UNIVERSITÉ DE MONTRÉAL

PRÉDICTION DE PERFORMANCE DE MATÉRIEL GRAPHIQUE DANS UN CONTEXTE AVIONIQUE PAR APPRENTISSAGE AUTOMATIQUE

SIMON RIVARD-GIRARD

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

AOÛT 2015

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

PRÉDICTION DE PERFORMANCE DE MATÉRIEL GRAPHIQUE DANS UN CONTEXTE AVIONIQUE PAR APPRENTISSAGE AUTOMATIQUE

présenté par : RIVARD-GIRARD Simon

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. BELTRAME Giovanni, Ph. D. , président

M. BOIS Guy, Ph. D. , membre et directeur de recherche

M. DAGENAIS Michel, Ph. D. , membre

DÉDICACE

« No great genius has ever existed without some touch of madness »

- Aristote

REMERCIEMENTS

Mon directeur de recherche, M. Guy Bois, m'a gracieusement invité à prendre part à son projet de recherche malgré mon ignorance en rapport aux systèmes embarqués en avionique. Malgré nos différences d'expertises, j'espère que ma contribution en vision par ordinateur enrichira son projet et les relations qu'il entretient avec nos entreprises partenaires. Grâce à lui, il m'a été possible d'obtenir l'expérience d'un stage Mitacs Accélération qui a pu me guider vers un sujet de recherche d'actualité. Grâce à lui, il m'a aussi été possible de jouir sans casse-tête d'une aide financière tout à fait agréable. À vous, je dis merci! Grâce à vous, je suis maintenant un passionné des circuits numériques.

Bien que remise par l'intermédiaire de mon directeur de recherche, mon aide financière provient de certains organismes qui contribuent grandement à la recherche scientifique au Canada. Je tiens à remercier le *Centre de recherche en sciences naturelles et en génie du Canada (CRSNG)*, l'organisme à but non-lucratif *Mitacs*, ainsi que le *Consortium en aérospatial pour la recherche et l'innovation au Canada (CRIAQ).* Grâce à vous, il m'a été possible de répondre à mes besoins primaires et d'avoir pu mener à succès toute cette merveilleuse expérience qu'est ma maîtrise.

Une grande inspiration en lien au choix de mon sujet de recherche provient de l'entreprise partenaire à mon stage *Mitacs*. Il s'agit d'un grand meneur dans le domaine de la fabrication et la vente de simulateurs de vols. Je tiens à remercier CAE, et en particulier mon superviseur de stage M. Jean-Pierre Rousseau, qui a pu me permettre de choisir un sujet de recherche dont les répercussions sont directement applicables à un problème réel.

J'aimerais aussi remercier mon collègue et ami Bao Lin, qui m'a fait parvenir du thé de Chine durant toutes mes études de maîtrise. Nos discussions de politique et d'histoire chinoise, ainsi que nos rêves de projets futurs ont pu carburer nos nuits blanches dans le laboratoire.

Enfin, j'aimerais remercier mes parents, mon amoureuse et mes ami(e)s pour leur support moral, affectif et psychologique. Il arrive qu'on néglige l'importance de ce type de support, mais je suis maintenant certain que sans vous je n'aurais pas pu mener à terme cette aventure.

RÉSUMÉ

Le matériel informatique graphique destiné aux ordinateurs de bureau ou aux systèmes embarqués traditionnels, ainsi que leur interface de programmation ne peuvent pas être utilisés dans les systèmes avioniques puisqu'ils ne se conforment pas aux règles de certifications DO-254 et DO-178B. Toutefois, on remarque le faible nombre d'outils de conceptions qui encadrent le développement d'applications graphiques avioniques, et ce malgré l'apparition de matériel graphique avionique de plus en plus performants. Suivant par exemple la méthode classique de conception en V, les ingénieurs doivent d'abord effectuer des choix de conception reliés à la sélection du matériel graphique avant de débuter une quelconque implémentation de code. Ainsi, il peut être difficile d'évaluer la pertinence de ces choix en évaluant les performances de traitement du matériel graphique puisque l'engin graphique n'aurait pas nécessairement été développé. Je propose donc un outil de conception permettant de prédire les performances de matériel graphique en termes d'images par secondes (FPS), basé sur *OpenGL SC*. L'outil crée des modèles non-paramétriques de performance du matériel en analysant, à l'aide d'algorithmes d'apprentissage, le temps de dessin de chaque image, lors du rendu d'une scène 3D synthétique. Cette scène est rendue à quelques reprises en faisant varié certaines de ses caractéristiques spatiales (nombre de sommets, taille de la scène, taille des textures, etc.) qui font parte intégrante du logiciel de vision synthétique habituellement développé dans ce domaine. Le nombre de combinaisons de ces caractéristiques utilisées durant l'entraînement supervisé des modèles de performance n'est qu'un très petit sous-ensemble de toutes les combinaisons, le but étant de prédire par extrapolation celles manquantes. Pour valider les modèles, une scène 3D fournie par un partenaire industriel est dessinée avec des caractéristiques non traitées durant la phase d'entraînement, puis le FPS de chaque image rendue est comparé au FPS prédit par le modèle. La tendance centrale de l'erreur de prédiction est ensuite démontrée comme étant moins de 4 FPS.

