles. On veut que le temps total de l'estimation plus le temps total de rendu après le découpage intelligent selon l'estimation soit moins important que le temps total de rendu en utilisant le découpage par défaut. Cette approche comprend deux phases : (1) celle d'estimer précisément le temps de rendu de chaque sous-région ; (2) celle de calculer la meilleure répartition des sous-régions entre des processeurs. Le problème est à considérer au sein d'un dilemme exploration-exploitation : l'exploration permet de collecter des informations pour estimer les temps de rendus de sorte à optimiser le calcul de la meilleure répartition (exploitation). On veut balancer le temps de l'exploration et le temps de l'exploitation pour que le temps total soit minimisé car plus le temps de l'exploration est important, plus le temps de l'exploitation est court et inversement.

3.1.3 Différentes étapes conçues pour atteindre l'objectif

Mon stage est constitué de six étapes. Dans la première étape, j'ai effectué l'analyse de la statistique descriptive sur les données des tâches de rendu lancées dans le système de Cloud Computing. Après, j'ai analysé la corrélation entre le temps de rendu et le nombre d'échantillons ou la taille de l'image. Ensuite, j'ai conçu un algorithme que j'appelais Rectangle Stacking pour trouver un découpage intelligent après lequel les temps de rendu des parties résultantes sont plus équitables. Puis, j'ai modifié les configurations dans Blender et dans le système de Q.rads pour que le chargement des fichiers ne soit effectué qu'une fois dans l'estimation et pour que le nouveau découpage soit réalisable dans le système de Cloud Computing car le système existant ne peut

qu'effectuer le découpage régulier et uniforme. Dans la cinquième étape, j'ai analysé la corrélation entre le nombre d'échantillons utilisé dans l'estimation et le temps gagné après un découpage intelligent par rapport au découpage par défaut. Enfin, j'ai entamé l'écrit d'un article sur mon projet et j'ai fini la partie des travaux connexes.



Graphique 3.2 – Diagramme de Gantt des étapes conçues

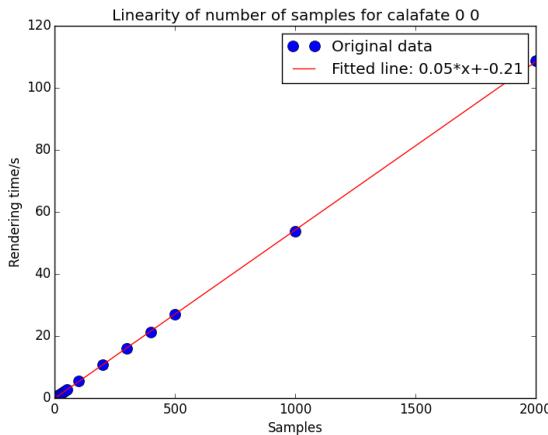
3.2 Analyse des résultats

3.2.1 Description des résultats

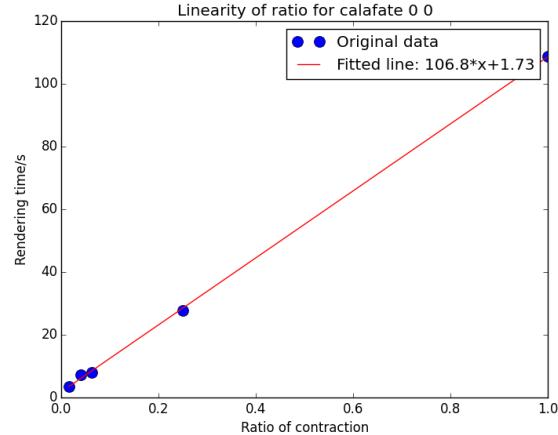
Pour ce stage, j'ai obtenu quatre résultats principaux. En premier lieu, j'ai prouvé la linéarité entre le temps de rendu et le nombre d'échantillons ou la taille de l'image engendrée. En second lieu, j'ai créé un algorithme qui pouvait découper une région en plusieurs parties sous forme de rectangles en regroupant les petites sous-régions en fonction de leurs temps de rendu pour que les temps de rendu des parties sont à peu près équitables. De plus, à partir d'un petit nombre d'échantillons, on peut effectuer une estimation assez précise. Finalement, en recherchant les travaux connexes, j'ai comparé mon projet avec ces travaux et j'ai trouvé que ce projet était relativement innovant en faisant une estimation des temps de rendu sur Blender et en utilisant un nouveau algorithme de «rectilinear tiling en 2 dimensions».

Premier résultat

En analysant les données des temps de rendu en utilisant différents nombres d'échantillons ou en mettant différentes tailles pour l'image finale, j'ai obtenu les deux graphiques ci-dessous en utilisant le modèle de la régression linéaire en Python. Dans le graphique à gauche, l'abscisse représente le nombre d'échantillons alors que l'ordonnée représente le temps de rendu de ce fichier. On peut constater que les points se trouvent autour d'une droite et le R-squared de ce modèle linéaire est plus que 99% qui montre que le modèle correspond bien aux données. Les autres 47 fichiers montrent le même résultat, ce qui vérifie la linéarité entre le temps de rendu et le nombre d'échantillons. La même conclusion a été tirée avec la taille de l'image finale. Dans le graphique à droite, l'abscisse représente le taux de contraction et la taille réelle égale à 1280×960 pixels \times taux de contraction. De même, l'ordonnée représente le temps de rendu. Le graphique



Graphique 3.3 – Linéarité entre le nombre d'échantillon et le temps de rendu



Graphique 3.4 – Linéarité entre la taille de l'image finale et le temps de rendu

montre que plus la taille est grande, plus le temps de rendu est long. En outre, on constate que pour toutes les droites obtenues par la régression linéaire, les termes de constant pour les droites sont toujours proches du point d'origine, c'est-à-dire que le temps de rendu est proportionnel au nombre d'échantillons et à la taille de l'image finale.

Par ailleurs, en termes de la représentativité des fichiers choisis dans les expériences, j'ai choisi 24 fichiers de différents types, par exemple, figure, paysage, animal, objet etc. Les niveaux de difficulté de rendre ces fichiers sont aussi très diversifiés.

En effet, chez Qarnot Computing, on utilise le moteur Cycles pour le rendu qui utilise la méthode «Ray Tracing». Cette méthode procède par lancer de rayons qui rebondissent partout depuis le caméra jusqu'à ce qu'on trouve la source de lumière. Un rayon est un échantillon. Plus la région est compliquée, plus le temps pour un rayon ou un échantillon de trouver la source de lumière est long [4]. Également, plus la taille de l'image finale est petite, plus le nombre total de rayons est petit. C'est parce que le nombre d'échantillons par pixel est fixé et la somme de pixels est petit quand la taille est petite.

Pour obtenir ce résultat, j'ai passé trois semaines pour la collecte des données précises. Au début, je n'ai pas soustrait le temps du chargement pour les données et donc les données collectées étaient erronées. Un collègue m'a rappelée de cette erreur et j'ai finalement obtenu ce résultat.

Deuxième résultat

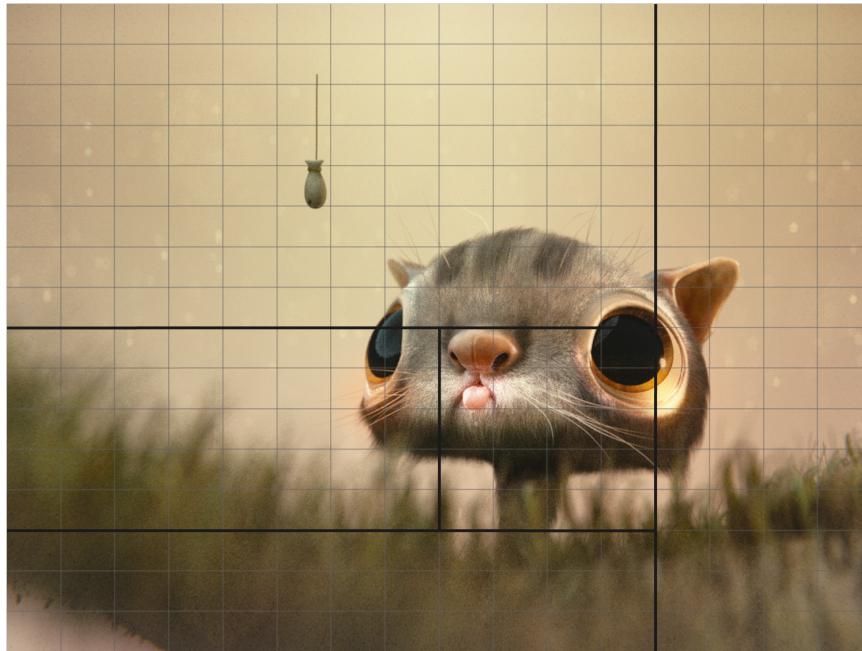
J'ai créé un algorithme et je l'ai appelé Rectangle Stacking. C'est un algorithme qui peut regrouper les petites sous-régions en fonction du temps de rendu de ces sous-régions. Le graphique ci-dessous est le regroupement donné par l'algorithme quand il y a 5 processeurs. Les temps de rendu pour les régions dont les bords sont tracés en noir sont 1005, 1011, 996, 952 et 971 secondes respectivement.

Le principe de cet algorithme est de d'abord chercher une combinaison des sous-régions sous forme de rectangle et en condition que la différence entre la somme des temps de rendu des sous-régions dans le rectangle et le temps moyen de rendu soit moins qu'un seuil. On compare la somme des temps avec le temps moyen car le découpage idéal est que le temps de rendu de chaque processeur égal au temps moyen. Après on cherche le deuxième rectangle pour remplir l'image comme dans

le jeu Rectangle Stacking. On cherche un regroupement sous forme de rectangles car maintenant dans le système de Cloud Computing, on ne peut que rendre les régions sous forme de rectangles. L'extrémité se produit à la fin de notre recherche de rectangles car plus on est proche de notre remplissage de l'image, moins on a les choix de combinaisons sous forme de rectangles. Afin de atténuer cela, on effectue la même recherche sur les trois autres bords de l'image et on choisit le découpage le plus équitable. De plus, pour mieux atténuer l'extrémité de découpage, s'il reste moins que 3 rectangles à choisir, l'algorithme énumère tous les choix possibles et choisit ceux qui sont les plus équitables.