ABSTRACT

Within the strongly regulated avionics engineering field, conventional graphical desktop hardware and software API cannot be used because they do not conform to the DO-254 and DO-178B certifications. We observe the need for better avionic graphical hardware, but system engineers lack system design tools related to graphical hardware. The endorsement of an optimal hardware architecture by estimating the performance of a graphical software, when a stable rendering engine does not yet exist, represents a major challenge. There is also a high potential for development cost reduction, by enabling developers to have a first estimation of the performance of its graphical engine at a low cost. In this paper, we propose to replace expensive development platforms by a predictive software running on desktop. More precisely, we present a system design tool that helps predict the rendering performance of graphical hardware based on the OpenGL SC API. First, we create non-parametric models of the underlying hardware, with machine learning, by analyzing the instantaneous frames-per-second (FPS) of the rendering of a synthetic 3D scene and by drawing multiple times with various characteristics that are typically found in synthetic vision applications. The number of characteristic combinations used during this supervised training phase is a subset of all possible combinations, but performance predictions can be arbitrarily extrapolated. To validate our models, we render an industrial scene with characteristics combinations not used during the training phase and we compare the predictions to real values. We find a median prediction error of less than 4 FPS.

TABLE DES MATIÈRES

DÉDICACE III

REMERCIEMENTS IV

RÉSUMÉ V

ABSTRACT VI

TABLE DES MATIÈRES VII

LISTE DES TABLEAUX X

LISTE DES FIGURES XI

LISTE DES SIGLES ET ABBRÉVIATIONS XIII

CHAPITRE 1 INTRODUCTION 1

1.1 Problématique 2

1.2 Objectifs 5

1.3 Méthodologie 6

1.4 Contribution 7

1.5 Organisation du mémoire 8

CHAPITRE 2 REVUE DE LA LITTÉRATURE 9

2.1 Outils d'évaluation de performance graphique 9

2.1.1 *Basemark® ES* 9

2.1.2 *SPECviewperf* 9

2.1.3 Outil d'évaluation de performance du projet *AREXIMAS* 10

2.2 Méthode de prédiction de performance de matériel informatique 16

2.2.1 Outils d'analyse de performance disponibles sur le marché 16

2.2.2 Prédiction par modélisation analytique et simulation 17

2.2.3 Prédiction de performance par modélisation paramétrique 18

2.2.4 Prédiction de performance par apprentissage automatique 19
2.2.5 Détermination du seuil acceptable d'erreur de prédiction 26
2.2.6 Prédiction du temps de transfert entre la mémoire centrale et graphique 27
2.3 Simulation de l'environnement *OpenGL SC* 27
CHAPITRE 3 PRÉSENTATION DE L'ARTICLE 29
CHAPITRE 4 ARTICLE 1 : AVIONICS GRAPHICS HARDWARE PERFORMANCE PREDICTION WITH MACHINE LEARNING 30
4.1 Abstract 30
4.2 Introduction 30
4.3 Background 32
4.4 Avionics Graphic Hardware Performance Benchmarking 34
4.4.1 Synthetic Scene Generation 36
4.4.2 Camera Movement Pattern 38
4.4.3 Loading Data to the Graphic Memory 39
4.4.4 Analysis of the Percentage of Scene Drawn 39
4.4.5 Rendering Performance Metrics 40
4.5 Avionics Graphic Hardware Performance Modeling 40
4.5.1 Machine Learning Algorithms Configuration 41
4.5.2 Performance Data Smoothing 42
4.5.3 Quantifying Scene Characteristics Equivalency 43
4.5.4 Identifying the Percentage of Space Parameters Evaluated 45
4.6 Prediction Error Evaluation 47
4.6.1 Experimental Setup 47
4.6.2 Performance Model Validation 47
4.6.3 Metric Choice and Interpretation 49

4.7 Results 49
4.8 Discussion 52
4.9 Conclusion 53
4.10 Acknowledments 53
4.11 References 54
CHAPITRE 5 RÉSULTATS COMPLÉMENTAIRES 56
5.1 Vers une évaluation de performance plus exacte 56
5.1.1 Calcul du nombre de sommets tenant compte de l'élimination des faces arrières 56
5.1.2 Réduction du taux moyen d'échec de cache 57
5.1.3 Utilisation du IFPS sans moyenne arithmétique 57
5.2 Évaluation quantitative des améliorations 58
5.3 Discussion partielle reliée à l'OS2 62
CHAPITRE 6 DISCUSSION GÉNÉRALE 63
CHAPITRE 7 CONCLUSION ET RECOMMANDATIONS 65
RÉFÉRENCES 66

LISTE DES TABLEAUX

Tableau 2-1: Pourcentage d'erreur moyen de prédiction et écart type d'erreur de prédiction de performance dans la littérature.....27

Table 4-1: Values used for the tile resolution grid size tests.....41

Table 4-2: Parameters for the machine learning algorithms used.....42

Table 4-3: Values of the World CDB scene characteristics.....48

Table 4-4: Central tendencies of the prediction error distributions for each machine learning algorithm and for each validation dataset. 50

Tableau 5-1: Mesures de tendances centrales des erreurs de prédiction pour les tests d'évaluation de performance de la carte Nvidia QuadroFX570 – Comparaison entre les méthodes avec et sans améliorations pour chaque scènes de validation et pour chaque algorithme d'apprentissage.....59

LISTE DES FIGURES

Figure 1-1: Système de vision synthétique, par *Honneywell.* 2

Figure 1-2: Cycle de développement en V. 3

Figure 2-1: Scène synthétique générée par l'outil d'évaluation de performance du projet AREXIMAS. 10

Figure 2-2: Démonstration du nombre de points et de surfaces effectivement affichés après l'élimination des faces arrière et sortantes du tronc de projection, ainsi que du z-test. 13

Figure 2-3: Exemple d'arbre de classification *CART* 21

Figure 2-4: Réseau neuronal acyclique avec un seul niveau de nœuds internes. 23

Figure 4-1: Dataflow of the proposed tool in an experimental context. 34

Figure 4-2: Pyramid vertices generated with a c-by-c dimension top-facing (top-left) and front-facing (top-right). Pyramid rendered mesh without added noise (bottom-left) and with added noise (bottom-right). 36

Figure 4-3: Various intensity of noise depending on the height of the pyramid. Low noise amplitude (left) to high noise amplitude (right). 37

Figure 4-4: Overall generated synthetic scene with pyramids height varying according to their position in the grid. 37

Figure 4-5: Comparison between smoothed (left) and unsmoothed (right) performance data. 43

Figure 4-6: Mesh and normals of one tile of the validation scene sampled at a resolution of 9x9 (left), 19x19 (middle) and 31x31 (right) shown before applying the randomized texture. 48

Figure 4-7: Prediction error distribution of the validation dataset #1 for the artificial neural network. 51

Figure 4-8: Prediction error distribution of the validation dataset #2 for the artificial neural network 51

Figure 4-9: Prediction error distribution of the validation dataset #3 for the artificial neural network. 52

Figure 4-10: Prediction error distribution of the validation dataset #4 for the artificial neural network. 52

Figure 5-1: Distribution des IFPS pour l'ensemble de données d'entraînement – **avec** contribution de l'OS2.....58

Figure 5-2: Distribution des IFPS pour l'ensemble de données d'entraînement – **sans** contribution de l'OS2.....58

Figure 5-3: Distribution d'erreur de prédiction pour la scène de validation #1 pour les modèles entraînés par réseaux neuronaux **avec** la contribution de l'OS2. 60

Figure 5-4: Distribution d'erreur de prédiction pour la scène de validation #1 pour les modèles entraînés par réseaux neuronaux **sans** la contribution de l'OS2. 60

Figure 5-5: Distribution d'erreur de prédiction pour la scène de validation #2 pour les modèles entraînés par réseaux neuronaux **avec** la contribution de l'OS2. 60