Graphique 3.5 – Processus de l'algorithme Rectangle Stacking



Graphique 3.6 – Regroupement donné par Rectangle Stacking en supposant qu'on a 5 processeurs

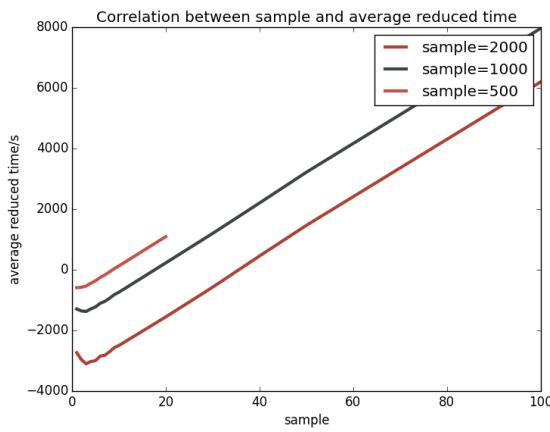
Troisième résultat

Le but de la cinquième étape est de déterminer le nombre d'échantillons à partir duquel on a une estimation efficace des temps de rendu. Il est important de trouver ce nombre car si le nombre d'échantillons utilisés dans l'estimation est trop petit, l'estimation est imprécise à cause des erreurs systématiques dans la mesure. En revanche, si le nombre est trop grand, le temps d'estimation est trop grand et on ne gagnera pas du temps par rapport au découpage par défaut. J'ai analysé les variations de la différence et de l'erreur relative en fonction de variation du nombre d'échantillons utilisés dans l'estimation et trouvé qu'à partir d'un petit nombre d'échantillons, on pouvait avoir une estimation efficace, réaliser la résolution proposée et utiliser moins de temps que le découpage par défaut.

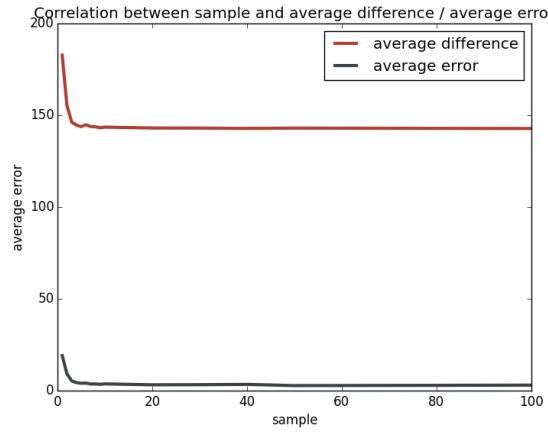
Principalement, les résultats se résument dans les graphiques 3.7 et 3.8. Le graphique 3.7

montre que le temps réduit augmente avec l'augmentation du nombre d'échantillons utilisés dans l'estimation avant un certain nombre d'échantillons. Par exemple, ce seuil est de 3 échantillons quand on fixe le nombre d'échantillons final à 2000. Après, le temps réduit diminue ou la différence entre le temps de rendu en utilisant notre résolution et le temps de rendu en utilisant le système disponible augmente avec l'augmentation du nombre d'échantillons utilisés dans l'estimation. À partir d'environ 40 échantillons, on ne peut pas réduire le temps de rendu en faisant une estimation et en regroupant les sous-régions en fonction du résultat de l'estimation. En revanche, on perd du temps par rapport au découpage par défaut. Donc, il est recommandé d'utiliser 3 échantillons dans l'estimation pour qu'on puisse gagner le plus de temps par rapport au découpage par défaut. Les courbes pour les nombres d'échantillon 500 et 1000 respectivement ont la même tendance. Elles diminuent d'abord avec le nombre d'échantillons et augmentent après. Ce résultat est obtenu après les expériences sur 24 fichiers du type «.blender» et le temps réduit est calculé en prenant la moyenne des temps réduits de ces 24 fichiers. Une remarque sur ce résultat est que le temps de l'estimation est calculé en supposant on a qu'un processeur pour faire l'estimation. En pratique, on a plus qu'un processeur et le temps de l'estimation doit être divisé par le nombre de processeurs disponibles, c'est-à-dire qu'on peut gagner plus de temps en pratique par rapport au découpage par défaut.

Le graphique 3.8 représente la corrélation entre le nombre d'échantillons utilisés dans l'estimation et la moyenne de différence des temps de rendu des sous-régions ou la moyenne d'erreurs relatives. D'abord, pour la différence, on a obtenu la valeur absolue du résultat des temps de rendu de sous-région moins la moyenne des temps de rendu des sous-régions et puis cette valeur a été divisée par la moyenne des temps de rendu des sous-régions. Quand on a eu la différence de toutes les sous-régions, on a calculé la somme de ces données pour chaque fichier à rendre et ensuite calculé la moyenne de cet indice de 24 fichiers qui est représentée par l'ordonnée dans le graphique 3.8. On constate que la différence se réduit avec l'augmentation du nombre d'échantillons utilisés dans l'estimation et puis elle se stabilise. D'abord, quand on utilise un petit nombre d'échantillons, le temps de rendu n'est pas strictement proportionnel à celui-là car il existe des erreurs systématiques pour la mesure du temps. Quand on utilise un petit nombre d'échantillons, les erreurs systématiques sont relativement grandes par rapport au temps de rendu réel. Par conséquent, l'erreur relative est relativement grande pour le temps de rendu réel. Donc, la proportionnalité entre le temps de rendu et le nombre d'échantillons ne peut pas être constatée pour les petits nombres d'échantillons. Sinon, on aurait eu une droite horizontale car la différence moyenne est un ratio et quand on multiplie le numérateur et le dénominateur par le même constant, le ratio ne change pas. Par ailleurs, pour la tendance décroissante dans un premier temps, l'explication est que quand on utilise un petit nombre d'échantillon pour l'estimation, la moyenne des temps de rendu des sous-régions est aussi petite. Donc, la différence relative est grande comme la moyenne est le dénominateur et une petite variation d'une donnée entraîne une grande différence relative. Plus le nombre d'échantillons est petit, plus ce phénomène est évident, ce qui explique la tendance décroissante de la moyenne de différence relative. En second lieu, pour l'erreur moyenne, on a calculé les résultats de découpage avec l'estimation, c'est-à-dire que le temps de rendu pour le rectangle dont la somme des temps de rendu de ses sous-régions était le plus long dans le regroupement en utilisant un plus petit nombre d'échantillons dans l'estimation. Puis, on a calculé aussi les résultats de découpage sans l'estimation, c'est-à-dire qu'on a découpé la région selon les résultats des temps de rendu avec le nombre d'échantillon réel. On a considéré les résultats-ci comme les résultats exacts et on a calculé l'erreur relative des résultats obtenus en utilisant l'estimation. On a fait les expériences en supposant qu'on a 5, 10, 15, 20, 30, 40 et 50



Graphique 3.7 – Corrélation entre le nombre d'échantillon utilisé dans l'estimation et le temps réduit



Graphique 3.8 – Corrélation entre le nombre d'échantillon utilisé dans l'estimation et la moyenne de différence ou la moyenne des erreurs relatives

processeurs et ensuite on a calculé la moyenne de ces erreurs relatives par fichier. Enfin, on a calculé la moyenne des 24 fichiers. Le graphique montre que plus le nombre d'échantillons est petit, plus l'erreur relative est grande. De plus, après un certain nombre d'échantillons qui est 5 dans nos expériences, l'erreur relative se stabilise à un niveau faible. Cela indique qu'avec un petit nombre d'échantillon, on peut déjà obtenir une estimation précise comme avec un nombre d'échantillons beaucoup plus grand.

Quatrième résultat

En recherchant les travaux connexes sur le même sujet, j'ai trouvé qu'il n'y avait pas de travail en estimation du temps de rendu sur le logiciel Blender et donc notre travail est relativement innovant. En outre, je n'ai pas trouvé des algorithmes semblables à l'algorithme Rectangle Stacking. Comme ce résultat est un peu technique, on ne le discutera pas en détail et l'écrit est joint en annexe.

3.2.2 Observations et impressions

Tout d'abord, ce stage m'a montré la importance de collaboration dans le travail. Mon projet n'aurait pas été fini si je travaillais toute seule. Comme mon projet concernait l'usage du logiciel Blender et l'usage du système Linux que je n'avais jamais utilisé avant ce stage, j'ai surmonté certaines difficultés avec l'aide de mon maître de stage et des membres des autres équipes. Par exemple, c'était mon collègue Gwenn qui m'a proposé d'utiliser la mesure du temps dans Blender quand j'ai trouvé que le temps du chargement des fichiers était trop important en utilisant la résolution proposée. En effet, mon projet s'agissait du service de rendu dans le système de Cloud Computing qui était maintenu par plusieurs équipes chez Qarnot Computing. C'était nécessaire de collaborer avec les autres équipes pour que la résolution sur laquelle je travaillais soit faisable en pratique. Dans le futur, la mise en place de ma résolution requiert également la collaboration des différentes équipes.

En second lieu, ce stage m'a montré la nécessité de la communication entre les différentes équipes et entre les différents niveaux dans la hiérarchie. D'un côté, dans un premier temps, je ne savais

pas qu'il y avait une grande négligence dans la résolution proposée qui aurait entraîné l'échec de mon projet. Dans la résolution proposée, on doit d'abord découper la région à rendre de façon très fine. Mais au milieu de mon stage, quand je communiquais avec les membres de l'équipe du système de Cloud Computing, il m'a dit qu'on ne pouvait pas effectuer une découpage de façon très fine qui coûtait trop de temps dans le chargement des fichiers. Mon projet aurait été infaisable si je ne communiquais pas avec eux. De l'autre côté, comme je travaillais relativement indépendamment sur mon projet, c'était important pour moi de communiquer l'avancement de mon projet à mon maître de stage, Yanik. Pour mon maître de stage, il fallait qu'il bien communique l'avancement de ce projet au président-directeur général.

Finalement, ce stage m'a montré ce qu'un data scientist faisait dans une PME. Au lieu de traiter un aspect concret, un data scientiste dans une PME pouvait s'occuper de tous les sujets concernant les données. Mon maître de stage est un data scientiste chez Qarnot Computing. Il n'y avait pas une fonction fixée dans son travail. Il s'occupait de mon projet, le traitement du signal etc. Il collaborait avec les autres équipes sur les sujets concrets pour améliorer les services offerts par Qarnot Computing.