Figure 5-6: Distribution d'erreur de prédiction pour la scène de validation #2 pour les modèles entraînés par réseaux neuronaux **sans** la contribution de l'OS2. 60

Figure 5-7: Distribution d'erreur de prédiction pour la scène de validation #3 pour les modèles entraînés par réseaux neuronaux **avec** la contribution de l'OS2. 61

Figure 5-8: Distribution d'erreur de prédiction pour la scène de validation #3 pour les modèles entraînés par réseaux neuronaux **sans** la contribution de l'OS2. 61

Figure 5-9 : Distribution d'erreur de prédiction pour la scène de validation #4 pour les modèles entraînés par réseaux neuronaux **avec** la contribution de l'OS2. 61

Figure 5-10 : Distribution d'erreur de prédiction pour la scène de validation #4 pour les modèles entraînés par réseaux neuronaux **sans** la contribution de l'OS2. 61

LISTE DES SIGLES ET ABRÉVIATIONS

ASIC	Circuit intégré propre à une application (Application-Specific Integrated Circuits)
CART	Arbre de classification et de régression (Classification and Regression Trees)
CRIAQ	Consortium en aérospatial pour la recherche et l'innovation au Canada
FPS	Images par seconde (Frame Per Second)
H#	Hypothèse numéro #
IFPS	Images par seconde instantanée (Instantaneous Frame Per Second)
LiDAR	Senseur de données 3D dont l'abréviation vient de la fusion du mot “light” et “radar”
MART	Arbres de régression multiple et accumulatif (Multiple Additive Regression Trees)
MSE	Écart-quadratique moyen (*Mean Squared Error*)
OG#	Objectif général numéro #
OS#	Objectif spécifique numéro #
RMSE	Écart-type de l’erreur (Rooted Mean Squared Error)
RTCA	Commission de radiotransmission pour l’aéronautique (Radio Technical Commission for Aeronautics)

CHAPITRE 1 INTRODUCTION

Le projet CRIAQ AVIO509 a pour but d'explorer de nouvelles méthodes de conception pour systèmes avioniques modulaires intégrés (IMA). L'étude de l'impact des décisions architecturales peut mener vers une réduction des coûts de développements déjà exorbitants de tels systèmes. La conception de systèmes logiciels/matériels pour de telles plates-formes s'avère une tâche beaucoup plus complexe que dans un cadre normal. Étant des systèmes critiques dont les fautes peuvent mettre en danger la vie humaine, plusieurs règles de sécurité et de certification viennent influencer leur développement. Ce qui différentie le développement de tels systèmes d'un développement dans un cadre conventionnel sont ces normes qui régissent à la fois le développement logiciel (DO-178B) et matériel (DO-254) [1]. Toutefois, leur étude approfondie dépasse le cadre de ce travail. Ainsi, la plupart des logiciels et du matériel conventionnels non-avionique ne peuvent pas être employés directement dans ces systèmes dû à des contraintes de certification.

D'autre part, il existe un besoin grandissant dans l'industrie aérospatiale de développer des applications graphiques, que ce soit sous forme de logiciel de vision synthétique, de cartes interactives ou de réalité augmenté, pour n'en nommer que quelques-uns. Les technologies graphiques actuelles pour ordinateur de bureau – jeux vidéo, conception assistée par ordinateur, etc. – seraient plus que suffisantes pour répondre aux besoins en haute performance de ces applications, mais elles ne sont hélas pas certifiables et ne peuvent être utilisées directement dans ces systèmes. L'industrie avionique utilise donc diverses alternatives : notamment à l'aide de processeurs tout-usage certifiables ou de vieux modèles d'architectures de processeurs graphiques, plus faciles à certifier, mais ayant un écart technologique de plus de dix ans avec la technologie actuelle. Toutefois, la demande de performance toujours plus haute, par des applications de plus en plus gourmande, alimente donc la recherche de matériel graphique certifiable [2, 3].

Parmis les applications graphiques les plus développées dans le domaine, on retrouve les systèmes de vision synthétique, qui permettent de reproduire artificiellement et en temps-réel l'environnement spatial autour du véhicule. Ce genre d'outil d'aide à la navigation possède plusieurs utilisations dont l'aide à l'atterrissage dans des conditions visibilité nulle, l'évitement de câbles à haute-tension, etc. Ce travail s'intéresse particulièrement à l'analyse des outils de conception offerts dans la littérature et à la comparaison entre la disponibilité de ces outils pour le

développement graphique conventionnel non-avionique et leur disponibilité dans le cadre avionique. De surface, on remarque qu'une grande majorité de ces outils de conception ne peuvent être utilisés dans un contexte avionique et qu'il existe donc un besoin de les adapter aux contraintes du domaine.

Figure 1-1: Système de vision synthétique, par *Honneywell*.
Source: http://en.wikipedia.org/wiki/File:Synthetic_Vision.JPG.
License: cc-by-sa 3.0 http://creativecommons.org/licenses/by-sa/3.0/

1.1 Problématique

Dans un contexte de développement conventionnel, plusieurs outils sont offerts aux développeurs d'applications graphiques : boîte à outils pour interfaces graphiques, évaluateurs de performances ou *benchmarks* d'applications au niveau du transfert de données de la mémoire vive vers la carte graphique et/ou au niveau du temps d'affichage, prédicteurs de performances des applications par modélisation des caractéristiques internes du matériel graphique, etc.

Du côté avionique, plusieurs outils commerciaux sont déjà disponible aux développeurs d'applications graphiques avionique : notamment des outils d'aide à la conception d'interface graphique conformant au standard ARINC-661 [4, 5]. Il y a donc peu d'intérêt de recherche dans ce genre d'outil. Par contre, il existe très peu d'outils d'évaluation des performances [6] et il n'existe aucun outil de prédiction de performance. L'intérêt d'adapter les *benchmarks* et les outils de prédictions de performance au domaine de l'avionique est donc intéressant.

L'intérêt d'utiliser de tels outils d'évaluation de performance en ingénierie vient du fait qu'au moment de choisir le matériel graphique pour le système, l'engin graphique logiciel n'a pas

nécessairement été développé. Une équipe de développement a donc intérêt à pouvoir évaluer les performances du matériel sans avoir à attendre après le développement du logiciel graphique. La figure 1.2 montre que la conception architecturale se fait bien avant le codage pour le cycle de développement en V.

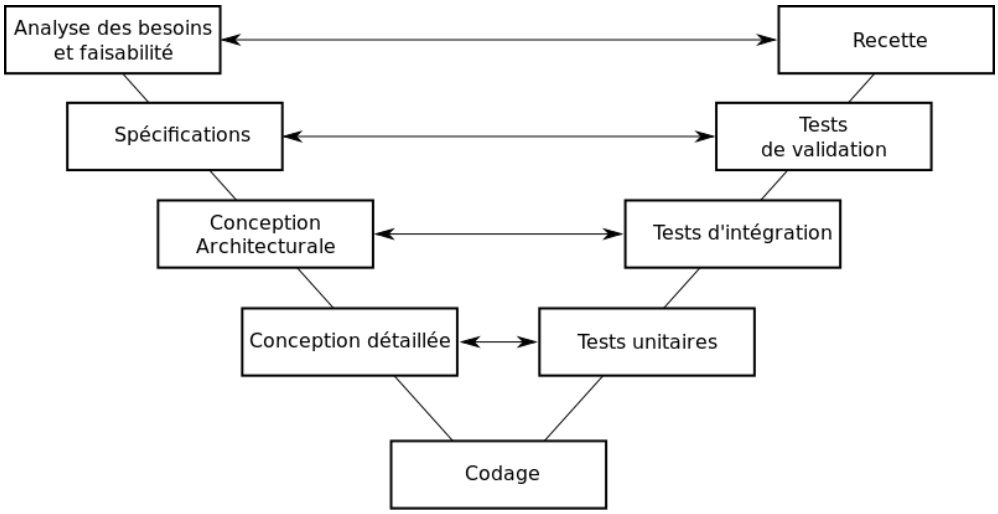

Figure 1-2: Cycle de développement en V.
Source: http://commons.wikimedia.org/wiki/File:Cycle_de_developpement_en_v.svg
Licence: CC-BY-SA 3.0 http://creativecommons.org/licenses/by-sa/3.0/

Les outils de prédiction de performance vont encore plus loin que la simple évaluation en pire cas effectuée par les outils d'évaluation de performance : ils permettent de prédire avec un certain seuil de certitude le comportement du matériel en fonction de ses entrées et/ou de ses caractéristiques internes. On peut donc interpoler ou extrapoler les résultats obtenus par un outil d'évaluation de performance en créant un modèle prédictif du comportement du matériel. Une équipe de développement qui connaît déjà la quantité et la taille des données que devra afficher l'application graphique dans son contexte d'exécution (pas nécessairement le pire cas) peut donc comparer divers matériel avec une plus grande pertinence. Plusieurs problèmes sont néanmoins rencontrés lors de la tentative d'adapter ces outils d'un cadre standard vers un cadre avionique.