Bilan de stage

D'abord, j'ai acquis les connaissances concernant la data science dans mon stage. Dans la première semaine, j'ai effectué une analyse de la statistique descriptive sur les données et j'ai appris à utiliser les packages de Python pour obtenir les indices statistiques et à dessiner les graphiques statistiques. Dans les trois semaines après, je me suis concentrée sur la collection des données. C'était une étape importante pour mon projet de data science. Dans un premier temps, j'ai collecté les données erronées et cela a totalement perturbé le projet comme les données montrait qu'on ne pouvait pas effectuer une estimation avec un petit nombre d'échantillons. Heureusement, on a corrigé l'erreur et continué le projet. Pour analyser les données, j'ai appris à utiliser le package pandas pour lire, traiter et sauvegarder les données proprement. Dans cette étape, j'ai utilisé la régression linéaire que j'ai apprise à l'école et j'ai appris comment l'utiliser en Python. J'ai appris aussi un autre modèle qui s'appelait K-means pour l'analyse des données.

En second lieu, j'ai aussi enrichi mes connaissances en informatique. Dans le stage, bien que le sujet s'agissait de la data science, il me fallait aussi avoir les connaissances sur le logiciel Blender et sur le système de Cloud Computing. Pour obtenir les connaissances sur le logiciel Blender, j'ai consulté le mode d'emploi de Blender et également la documentation officielle du package bpy qui permettait l'usage de Blender en Python. En outre, j'ai aussi posé des questions à un collègue qui connaissait bien Blender. Pour mieux connaître le système de Cloud Computing, j'ai posé souvent des questions aux membres de l'équipe de développement du système de Cloud Computing. De plus, comme le sujet était mis en oeuvre sur le système Linux, j'ai consulté sur internet pour maîtriser les commandes de Linux et pour savoir comment programmer en Bash. Par ailleurs, j'ai aussi appris à modifier le open-source logiciel Blender pour qu'il réponde à notre besoin, ce qui s'agissait de la construction des open-source logiciels. Enfin, comme je codais en Python pendant le stage, je suis devenue plus familière avec ce langage.

En troisième lieu, j'ai découvert le monde professionnel pour un data scientiste dans une PME. J'ai mieux compris ce qu'un data scientiste faisait et comment une PME fonctionnait, ce qui serait utile pour moi de réfléchir sur mon projet professionnel.

Finalement, comme une étrangère, j'ai bien amélioré mon français dans ce stage. Au début, c'était difficile pour moi de comprendre les conversations entre mes collègues. Pendant le stage, je mangeais avec mes collègues tous les midis et je communiquais aussi avec mon maître de stage et mes collègues. À la fin du stage, j'ai pu comprendre en gros les conversations et même certaines blagues de mes collègues.

Le seul aspect négatif de ce stage est que le projet s'appuie un peu trop sur les connaissances en informatique. Par conséquent, j'ai passé un peu trop de temps sur les tâches concernant l'aspect informatique et en fait, je voulais acquérir plutôt les connaissances en Data Science.

Conclusion

Pour une présentation générale de l'organisme d'accueil, avec l'idée innovante d'utiliser la chaleur émise par les processeurs, Qarnot Computing se développe rapidement depuis sa fondation en 2010. Chez Qarnot Computing, les équipes s'efforcent de fournir les radiateurs intelligents et le service stable et efficace de Cloud Computing. Cette entreprise a une structure affinée et flexible avec les petites équipes et la possibilité pour une personne d'appartenir à plus d'une équipe.

En outre, en termes d'une présentation profonde de Qarnot Computing, physiquement, elle fournit un environnement physique favorable pour coder et pour la communication. Les grands écrans, l'application de communication, la mise en place du service de mail et les endroits pour discuter facilitent le travail. L'environnement culturel de l'entreprise m'a aussi impressionnée. Une réunion chaque semaine pour chaque équipe de présenter leur travail, les activités diverses, le fait que chaque membre est toujours occupé et la passion de son personnel montrent la cohésion, la vitalité et la passion de son environnement culturel. Quant à l'organisation, on trouve une hiérarchie de PDG-chef de l'équipe-membres de l'équipe. Mais concrètement dans le travail, on est donné une grande liberté. Quant aux relations professionnelles, j'entendais bien avec les autres. Surtout, j'apprécie la grande liberté que mon maître de stage m'a donnée dans mon travail et l'égalité que je sentais dans les conversations avec lui. L'entreprise a aussi une relation harmonieuse avec ses clients.

Finalement, mon stage était sur le sujet «Découpage intelligent dans Blender», c'est-à-dire de bien distribuer les tâches de rendu aux processeurs pour que les temps de rendu pour les processeurs soient plus équitables et par conséquent, le temps total de rendu soit réduit. Selon la proposition de Paul et un collègue, j'ai utilisé une méthode semblable à la résolution s'agissant du dilemme exploration-exploitation. J'ai vérifié la proportionnalité entre le nombre d'échantillons ou la taille de l'image finale et le temps de rendu, créé un algorithme qui peut effectuer un découpage intelligent et vérifié qu'un petit nombre d'échantillons assurait une estimation précise. Par ailleurs, la collaboration et la communication avec les autres personnels dans l'entreprise étaient nécessaires afin de finir mon projet. La proposition de présentation de mon travail a été accepté dans la Conférence de Blender qui aura lieu en octobre 2016 à Amsterdam. L'application de mon projet est aussi envisageable, ce qui permettrait Qarnot Computing à fournir un service de rendu plus rapide.

Ce stage m'a permis de découvrir le monde professionnel d'une data scientist dans une PME, a enrichi mes connaissances en Data Science et en informatique et m'a préparée pour l'intégration future dans le monde professionnel.

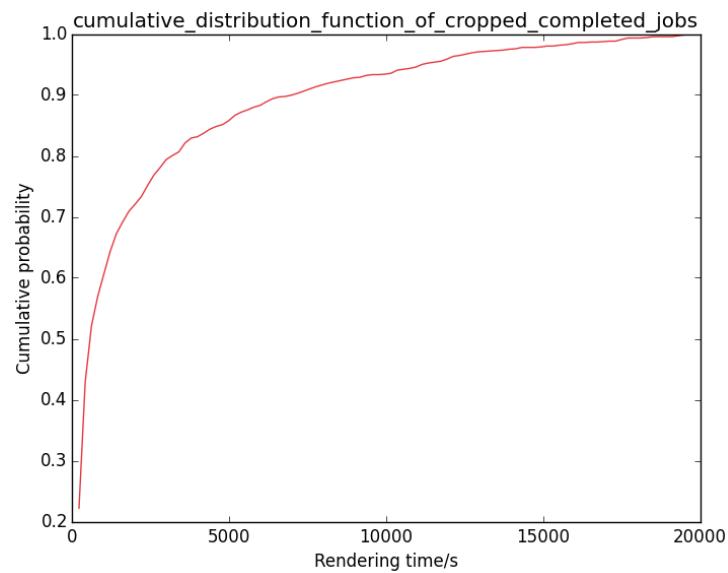
Bibliographie

- [1] Document interne.
- [2] Wikipédia de «Cloud computing».
- [3] Wikipédia de «Rendering».
- [4] Mode d'emploi de Blender 2.77.
- [5] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs, "A Sorting Classification of Parallel Rendering," *IEEE Computer Graphics and Applications*, Vol 14, No. 4, July 1994, pp. 23-32.
- [6] Rudrajit Samanta, Jiannan Zheng, Thomas Funkhouser, Kai Li, and Jaswinder Pal Singh, "Load Balancing for Multi-Projector Rendering Systems," *HWWS '99 Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, July 1999, pp. 107-116.
- [7] Erik Reinhard, Arjan J. F. Kok, Alan Chalmers, "Cost Distribution Prediction for Parallel Ray Tracing," *Second Eurographics Workshop on Parallel Graphics and Visualisation*, September 1998, pp. 77-90.
- [8] Sanjay Goil, and Sanjay Ranka, "Dynamic Load Balancing for Raytraced Volume Rendering on Distributed Memory Machines," May 1996.
- [9] Michael Wimmer, Peter Wonka, "Rendering Time Estimation for Real-Time Rendering," *Rendering Techniques 2003 (Proceedings of the Eurographics Symposium on Rendering 2003)*, June 2003, pp. 118-129.
- [10] Richard Gillibrand, Kurt Debattista, and Alan Chalmers, "Cost Prediction Maps for Global Illumination," *EG UK Theory and Practice of Computer Graphics*, 2005, pp. 97-104.
- [11] Richard Gillibrand, Peter Longhurst, Kurt Debattista, and Alan Chalmers, "Cost Prediction for Global Illumination using a Fast Rasterised Scene Preview," *AFRIGRAPH 2006 4th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, 2006, pp. 41-48.
- [12] Biagio Cosenza, Carsten Dachsbaecher, and Ugo Erra, "GPU Cost Estimation for Load Balancing in Parallel Ray Tracing," *International Conference on Computer Graphics Theory and Applications (GRAPP)*, 2013, pp. 139-151.
- [13] Ali Pinar, and Cevdet Aykanat, "Fast optimal load balancing algorithms for 1D partitioning," *Journal of Parallel and Distributed Computing*, August 2004, Vol 64, No. 8, pp. 974-996.
- [14] Bjørn Olstad, and Fredrik Manne, "Efficient partitioning of sequences," *IEEE Transactions on Computers*, November 1995, Vol 44, No. 11, pp. 1322-1326.
- [15] Fredrik Manne, and Tor Sørevik, "Optimal partitioning of sequences," *Journal of Algorithms*, September 1995, Vol 19, No. 2, pp. 235-249.