Il y a d'abord l'utilisation d'une interface de programmation spécifique pour communiquer avec le matériel graphique, afin de faciliter le respect des normes de certification. Dans un cadre avionique, cette interface est *OpenGL SC*, basé sur la populaire interface *OpenGL*, et consiste en un sous-ensemble de fonctions imitant les fonctionnalités offertes par la version 1.3 de l'API. Les outils de conception pour matériel conventionnel utilisent plutôt les plus récentes versions (3.x ou 4.x) de la

pleine interface de programmation [7], ce qui les rend plus ou moins utilisables dans un contexte avionique.

De plus, les normes de certification imposent l'utilisation d'un pipeline graphique fixe, signifiant que le programme interne exécuté par un processeur graphique ou un circuit numérique ne peut être modifié par les développeurs de l'application graphique. Ces programmes, aussi appelés *shaders,* sont utilisés conventionnellement pour créer des effets graphiques de haute qualité ou pour le traitement parallèle *(GPGPU).* Les outils de conception standards cherchent donc plutôt à pousser les limites de ces programmes ou à évaluer quels effets visuels un matériel d'accélération graphique peut supporter, ce qui n'est pas nécessairement recherché dans un cadre avionique [8].

D'autre part, on dénote une grande diversité de matériel utilisé ou dont l'utilisation est prévue dans le futur pour exécuter des fonctionnalités graphiques. Dans la littérature, on retrouve des processeurs graphiques standards ou personnalisés, des processeurs tout-usage avec une unité graphique intégrée, des processeurs graphiques logiciels, des processeurs graphiques, des ASIC, ou des processeurs graphiques implémentés sur un FPGA [2]. Toutefois, plusieurs des types de matériel précédemment cité ne sont que des prédictions ou des intérêts de recherche envers leur utilisation. Pour l'instant, l'industrie semble avoir adopté l'utilisation de processeur graphique implémentés en logiciel sur un processeur tout-usage, car il s'agirait de la façon la plus simple de répondre aux critères de certification de matériel DO-254. La majorité des outils de conception standards vont être plutôt développés pour évaluer des processeurs graphiques en utilisant des métriques telles que le nombre d'instructions machine dans les *shaders* (plus de détails dans la revue de littérature). Ainsi, ils ne pourraient être utilisés par exemple pour évaluer un circuit sur FPGA. De plus, la majorité des cartes disponibles commercialement sont des boîtes noires très sécurisées dont il est impossible d'accéder légalement aux composantes internes. Il faut donc que l'outil de conception qui évalue ou prédit les performances d'applications graphiques avioniques soit généralisable pour tous ces types de matériel en n'utilisant que leur interface.

La question que l'on se pose alors : est-il possible d'adapter les techniques de prédiction de performances d'un cadre standard vers un cadre avionique tout en tenant compte des problèmes mentionnés précédemment?

1.2 Objectifs

L'objectif général #1 de l'activité de recherche (OG1) est d'identifier les techniques de prédiction de performance de matériel graphique qui ont un potentiel à s'appliquer au domaine de l'avionique, tout en respectant les problèmes énumérés précédemment (avec ou sans adaptation). On veut ainsi prouver qu'il est possible d'adapter des outils de conception permettant la prédiction des performances du matériel graphique dans un cadre avionique. Pour ce faire, un outil de prédiction de performance est donc développé. Cet outil se doit d'être réutilisable et portable en ne dépendant pas d'un processeur en particulier ou d'un matériel graphique en particulier. Cet outil génère un modèle prédictif des performances du matériel graphique à partir des données obtenues en évaluant ses performances. L'adaptation d'outils d'évaluation de performance sera potentiellement nécessaire afin de permettre des prédictions efficaces subséquemment. Les modèles prédictifs servent donc à interpoler ou à extrapoler les données obtenues durant cette évaluation. L'outil développé ne peut toutefois pas être utilisé afin d'évaluer si un certain matériel ou logiciel graphique est certifiable ou non selon les divers standards du RTCA.

L'objectif spécifique #1 (OS1) est de comparer le pouvoir prédictif de diverses méthodes de régression statistiques permettant de modéliser les performances d'une application graphique en fonction de ses entrées. La comparaison de ces méthodes doit se faire de façon quantitative et statistiquement significative. Le choix d'une métrique permettant de comparer le pouvoir prédictif fait donc parti de la tâche à faire.

L'objectif spécifique #2 (OS2) est de comparer et/ou d'adapter diverses méthodes permettant de récolter les performances de matériel graphique dans un cadre avionique à partir de ses entrées. Le choix de la métrique de performance et des paramètres d'entrées du matériel graphique fait donc parti de la tâche à faire. Pour se faire, il serait intéressant d'apporter des améliorations en termes de précision à l'outil d'évaluation de performance développé dans des travaux de recherches antérieurs reliés au projet CRIAQ-AVIO509.

L'objectif spécifique #3 (OS3) est de trouver une méthode permettant d'éviter l'achat de matériel graphique avionique, tout en permettant d'utiliser *OpenGL SC* avec des cartes graphiques conventionnelles dans l'optique où l'acquisition de matériel graphique avionique dépasse les limites du budget et/ou de temps de ce projet de recherche.

Une hypothèse est donc émise (H1) : la prédiction de performance de matériel graphique en avionique est considérée comme significativement utilisable si l'erreur de prédiction est similaire à celle des autres méthodes dans la littérature appliquée à tout domaine confondu. La quantification de l'erreur de prédiction « acceptable » est détaillée dans la revue de la littérature au chapitre 2.

1.3 Méthodologie

L'activité de recherche est divisée en trois phases : phase de recherche, phase d'implémentation et phase d'analyse. La phase de recherche consiste principalement à faire la revue de la littérature pour évaluer l'intérêt de la problématique et les diverses solutions déjà proposées. Cette phase s'est réalisée en six étapes :

1. Parcourir la littérature pour comparer les caractéristiques des outils d'évaluation de performance de matériel graphique dans un contexte standard avec celles des mêmes outils dans un contexte avionique.
2. Parcourir la littérature pour recenser et comparer les diverses méthodes utilisées pour prédire les performances de matériel graphique dans un contexte standard. Puis, évaluer le potentiel d'adaptation de chacune à un contexte avionique.
3. Parcourir la littérature pour recenser des méthodes de simulation d'environnement d'*OpenGL SC* dans un cadre non-avionique.
4. Obtenir l'outil d'évaluation des performances de matériel graphique avionique développé par notre laboratoire, évaluer les améliorations à y apporter et obtenir l'autorisation de son utilisation par son auteur original.
5. Dans l'optique de comparer les prédictions faites par le modèle avec les vrais valeurs de performances, il faut trouver une scène à afficher qui diffère de celle utilisée lors de l'entraînement du modèle. L'obtention de scènes 3D utilisées par l'industrie dans un contexte commercial se fait par l'entremise de nos partenaires industriels.
6. Établir les divers paramètres variables que l'on appliquera au matériel afin d'en évaluer l'impact sur ses performances. Établir la métrique utilisée pour représenter la performance du matériel.

La phase d'implémentation consiste à l'activité de programmation reliée aux divers objectifs de l'activité de recherche. Elle se fait en trois étapes :

7. Implémenter les améliorations à faire à l'outil d'évaluation de performances obtenues à l'étape 3. De plus, l'outil génère initialement une scène synthétique et il faut donc lui permettre de récolter les performances des scènes 3D « réelles » utilisées dans l'industrie.
8. Une fois ces modifications apportées on peut procéder à la récolte de données de performance pour les deux types de scènes.
9. Bâtir divers modèles de performance selon ceux trouvés à l'étape 2 en utilisant les données de performance de la scène synthétique générée par l'outil amélioré à l'étape 7.

La dernière phase consiste à analyser les résultats obtenus et s'est faite en une unique étape:

10. Comparer et analyser le pouvoir prédictif des modèles de performance en comparant la différence entre la performance prédite et la performance véritable recueillis lors de l'affichage de la scène « réelle » utilisée en industrie trouvée à l'étape 4.