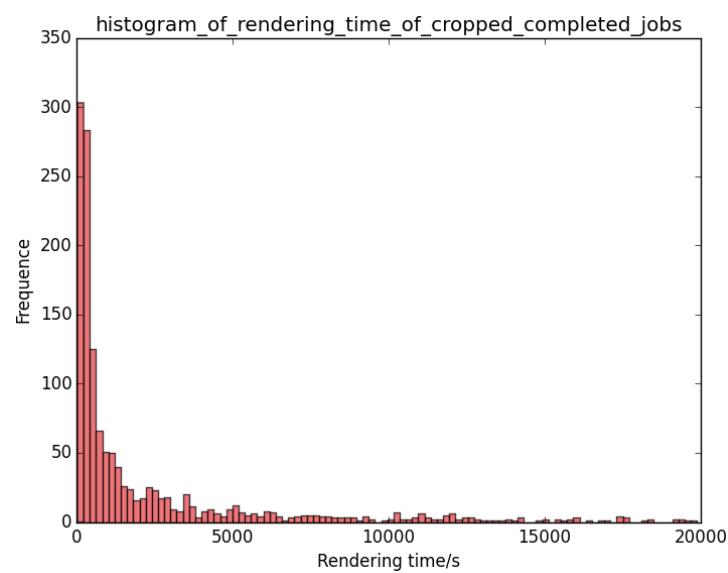
- [16] Serge Miguet, and Jean-Marc Pierson, "Heuristics for 1D Rectilinear Partitioning as a Low Cost and High Quality Answer to Dynamic Load Balancing," *Lecture Notes in Computer Science*, 25 June 2005, Vol 1225, pp. 550-564. Springer Verlag.
- [17] Y. Han, B.Narahari, and H-A. Choi, "Mapping a Chain Task to Chained Processors," *Information Processing Letters*, 30 November 1992, Vol 44, No. 3, pp. 141-148.
- [18] Sanjeev Khanna, S. Muthukrishnan, and Steven Skiena, "Efficient Array Partitioning," *ICALP '97 Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, 1997, pp. 616-626.
- [19] Michelangelo Grigni, and Fredrik Manne, "On the Complexity of the Generalized Block Distribution," *Proc. of IRREGULAR '96*, 1996, pp. 319-326.
- [20] Erik Saule, Erdeniz Ö. Baş, and Ümit V. Çatalyürek, "Partitioning Spatially Located Computations using Rectangles," *Conference : 25th IEEE International Symposium on Parallel and Distributed Processing*, 2011, pp. 16-20.
- [21] David M. Nicol, "Rectilinear Partitioning of Irregular Data Parallel Computations," *Journal of Parallel and Distributed Computing*, November 1994, pp. 119-134.
- [22] Fredrik Manne, and Tor Sórevik, "Partitioning an Array onto a Mesh of Processors," *Proc. of the Workshop on Applied Parallel Computing in Industrial Problems*, 1996.
- [23] Carl Mueller, "The Sort-First Rendering Architecture for High-Performance Graphics," *I3D '95 Proceedings of the 1995 symposium on Interactive 3D graphics*, 1995, pp. 75-ff.
- [24] Daya Ram Gaur, Toshihide Ibaraki, and Ramesh Krishnamurti, "Constant Ratio Approximation Algorithms for the Rectangle Stabbing Problem and the Rectilinear Partitioning Problem," *Journal of Algorithms*, Vol 43, No. 1, April 2002, pp. 138-152.
- [25] S. Muthukrishnan, and Torsten Suel, "Approximation algorithms for array partitioning problems," *Journal of Algorithms*, Vol 54, No. 1, January 2005, pp. 85-104.
- [26] Stéphane Marchesin, Catherine Mongenet, and Jean-Michel Dischler, "Dynamic load balancing for parallel volume rendering," *EGPGV '06 Proceedings of the 6th Eurographics conference on Parallel Graphics and Visualization*, 2006, pp. 43-50.

Appendix A

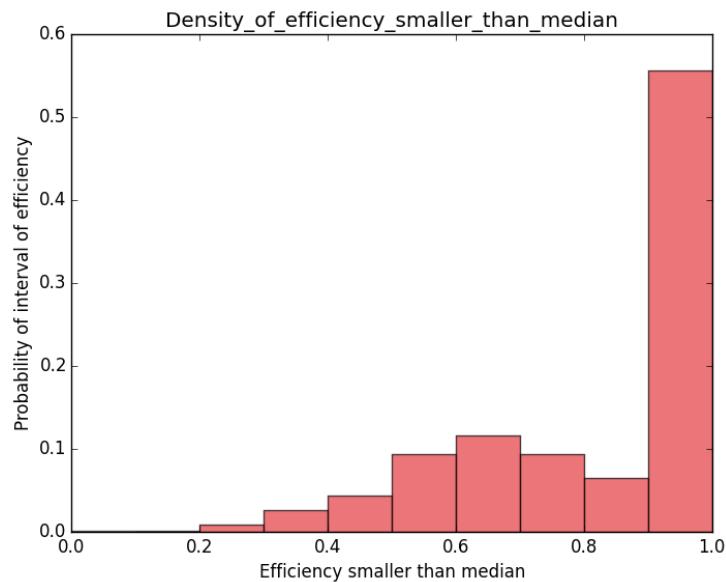
Résultat de l'analyse de la statistique descriptive



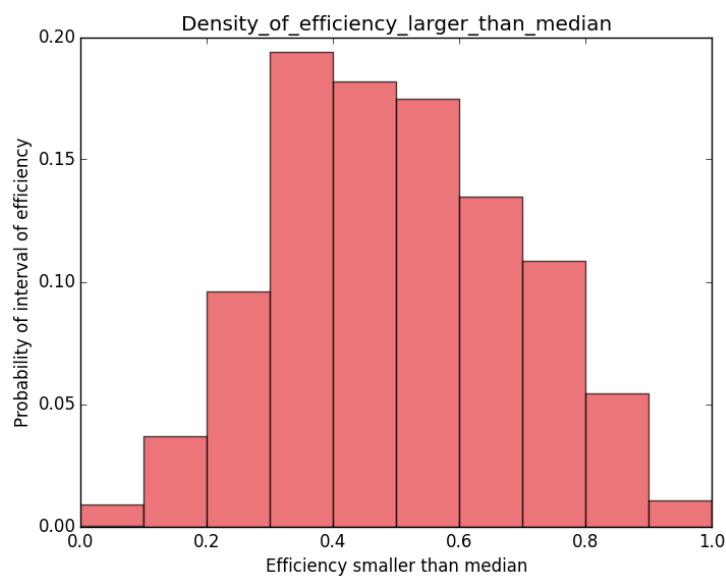
Graphique A..1 – Fonction de répartition des tâches de rendu complétées



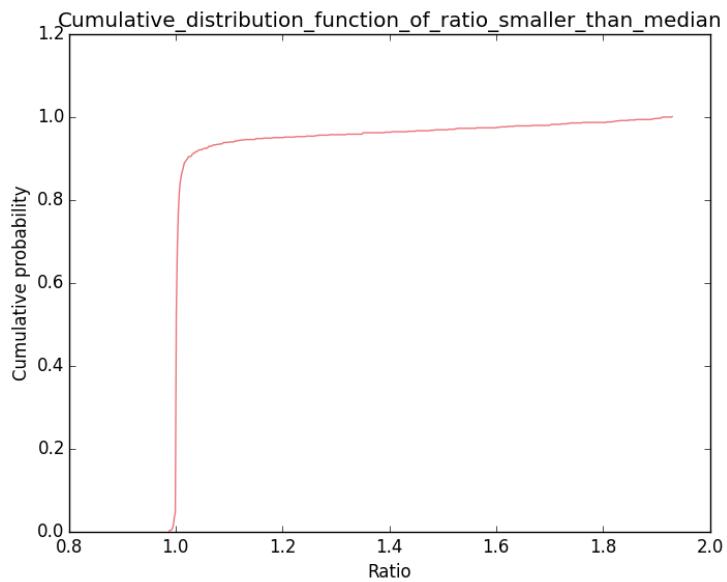
Graphique A..2 – Histogramme des temps de rendu des tâches complétées



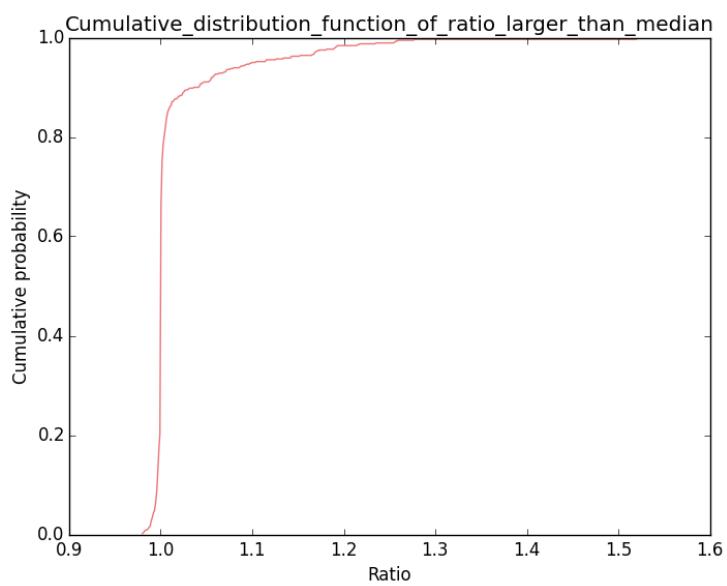
Graphique A..3 – Densité de l'efficacité avant la médiane



Graphique A..4 – Densité de l'efficacité après la médiane



Graphique A..5 – Fonction de répartition de ratio avant la médiane



Graphique A..6 – Fonction de répartition de ratio après la médiane

Appendix B

Code de l'algorithme Rectangle Stacking

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import random, math, copy, csv, pdb
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from pandas import DataFrame, Series

processors_num = 10
chunks_edge = 16
chunks_num = chunks_edge ** 2
kTestData = pd.read_csv("test_data.csv")

def EnumerationSplitOn2(data_matrix, position_x, position_y):
    possible_results = []
    width = data_matrix.shape[1]
    height = data_matrix.shape[0]
    if width!=1:
        for i in range(width - 1):
            chunks = [data_matrix[:, :i + 1], data_matrix[:, i + 1:]]
            possible_results.append(max([float(np.sum(chunk)) for chunk in chunks]))
    if height!=1:
        for i in range(height - 1):
            chunks = [data_matrix[:i + 1, :], data_matrix[i + 1:, :]]
            possible_results.append(max([float(np.sum(chunk)) for chunk in chunks]))
    best_choice_index = possible_results.index(min(possible_results))
    if best_choice_index <= width - 2:
        subwidth = best_choice_index + 1
        subheight = height
        y = subwidth
    return [position_x, position_y, subheight, subwidth], [position_x, y + pos]
    # we set the southeast corner as the origin of our coordinate system. How
```

```

# right side and vertical coordinate increases along the up side
else:
    subwidth = width
    subheight = best_choice_index - (width - 1) + 1
    x = height - subheight
    return [position_x + x, position_y, subheight, subwidth], [position_x, po
def EnumerationSplitOn3(data_matrix, position_x, position_y, original_height):
    final_split = []
    possible_results = []
    width = data_matrix.shape[1]
    height = data_matrix.shape[0]
    possible_splits = []
    possible_chunks = []
    if width!=1:
        for i in range(width - 1):
            chunks = [data_matrix[:, :i + 1], data_matrix[:, i + 1:]]
            selected_chunk_no = int(float(np.sum(chunks[0])) < float(np.sum(chunks[1])))
            if chunks[selected_chunk_no].shape[0] * chunks[selected_chunk_no].shape[1] == 0:
                selected_chunk_no = int(selected_chunk_no == 0)
            temp_split_on_2 = EnumerationSplitOn2(chunks[selected_chunk_no], position_x, position_y + (i + 1) * int(selected_chunk_no), original_height)
            possible_splits.append((temp_split_on_2, selected_chunk_no))
            chunks = [chunks[selected_chunk_no * (-1) + 1], data_matrix[:, original_height - temp_split_on_2[0][0] - temp_split_on_2[0][1]:0], data_matrix[:, temp_split_on_2[0][1] - position_y:temp_split_on_2[0][1] + temp_split_on_2[0][0]], temp_split_on_2[1][1] - position_y:temp_split_on_2[1][1] + temp_split_on_2[1][0]]
            possible_chunks.append(chunks)
            possible_results.append(max([float(np.sum(chunk)) for chunk in chunks]))
    if height!=1:
        for i in range(height - 1):
            chunks = [data_matrix[:i + 1, :], data_matrix[i + 1:, :]]
            selected_chunk_no = int(float(np.sum(chunks[0])) < float(np.sum(chunks[1])))
            if chunks[selected_chunk_no].shape[0] * chunks[selected_chunk_no].shape[1] == 0:
                selected_chunk_no = int(selected_chunk_no == 0)
            temp_split_on_2 = EnumerationSplitOn2(chunks[selected_chunk_no], position_x + (height - i - 1) * int(selected_chunk_no), position_y, original_height)
            possible_splits.append((temp_split_on_2, selected_chunk_no))
            chunks = [chunks[selected_chunk_no * (-1) + 1], data_matrix[:, original_height - temp_split_on_2[0][0] - temp_split_on_2[0][1]:0], data_matrix[:, temp_split_on_2[0][1] - position_y:temp_split_on_2[0][1] + temp_split_on_2[0][0]]]
            possible_chunks.append(chunks)
            possible_results.append(max([float(np.sum(chunk)) for chunk in chunks]))

```

```

        0] ,
temp_split_on_2[0][1] - position_y : temp_split_on_2[0][1] +
data_matrix[
original_height - temp_split_on_2[1][0] - temp_split_on_2[1][1],
0],
temp_split_on_2[1][1] - position_y : temp_split_on_2[1][1] +
possible_chunks.append(chunks)
possible_results.append(max([float(np.sum(chunk)) for chunk in chunks]))
minimum_index = possible_results.index(min(possible_results))
final_split.append(possible_splits[minimum_index][0][0])
final_split.append(possible_splits[minimum_index][0][1])
if minimum_index <= width - 2:
    subwidth = minimum_index + 1
    subheight = height
    y = subwidth
    if possible_splits[minimum_index][1]:
        final_split.append([position_x, position_y, subheight, subwidth])
    else:
        final_split.append([position_x, y + position_y, subheight, width - subwidth])
return final_split
else:
    subwidth = width
    subheight = minimum_index - (width - 1) + 1
    x = height - subheight
    if possible_splits[minimum_index][1]:
        final_split.append([x + position_x, position_y, subheight, subwidth])
    else:
        final_split.append([position_x, position_y, height - subheight, subwidth])
return final_split

def GetDataMatrix(raw_data, name, ratio, chunks_edge):
predict_data_matrix = [list(raw_data[raw_data['name'].isin([name]) & raw_data['position_y'].isin([i]) & raw_data['ratio'].isin([ratio])]['rendering_time']) for i in range(chunks_edge - 1, -1, -1)]
final_data_matrix = [list(raw_data[raw_data['name'].isin([name]) & raw_data['position_y'].isin([i]) & raw_data['ratio'].isin([ratio])]['rendering_time']) for i in range(chunks_edge - 1, -1, -1)]
return np.array(predict_data_matrix), np.array(final_data_matrix)

class Chunk(object):
"""chunk of the split"""

```

```

def __init__(self, subdata_matrix=np.zeros((1, 1)), position_x=0, position_y=0):
    self.subdata_matrix_ = subdata_matrix
    self.position_x_ = position_x
    self.position_y_ = position_y
    self.subheight_ = self.subdata_matrix_.shape[0]
    self.subwidth_ = self.subdata_matrix_.shape[1]
    self.subtotal_time_ = float(np.sum(self.subdata_matrix_))

class Image(object):
    """the image to be split"""

    def __init__(self, data_matrix, final_data_matrix, processors_num):
        self.data_matrix_ = data_matrix
        self.final_data_matrix_ = final_data_matrix
        self.height_ = self.data_matrix_.shape[0]
        self.width_ = self.data_matrix_.shape[1]
        self.chunks_ = []
        self.processors_num_ = processors_num
        self.total_time_ = float(np.sum(self.data_matrix_))
        self.unit_time_ = self.total_time_ / (self.height_ * self.width_) / 100
        self.average_time_ = self.total_time_ / self.processors_num_
        self.current_surface_chunk_index_ = 0
        self.undistributed_time_ = 0
        self.chunk_surfaces_ = [(0, 0, self.width_)]

    def ResetParameters(self):
        self.height_ = self.data_matrix_.shape[0]
        self.width_ = self.data_matrix_.shape[1]
        self.chunks_ = []
        self.average_time_ = self.total_time_ / self.processors_num_
        self.current_surface_chunk_index_ = 0
        self.undistributed_time_ = 0
        self.chunk_surfaces_ = [(0, 0, self.width_)]

    def ChangeAverageTime(self):
        total_time = 0
        for i in self.chunks_:
            total_time += i.subtotal_time_
        self.undistributed_time_ = self.total_time_ - total_time
        self.average_time_ = self.undistributed_time_ / (self.processors_num_ - 1)

    def IsValidUndistributedChunks(self, width, height):
        totalUndistributedArea = 0
        for chunk in self.chunks_:
            totalUndistributedArea += chunk.subheight_ * chunk.subwidth_
        if (self.height_ * self.width_ - totalUndistributedArea - width * height) < 0:
            return False
        else:
            return True

```

```

        self.processors_num_ - len(self.chunks_) - 1):
    return True
else:
    return False

def PutChunk(self):
    if len(self.chunk_surfaces_) == self.processors_num_ - len(self.chunks_):
        if len(self.chunk_surfaces_) > 1:
            chunks = []
            distance_to_average_time = []
            for i in range(len(self.chunk_surfaces_)):
                vertical_chunk = Chunk(self.data_matrix_[self.height_ - self.
                                                chunk_surfaces_[i][1]:self.chunk_
                                                surfaces_[i][0] - 1])
                self.chunk_surfaces_[i][0], self.chunk_
                self.chunk_surfaces_[i][1]:self.chunk_
                if i == len(self.chunk_surfaces_) - 1:
                    chunks.append(vertical_chunk)
                    distance_to_average_time.append(abs(vertical_chunk.subtotal_
                    else:
                        if self.chunk_surfaces_[i][1] + self.chunk_surfaces_[i][2] <
                            chunks.append(vertical_chunk)
                            distance_to_average_time.append(abs(vertical_chunk.subtotal_
                        else:
                            if self.chunk_surfaces_[i][0] > self.chunk_surfaces_[i][1]:
                                horizontal_chunk = Chunk(
                                    self.data_matrix_[self.height_ - self.chunk_
                                    self.chunk_surfaces_[i + 1][1]:self.chunk_
                                    self.chunk_surfaces_[i + 1][0], self.chunk_
                            else:
                                horizontal_chunk = Chunk(
                                    self.data_matrix_[self.height_ - self.chunk_
                                    self.chunk_surfaces_[i][1]:self.chunk_
                                    self.chunk_surfaces_[i][0], self.chunk_
                                if abs(vertical_chunk.subtotal_time_ - self.average_time) <
                                    horizontal_chunk.subtotal_time_ - self.average_time:
                                        chunks.append(horizontal_chunk)
                                        distance_to_average_time.append(
                                            abs(horizontal_chunk.subtotal_time_ - self.average_
                                else:
                                    chunks.append(vertical_chunk)
                                    distance_to_average_time.append(abs(vertical_chunk.subtotal_
selected_chunk_index = distance_to_average_time.index(min(distance_

```

```

        self.chunks_.append(chunks[selected_chunk_index])
        self.ChangeAverageTime()
        if chunks[selected_chunk_index].position_x_ + chunks[selected_chunk_index].width_ > deleted_chunk_index:
            deleted_chunk_index = selected_chunk_index
        if selected_chunk_index != len(self.chunk_surfaces_) - 1:
            if chunks[selected_chunk_index].position_x_ + chunks[selected_chunk_index].width_ > deleted_chunk_index:
                self.chunk_surfaces_[selected_chunk_index][0] and \
                    self.chunk_surfaces_[selected_chunk_index][2] == chunks[
                        selected_chunk_index].position_y_:
                    self.chunk_surfaces_[selected_chunk_index] = (
                        self.chunk_surfaces_[selected_chunk_index][0],
                        self.chunk_surfaces_[selected_chunk_index][1],
                        self.chunk_surfaces_[selected_chunk_index][2] + chunks[
                            selected_chunk_index].width_)
                    deleted_chunk_index = selected_chunk_index + 1
            if chunks[selected_chunk_index].position_x_ + chunks[selected_chunk_index].width_ > deleted_chunk_index:
                self.chunk_surfaces_[selected_chunk_index + 1][0] and \
                    selected_chunk_index].position_y_ + chunks[selected_chunk_index].width_ > \
                    self.chunk_surfaces_[selected_chunk_index + 1][1]:
                    self.chunk_surfaces_[selected_chunk_index] = (self.chunk_surfaces_[selected_chunk_index][0],
                                                                self.chunk_surfaces_[selected_chunk_index][1],
                                                                self.chunk_surfaces_[selected_chunk_index][2] + chunks[
                                                                    selected_chunk_index].width_)
                    deleted_chunk_index = selected_chunk_index + 1
        self.chunk_surfaces_.remove(self.chunk_surfaces_[deleted_chunk_index])
        if len(self.chunk_surfaces_) > 1:
            deleted_chunks_indices = []
            for i in range(len(self.chunk_surfaces_) - 1):
                if self.chunk_surfaces_[i + 1][0] == self.chunk_surfaces_[i][0] and \
                    self.chunk_surfaces_[i][2] == self.chunk_surfaces_[i + 1][1]:
                    deleted_chunks_indices.append(i)
            for i in deleted_chunks_indices:
                self.chunk_surfaces_[i] = (
                    self.chunk_surfaces_[i][0], self.chunk_surfaces_[i][1],
                    self.chunk_surfaces_[i][2] + self.chunk_surfaces_[i + 1][2])
                self.chunk_surfaces_.remove(self.chunk_surfaces_[i + 1])
            self.current_surface_chunk_index_ = 0
        else:
            self.chunks_.append(Chunk(self.data_matrix_[:self.height_ - self.chunk_surfaces_[0][1]:self.chunk_surfaces_[0][2]],
                                      self.chunk_surfaces_[0][0], self.chunk_surfaces_[0][1]))
        surface_chunk_position_x = self.chunk_surfaces_[self.current_surface_chunk_index_]
        surface_chunk_position_y = self.chunk_surfaces_[self.current_surface_chunk_index_]
        surface_chunk_width = self.chunk_surfaces_[self.current_surface_chunk_index_]
    
```

```

is_valid_chunk = False
is_better_split_on_2 = False
is_better_split_on_3 = False
threshold = self.unit_time_
for i in range(len(self.chunk_surfaces_)):
    if 0 < i < len(self.chunk_surfaces_) - 1:
        if self.chunk_surfaces_[i][1] != self.chunk_surfaces_[i - 1][1]:
            self.chunk_surfaces_[i][1] += \
                self.chunk_surfaces_[i][2] != self.chunk_surfaces_[i - 1][2]
temp_chunk = Chunk(self.data_matrix_[:self.height_ - self.chunk_surfaces_[i][1]:self.chunk_surfaces_[i][1]+1],
                    self.chunk_surfaces_[i][0], self.chunk_surfaces_[i][1])
if temp_chunk.subtotal_time_ > self.average_time_ * (float(len(self.chunk_surfaces_)) / self.processors_num_ - 1 / self.chunks_) - 2:
    is_better_split_on_3 = True
    break
if temp_chunk.subtotal_time_ > self.average_time_ * (float(len(self.chunk_surfaces_)) / self.processors_num_ - 1 / self.chunks_) - 2 and len(self.chunk_surfaces_) == 1) and (self.processors_num_ - len(self.chunk_surfaces_) == 1, 2)):
    is_better_split_on_2 = True
    break
if i == 0:
    if len(self.chunk_surfaces_) == 1 or self.chunk_surfaces_[i][1] != self.chunk_surfaces_[i + 1][1]:
        temp_chunk = Chunk(self.data_matrix_[:self.height_ - self.chunk_surfaces_[i][1]:self.chunk_surfaces_[i][1]+1],
                            self.chunk_surfaces_[i][0], self.chunk_surfaces_[i][1])
        if temp_chunk.subtotal_time_ > self.average_time_ * (float(len(self.chunk_surfaces_)) / self.processors_num_ - 1 / self.chunks_) - 2:
            is_better_split_on_3 = True
            break
        if temp_chunk.subtotal_time_ > self.average_time_ * (float(len(self.chunk_surfaces_)) / self.processors_num_ - 1 / self.chunks_) - 2 and len(self.chunk_surfaces_) == 1) and self.processor
self.chunk_surfaces_) in (1, 2):
            is_better_split_on_2 = True

```

```

        break
    if i == len(self.chunk_surfaces_) - 1:
        if self.chunk_surfaces_[i][1] != self.chunk_surfaces_[i - 1][1]:
            temp_chunk = Chunk(self.data_matrix_[:self.height_ - self.chunk_surfaces_[i][1]:self.chunk_surfaces_[i][1]:self.chunk_surfaces_[i][1]], self.chunk_surfaces_[i][0], self.chunk_surfaces_[i][1])
            if temp_chunk.subtotal_time_ > self.average_time_ * (float(len(self.chunk_surfaces_)) / self.processors_num_ - 1 / self.chunks_) - 2:
                is_better_split_on_3 = True
            break
        if temp_chunk.subtotal_time_ > self.average_time_ * (float(len(self.chunk_surfaces_)) / self.processors_num_ - 1 / self.chunks_) - 2 and len(self.chunk_surfaces_) == 1) and (
                self.processors_num_ - len(self.chunk_surfaces_) == 1, 2)):
            is_better_split_on_2 = True
        break
    if len(self.chunk_surfaces_) == self.processors_num_ - len(self.chunks_):
        if is_better_split_on_2:
            split_on_2 = EnumerationSplitOn2(temp_chunk.subdata_matrix_, temp_chunk.position_y_)
            self.chunks_.append(Chunk(self.data_matrix_[self.height_ - split_on_2[0][0] : self.height_ - split_on_2[0][1] + split_on_2[0][1]]))
            self.chunks_.append(Chunk(self.data_matrix_[self.height_ - split_on_2[1][0] : self.height_ - split_on_2[1][1] + split_on_2[1][1]]))
            self.chunk_surfaces_.remove(self.chunk_surfaces_[i])
        if len(self.chunk_surfaces_):
            self.current_surface_chunk_index_ %= len(self.chunk_surfaces_)
        if self.processors_num_ != len(self.chunks_):
            self.ChangeAverageTime()
    else:
        while not is_valid_chunk:
            vertical_chunks = []
            horizontal_chunks = []
            distance_to_average_time = []
            if len(self.chunk_surfaces_) == 1:
                limit_height = self.height_ - surface_chunk_position_

```

```

        limit_width = surface_chunk_width
    else:
        limit_height = self.height_ - surface_chunk_position_
        limit_width = surface_chunk_width + 1
    for i in range(1, limit_height):
        temp_chunk = Chunk(self.data_matrix_[self.height_ - surface_chunk_position_:surface_chunk_position_y:surface_chunk_position_x, surface_chunk_position_x + i, surface_chunk_position_y:i])
        upperUndistributedChunk = Chunk(
            self.data_matrix_[:self.height_ - surface_chunk_position_, surface_chunk_position_y:surface_chunk_position_y:surface_chunk_position_x + i, surface_chunk_position_y])
        if abs(
                temp_chunk.subtotal_time_ - self.average_time,
                temp_chunk.subwidth_,
                temp_chunk.subheight_) and (
                    upperUndistributedChunk.subtotal_time -
                    vertical_chunks.append(temp_chunk)
                    distance_to_average_time.append(abs(temp_chunk.subtotal_time_ - self.average_time)))
    for j in range(1, limit_width):
        temp_chunk = Chunk(self.data_matrix_[:self.height_ - surface_chunk_position_y:surface_chunk_position_x:surface_chunk_position_x, surface_chunk_position_y + j, surface_chunk_position_y])
        rightUndistributedChunk = Chunk(
            self.data_matrix_[:self.height_ - surface_chunk_position_y:surface_chunk_position_y + j:surface_chunk_position_x, surface_chunk_position_x, surface_chunk_position_y])
        if abs(
                temp_chunk.subtotal_time_ - self.average_time,
                temp_chunk.subwidth_,
                temp_chunk.subheight_) and (
                    rightUndistributedChunk.subtotal_time -
                    horizontal_chunks.append(temp_chunk)
                    distance_to_average_time.append(abs(temp_chunk.subtotal_time_ - self.average_time)))
    if len(vertical_chunks) or len(horizontal_chunks):
        is_valid_chunk = True
        selected_chunk_index = distance_to_average_time.index(min(distance_to_average_time))
    if is_valid_chunk:
        if selected_chunk_index < len(vertical_chunks) and len(vertical_chunks) > 1:
            temp_chunk = vertical_chunks[selected_chunk_index]
            self.chunks_.append(temp_chunk)
            self.ChangeAverageTime()
            self.chunk_surfaces_[self.current_surface_chunk_index][surface_chunk_position_x + temp_chunk.subheight_, surface_chunk_position_y:surface_chunk_position_y + temp_chunk.subwidth_]
        if temp_chunk.subheight_ == self.height_ - surface_chunk_position_y:
            self.chunks_.append(temp_chunk)
            self.ChangeAverageTime()
            self.chunk_surfaces_[self.current_surface_chunk_index][surface_chunk_position_x + temp_chunk.subheight_, surface_chunk_position_y:surface_chunk_position_y + temp_chunk.subwidth_]

```

```

        self.chunk_surfaces_.remove(self.chunk_surface_
        self.current_surface_chunk_index_ % len(self.chunk_
else:
    deleted_chunks_indices = []
    if self.current_surface_chunk_index_ > 0:
        if self.chunk_surfaces_[self.current_surface_
            self.chunk_surfaces_[self.current_
                self.chunk_surfaces_
                    self.chunk_surfaces_[self.current_
                        self.chunk_surfaces_[self.current_
                            self.chunk_surfaces_[self.current_
                                self.chunk_surfaces_[self.current_
                                    self.chunk_surfaces_[self.current_
                                        self.chunk_surfaces_[self.current_
                                            self.chunk_surfaces_[self.current_
                                                self.chunk_surfaces_[self.current_
                                                    self.chunk_surfaces_[self.current_
                                                        self.chunk_surfaces_[self.current_
                                                            self.chunk_surfaces_[self.current_
                                                                self.chunk_surfaces_[self.current_
                                                                    self.chunk_surfaces_[self.current_
                                                                        self.chunk_surfaces_[self.current_
                                                                            self.chunk_surfaces_[self.current_
                                                                                self.chunk_surfaces_[self.current_
                                                                                    self.chunk_surfaces_[self.current_
                                                                
second_temp_chunk = self.chunk_surfaces_[self.current_
for k in range(len(deleted_chunks_indices)):
    self.chunk_surfaces_.remove(self.chunk_
    self.current_surface_chunk_index_ = (self.chunk_
        second_temp_chunk) + 1) % len(self.chunk_
else:
    temp_chunk = horizontal_chunks[selected_chunk_index_]
    self.chunks_.append(temp_chunk)
    self.ChangeAverageTime()
    if temp_chunk.subwidth_ == surface_chunk_width:
        self.chunk_surfaces_.remove(self.chunk_
            self.current_surface_chunk_index_ % len(self.chunk_
    else:
        self.chunk_surfaces_.insert(self.current_surface_
            surface_chunk_position_x, surface_chunk_

```

```

                surface_chunk_width - temp_chunk.subwidth
                self.chunk_surfaces_.remove(self.chunk_surfaces_
threshold += self.unit_time_
else:
    if is_better_split_on_2:
        split_on_2 = EnumerationSplitOn2(temp_chunk.subdata_matrix_,
                                         temp_chunk.position_y_)
        self.chunks_.append(Chunk(self.data_matrix_[self.height_ -
split_on_2[0][0] - self.height_ + split_on_2[0][1]:split_on_2[0][1] +
split_on_2[0][1]]))
        self.chunks_.append(Chunk(self.data_matrix_[self.height_ -
split_on_2[1][0] - self.height_ + split_on_2[1][1]:split_on_2[1][1] +
split_on_2[1][1]]))
        self.chunk_surfaces_.remove(self.chunk_surfaces_[i])
    if len(self.chunk_surfaces_):
        self.current_surface_chunk_index_ %= len(self.chunk_surfaces_)
        self.ChangeAverageTime()
elif is_better_split_on_3:
    split_on_3 = EnumerationSplitOn3(temp_chunk.subdata_matrix_,
                                     temp_chunk.position_y_, self.height_)
    self.chunks_.append(Chunk(self.data_matrix_[self.height_ -
split_on_3[0][0] - self.height_ + split_on_3[0][1]:split_on_3[0][1] +
split_on_3[0][1]]))
    self.chunks_.append(Chunk(self.data_matrix_[self.height_ -
split_on_3[1][0] - self.height_ + split_on_3[1][1]:split_on_3[1][1] +
split_on_3[1][1]]))
    self.chunks_.append(Chunk(self.data_matrix_[self.height_ -
split_on_3[2][0] - self.height_ + split_on_3[2][1]:split_on_3[2][1] +
split_on_3[2][1]]))
    self.chunk_surfaces_.remove(self.chunk_surfaces_[i])
    if len(self.chunk_surfaces_):
        self.current_surface_chunk_index_ %= len(self.chunk_surfaces_)
if self.processors_num_ != len(self.chunks_):
    self.ChangeAverageTime()
else:
    while not is_valid_chunk:
        limit_height = self.height_ - surface_chunk_position_x +

```

```

limit_width = surface_chunk_width + 1
chunks = []
distance_to_average_time = []
for i in range(1, limit_height):
    for j in range(1, limit_width):
        temp_chunk = Chunk(self.data_matrix_[
            self.height_ - surface_chunk_
            surface_chunk_position_y:surface_
            surface_chunk_position_x, surf
upper_undistributed_chunk = Chunk(
            self.data_matrix_[:self.height_ - surface_chunk_
            surface_chunk_position_y:surface_chunk_position_y,
            surface_chunk_position_x + i, surface_chunk_
right_undistributed_chunk = Chunk(self.data_matrix_[
            self.height_ - surface_chunk_
            surface_chunk_position_y:surface_
            surface_chunk_position_x, surf
        if abs(
            temp_chunk.subtotal_time_ - self.
            temp_chunk.subwidth_, temp_chunk.subheight_):
            if (
                upper_undistributed_chunk.subwidth_
                right_undistributed_chunk.subwidth_
                upper_undistributed_chunk.subheight_
                upper_undistributed_chunk.subwidth_
                chunks.append(temp_chunk)
                distance_to_average_time.append(
                    abs(temp_chunk.subtotal_time_ - self.
if len(chunks):
    temp_chunk = chunks[distance_to_average_time.index(m
    is_valid_chunk = True
if is_valid_chunk:
    self.chunks_.append(temp_chunk)
    self.ChangeAverageTime()
    self.chunk_surfaces_[self.current_surface_chunk_index_
        surface_chunk_position_x + temp_chunk.subheight_,
        temp_chunk.subwidth_)
    deleted_chunks_indices = []
if self.chunk_surfaces_[self.current_surface_chunk_index_
    deleted_chunks_indices.append(self.current_surface_
if self.current_surface_chunk_index_ > 0:
    if self.chunk_surfaces_[self.current_surface_chunk_
        self.chunk_surfaces_[self.current_surface_
            self.chunk_surfaces_[self.current_
            self.chunk_surfaces_[self.current_
                2] = \

```

```

                self.chunk_surfaces_[self.current_
                self.chunk_surfaces_[self.current_surface_chunk_
                self.chunk_surfaces_[self.current_surface_
                self.chunk_surfaces_[self.current_surface_
                self.chunk_surfaces_[self.current_surface_
                    2] + temp_chunk.subwidth_)]
                deleted_chunks_indices.append(self.current_
if self.current_surface_chunk_index_ < len(self.chunk_
    if temp_chunk.subwidth_ == surface_chunk_width:
        if self.chunk_surfaces_[self.current_surface_
            self.chunk_surfaces_[self.current_surface_
                self.chunk_surfaces_[self.current_
                    self.chunk_surfaces_[self.current_
                        2] == \
                self.chunk_surfaces_[self.current_
                self.chunk_surfaces_[self.current_surface_
                self.chunk_surfaces_[self.current_
                self.chunk_surfaces_[self.current_
                self.chunk_surfaces_[self.current_
                self.chunk_surfaces_[self.current_
                    deleted_chunks_indices.append(self.current_
second_temp_chunk = self.chunk_surfaces_[self.current_
if temp_chunk.subwidth_ < surface_chunk_width:
    self.chunk_surfaces_.insert(self.current_surface_
        surface_chunk_position_x, surface_chunk_position_y,
        surface_chunk_width - temp_chunk.subwidth_))
for k in range(len(deleted_chunks_indices)):
    self.chunk_surfaces_.remove(self.chunk_surfaces_
else:
    for k in range(len(deleted_chunks_indices)):
        self.chunk_surfaces_.remove(self.chunk_surfaces_
if self.current_surface_chunk_index_ in deleted_chunks_
    self.current_surface_chunk_index_ %= len(self.chunks_)
else:
    self.current_surface_chunk_index_ = (self.chunk_
        second_temp_chunk) + 1) % len(self.chunk_
threshold += self.unit_time_

def Split(self):
    self.ResetParameters()
    while len(self.chunks_) != self.processors_num_:
        self.PutChunk()

def FourDirectionSplit(self):
    split_results = []
    for i in range(4):
        self.data_matrix_ = self.data_matrix_.T

```

```

        self.data_matrix_ = np.array([self.data_matrix_[i, :][:: -1] for i in
        self.Split()
        split_results.append(copy.deepcopy(self.chunks_))
    return split_results

def IsBetterSplitExists(self):
    original_results = self.FourDirectionSplit()
    self.processors_num_ = self.processors_num_ - 1
    minus_1_processors_reuslts = self.FourDirectionSplit()
    self.processors_num_ = self.processors_num_ + 1
    orignal_results_time = []
    result_indices=[]
    for i in original_results:
        chunks_time = []
        for chunk in i:
            chunks_time.append(chunk.subtotal_time_)
        orignal_results_time.append(max(chunks_time))
    minimum_time = min(orignal_results_time)
    minus_1_procesors_results_time = []
    for i in minus_1_processors_reuslts:
        chunks_time = []
        for chunk in i:
            chunks_time.append(chunk.subtotal_time_)
        corrected_chunks_time = []
        if i[chunks_time.index(max(chunks_time))].subheight_*i[chunks_time.index(max(chunks_time))].subwidth_ < minimum_time:
            corrected_chunks_time.append(max(chunks_time))
            result_indices.append(minus_1_processors_reuslts.index(i))
        else:
            corrected_chunks_time.append(max(chunks_time) / 2)
            chunks_time.remove(max(chunks_time))
            corrected_chunks_time.append(max(chunks_time))
        minus_1_procesors_results_time.append(max(corrected_chunks_time))
    minus_1_processors_minimum_time = min(minus_1_procesors_results_time)
    if minus_1_processors_minimum_time + self.unit_time_ < minimum_time:
        if minus_1_processors_minimum_time in result_indices:
            return [True, minus_1_processors_reuslts[minus_1_procesors_results_time.index(minus_1_processors_minimum_time)]]
        else:
            minimum_index = minus_1_processors_results_time.index(minus_1_processors_minimum_time)
            return [True, minus_1_processors_reuslts[minimum_index], minimum_index]
    else:
        minimum_index = orignal_results_time.index(minimum_time)
        return [False, original_results[minimum_index], minimum_index]

def IsContiguous(self, chunk1, chunk2):
    if chunk1.position_y_ == chunk2.position_y_ and chunk1.subwidth_ == chunk2.subwidth_ and
       chunk1.position_x_ + chunk1.subheight_ == chunk2.position_x_ + chunk2.subheight_:
        if chunk1.position_x_ + chunk1.subheight_ == chunk2.position_x_:

```

```

        return (True, 'U') # U for up
    else:
        return (True, 'D') # D for down
elif chunk1.position_x_ == chunk2.position_x_ and chunk1.subheight_ == chunk2.subheight_
    and chunk1.position_y_ + chunk1.subwidth_ == chunk2.position_y_:
    if chunk1.position_y_ + chunk1.subwidth_ == chunk2.position_y_:
        return (True, 'R') # R for right
    else:
        return (True, 'L') # L for left
else:
    return (False, 'N') # N for none

def Adjustment(self, chunks):
    split_time = []
    for chunk in chunks:
        split_time.append(chunk.subtotal_time_)
    maximum_time_index = split_time.index(max(split_time))
    split_time[split_time.index(max(split_time))] = 0
    second_largest_time_index = split_time.index(max(split_time))
    contiguous_result = self.IsContiguous(chunks[maximum_time_index], chunks[second_largest_time_index])
    if chunks[second_largest_time_index].subtotal_time_ < chunks[maximum_time_index].subtotal_time_:
        contiguous_result[0]:
    if chunks[maximum_time_index].subheight_*chunks[maximum_time_index].subwidth_ > 1:
        split_on_2 = EnumerationSplitOn2(chunks[maximum_time_index].subdata_matrix_, chunks[maximum_time_index].position_x_, chunks[maximum_time_index].position_y_, chunks[maximum_time_index].subheight_, chunks[maximum_time_index].subwidth_)
        for i in range(2):
            chunks.append(Chunk(self.data_matrix_[self.height_ - split_on_2[i][0] - split_on_2[i][1]:split_on_2[i][1]+split_on_2[i][1]]))
    chunks.remove(chunks[maximum_time_index])
else:
    if contiguous_result[1] == 'U':
        enumeration_on_3_matrix = np.concatenate(
            (chunks[second_largest_time_index].subdata_matrix_, chunks[maximum_time_index].subdata_matrix_), axis=1)
        position_x = chunks[maximum_time_index].position_x_
        position_y = chunks[maximum_time_index].position_y_
    elif contiguous_result[1] == 'D':
        enumeration_on_3_matrix = np.concatenate(
            (chunks[maximum_time_index].subdata_matrix_, chunks[second_largest_time_index].subdata_matrix_), axis=1)
        position_x = chunks[second_largest_time_index].position_x_
        position_y = chunks[second_largest_time_index].position_y_
    elif contiguous_result[1] == 'R':
        enumeration_on_3_matrix = np.concatenate(
            (chunks[maximum_time_index].subdata_matrix_, chunks[second_largest_time_index].subdata_matrix_), axis=1)
        position_x = chunks[second_largest_time_index].position_x_
        position_y = chunks[second_largest_time_index].position_y_

```

```

position_x = chunks[maximum_time_index].position_x_
position_y = chunks[maximum_time_index].position_y_
else:
    enumeration_on_3_matrix = np.concatenate(
        (chunks[second_largest_time_index].subdata_matrix_, chunks[m
            axis=1)
    position_x = chunks[second_largest_time_index].position_x_
    position_y = chunks[second_largest_time_index].position_y_
    split_on_3 = EnumerationSplitOn3(enumeration_on_3_matrix, position_x,
    for i in range(3):
        chunks.append(Chunk(self.data_matrix_[
            self.height_ - split_on_3[i][0] - split_on_3[
                split_on_3[i][1]:split_on_3[i][1] + split_on_
                split_on_3[i][1]]))
    chunks.remove(chunks[maximum_time_index])
    chunks.remove(chunks[second_largest_time_index - 1 * (maximum_time_i
self.chunks_ = []
for chunk in chunks:
    self.chunks_.append(chunk)

def FindMinimum(self):
    is_better_split_exists_result = self.IsBetterSplitExists()
    self.chunks_ = []
    for chunk in is_better_split_exists_result[1]:
        if is_better_split_exists_result[2] == 3:
            self.chunks_.append(chunk)
        else:
            for i in range(is_better_split_exists_result[2] + 1):
                chunk.subdata_matrix_ = chunk.subdata_matrix_.T
                chunk.subdata_matrix_ = np.array(
                    [chunk.subdata_matrix_[i, :][::-1] for i in range(chunk.s
                if is_better_split_exists_result[2] == 0:
                    position_x = chunk.position_y_
                    position_y = self.width_ - (chunk.position_x_ + chunk.subheight_
                elif is_better_split_exists_result[2] == 1:
                    position_x = self.height_ - chunk.position_x_ - chunk.subheight_
                    position_y = self.width_ - chunk.position_y_ - chunk.subwidth_
                else:
                    position_x = self.height_ - chunk.position_y_ - chunk.subwidth_
                    position_y = chunk.position_x_
                self.chunks_.append(Chunk(chunk.subdata_matrix_, position_x, position_y))
    if is_better_split_exists_result[0]==True and is_better_split_exists_resu
        chunks = copy.deepcopy(self.chunks_)
        self.Ajustment(chunks)

def DisplayResult(self):
    split_time = []

```

```
for chunk in self.chunks_:
    split_time.append(Chunk(
        self.final_data_matrix_[self.height_- chunk.position_x_- chunk.subheight_:self.height_
        chunk.position_y_:chunk.position_y_ + chunk.subwidth_], chunk.pos_
        chunk.position_y_).subtotal_time_)
print(split_time)

if __name__ == '__main__':
    data = GetDataMatrix(kTestData, 'fishy_cat', 4, chunks_edge)
    split = Image(data[0], data[1], processors_num)
    split.FindMinimum()
    split.DisplayResult()
```

Appendix C

Related Works

Graphics rendering can be used in scientific visualization, CAD, vehicle simulation, animation etc [5]. Due to its demanding requirements, instead of relying on expensive high-end graphics machines, some engineers have turned to parallel rendering system which has a higher performance-to-price ratio [6]. Molnar et al. [5] propose a taxonomy for the parallel rendering system by dividing them into three classes, sort-first, sort-middle and sort-last depending on the timing when primitives are distributed to rendering pipelines comparing with the timing of geometry processing and rasterization. Molnar's analysis is based on the rasterization while our results are obtained by using the parallel rendering method of ray tracing. Besides, our work focuses on the partition on "coarse granularity", which means we focus on the distribution of rendering works to different processors instead of the work partitioning within a single processor for its different threads. Our work concerns two domains : load estimation and rectilinear partitioning on two dimensions. The previous works are presented as follows.

As for load estimation, our system is to some extent similar to the sort-first system of rasterization which divides the image at first and distributes the regions resulted from the division to the processors and usually suffers the load imbalance problem [5]. In order to reduce load imbalance of sort-first rasterization and of other parallel rendering methods, one can estimate the cost of rendering the image and calculate a beneficial distribution in order to reduce total running time. Goil et al. [8] present a method by using the distribution of scanlines of previous frame to estimate scanline distribution for the current frame. However, this method is limited by the change of viewpoint which may lead to serious load imbalance. Reinhard et al. [7] predict the cost of rendering each voxel by first estimating the time for rendering a ray based on its geometrical complexity and then estimating the number of primary and secondary rays generated by each voxel by using the number of pixels the voxel projects onto. The problem of this method is that the estimation is not exact enough because of some improper hypotheses and incapacity in exactly estimating the number of rays per pixel. Wimmer et al. [9] propose a framework for estimating the rendering time for the rasterization method by dividing the rendering process into a number of independent tasks et estimating the time separately. Three concrete methods are proposed but none of them give satisfying results in all circumstances and for a more satisfying result, some hardware improvements are required. Gillibrand et al. [10] present a method which is called "profiling" and to which our method of cost estimation is similar. They sample a small number of pixel regions to estimate the time for the final high resolution image but this method suffers the inaccuracy incurred by the sparse sampling. Gillibrand et al. [11] develop the previous method by exploiting the scene geometry and surface properties to better choose the position of the sampling pixels in order to make better cost prediction. Their method faces serious inaccuracy with the use of caching. Cosenza et al. [12] propose a method of per-pixel cost estimation by the information

available in G-buffer generated by deferred shading techniques. This method produces a good approximation but compared to our method, it has estimation error of more than 10% for some pixels while our method keeps the relative error under 10% in the experiments.

In terms of rectilinear partitioning, another problem to solve in order to achieve load balance, previous works can be divided into three classes : partitioning on one dimension, on two dimensions and on three dimensions.

Partitioning on one dimension is the problem to partition a chain of unevenly distributed works and distribute them to processors so that the running time of the most heavily loaded processor is minimized. For this problem, exact algorithms are considered inefficient and not affordable and many heuristics have been proposed [13]. Pinar et al. [13] divide these heuristics into three categories : dynamic programming, iterative refinement and parametric search. Olstad et al. [14] propose a dynamic programming algorithm by defining the problem as a dynamic programming problem which has the complexity of $O(p(n-p))$, which is the best result of dynamic programming algorithms for this problem that we are aware of. Manne et al. [15] present a method falling into the category of iterative refinement by finding a sequence of nonoptimal partition that can be improved in only one way and improving the partition iteratively. The complexity of this method is $O(p(n-p)\log p)$. The parametric search algorithms are to search the index of the partition according to the ideal partition. Miguet et al. [16] propose a heuristic based on finding the index of the workload when the cumulative workload first exceeds the cumulative workload of the ideal partition, i.e., the most equal partition and an improvement of this heuristic. Another technique called probing is proposed and this technique is to find a partition with its maximum load no more than a given probe value b . The best result for the method of probing that we are aware of is of complexity $O(n + p^{1+\epsilon})$ for any small $\epsilon > 0$ given by Han et al. [17]. In addition, Khanna et al. [18] present a heuristic which is the best result of heuristics of this problem according to our research of previous work with running time $O(n\log n)$ for arbitrary p . Besides these heuristics, Pinar et al. [13] propose an exact algorithm for this problem which increases the bottleneck value gradually and which shows a much better balance performance than heuristics and with proposed efficient implementations, this method can replace heuristics efficiently.

Computing the optimal rectilinear partition on two dimensions is an NP-Hard problem [19]. To measure the quality of heuristics, load imbalance is proposed which is calculated as $\frac{L_{max}}{L_{avg}} - 1$ with L_{max} representing the maximum load and L_{avg} the average load and also the ideal load [20]. Some heuristics of this problem use the results of this problem on one dimension. One example is the heuristic presented by Nicol [21]. This heuristic based on the idea of iterative refinement finds the optimal partition in one dimension when the partition in the other dimension is fixed and uses the probing method to find the optimal partition in one dimension. Then the partition found is fixed, the heuristic searches for the optimal partition of the other dimension. This procedure is repeated until there is no change in the result. Manne et al. [22] propose a heuristic which extends the dynamic programming method on one dimension to find the semi-generalized block distribution on two dimensions. Some methods are distinctive on two dimensions. Mueller [23] presents a method named MAHD which uses the hierarchical technique of median-cut if the number of processors is the power of 2 or appropriate split ratio otherwise. A 4-approximation (i.e., load imbalance equals 4) method is given by Gaur et al. [24] by introducing a decision version of the original problem and using an algorithm to some extent similar to probing and some results for solving rectangle stabbing problem. Improvements for these algorithms have been proposed. Muthukrishnan et al. [25] give an algorithm for the initial configuration of algorithms proposed in

[21, 22]. Finally, Saule et al. [20] show that their algorithms which fall in the categories of m-way jagged partition and hierarchical bipartition respectively have better load balance performance than the state-of-art algorithms.

The rectilinear partitioning problem on three dimensions is NP-complete proved by Nicol [21]. Some heuristics in two dimensions can be generalized in three dimensions. The algorithms proposed by Gaur et al. [24] and Muthukrishnan et al. [25] can be extended to three dimensions. Marchesin et al. [26] propose a method based on kd-tree which splits a plane which is orthogonal to one of the base axes alternatively according to the load estimation.

We point out here that different from most systems in load balancing, the cost of communication among different processors is not a worry of our system because our system executes the compositing step at last for different parts of the image. Therefore, we will not consider the reduction of communication cost in the rest of our paper as in most papers concerning the problem of load balancing.

Our paper is organized in the following way. In the next section we present our method of load estimation and the reason for using this method. Then in Section 4, we introduce our algorithm Tetris for the rectilinear partitioning problem on two dimensions. In Section 5 we present the experiments and the results of our load estimation method and algorithm. Finally, in Section 6 we conclude our work and point out what remains to be done in the future.