1.4 Contribution

La réalisation de l'OG1 entraîne la preuve que les techniques de prédiction de performance de matériel graphique sont adaptables au domaine de l'avionique, ce qui ne semble pas avoir été fait auparavant selon la littérature. Nous emmenons plus loin le concept de la simple évaluation de performance, en démontrant qu'il est possible de faire de la prédiction malgré les problèmes propres à ce domaine mentionnés au chapitre 1.1. Ainsi, un outil de prédiction de performance est développé utilisant plusieurs types de modèle de performance entraînés à l'aide de régressions non-paramétriques. Le pouvoir prédictif de ces divers types de régressions est récolté dans le but de confirmer ou d'infirmer l'hypothèse H1. Dans le cas où l'intérêt industriel est plutôt de faire une simple évaluation de performance sans prédiction, la seconde contribution est l'amélioration d'un outil d'évaluation de performance pouvant être utilisé dans un cadre avionique.

1.5 Organisation du mémoire

Ce mémoire est présenté « par article » et diverge donc du format standard. Le chapitre 2 présente une revue critique de la littérature. Le chapitre 3 introduit l'article et le situe par rapport aux objectifs de recherche. Le chapitre 4 est l'article en soi, présentant les divers aspects des algorithmes utilisés, les résultats et leur analyse. Le chapitre 5 présente des résultats complémentaires, principalement ceux reliés à l'OS2. Le chapitre 6 présente la discussion générale des résultats obtenus permettant d'inférer des conclusions par rapport aux objectifs et à l'hypothèse. Finalement le chapitre 7 présente la synthèse et les travaux futurs.

CHAPITRE 2 REVUE DE LA LITTÉRATURE

La revue de la littérature est divisée en trois parties. Les caractéristiques des outils d'évaluation de performance de matériel graphique dans un contexte standard sont d'abord comparées avec celles des mêmes outils dans un contexte avionique. De plus, l'outil d'évaluation de performance de matériel graphique avionique développé par notre laboratoire est évalué pour en trouver les défauts à régler afin d'être utilisé à des fins de prédiction de performance. Ensuite, les diverses techniques de prédiction de performance de matériel informatique sont recensés. Enfin, les méthodes de simulation d'environnement *OpenGL SC* sont évaluées afin d'éviter l'achat de matériel graphique avionique.

2.1 Outils d'évaluation de performance graphique

La recherche d'outils d'évaluation de performance est nécessaire puisque tout outil de prédiction de performance, ou plus abstraitement n'importe quelle régression statistique, nécessite d'abord un échantillon duquel un modèle peut être généré.

2.1.1 *Basemark® ES*

Ce banc de test est parmi les plus populaires dans le cadre d'évaluation de performance d'applications graphiques embarquées (c.f. [9]). Selon la description du produit offerte par le site web de la compagnie qui le développe, l'outil nécessite l'utilisation de la plus récente version d'*OpenGL ES* et récolte des données de performance en mettant la carte sous le stress de modèles lumineux et d'ombrages complexes, de rendu sur cibles multiples et autres fonctionnalités avancées, dépassant largement l'intérêt d'applications graphiques avioniques. La métrique utilisée pour mesurer les performances semble propriétaire et peu documentée. L'outil ne semble pas fournir de méthode de prédiction de performance. Ainsi, cet outil ne semble pas convenir pour permettre la génération d'un modèle de prédiction.

2.1.2 *SPECviewperf*

Selon l'information fournie sur leur site web [10], cet outil utilise *OpenGL* et permet de sélectionner une liste de tests à effectuer en chaîne, chacun agissant d'une façon différente sur le matériel graphique. La métrique mesurant la performance est la moyenne du nombre d'images

générées par seconde (*FPS* moyen) durant le rendu des diverses scènes offertes par l'outil. Plusieurs types de tests sont disponibles, incluant : la modification de la majorité des états d'*OpenGL*, l'utilisation ou non de textures, l'utilisation de filtres de textures, l'élimination ou non des faces avant ou arrière (front-face et back-face culling), le rendu à double ou simple tampon, l'utilisation de modèle lumineux simple ou complexe, etc. Hormis l'utilisation d'*OpenGL*, ce sont toutes des caractéristiques intéressantes dans le cadre d'évaluation de performances pour application graphique avionique. Toutefois, l'utilisation du *FPS* moyen semble posséder peu de significativité statistique puisque la distribution de FPS n'est pas normale. En plus, dans le cadre de l'OG1, l'utilisation de cette métrique ne permettrait pas d'entraîner des prédicteurs de performance robustes pour cette même raison. Enfin, l'outil n'offre pas de fonctionnalités de prédiction de performance.

2.1.3 Outil d'évaluation de performance du projet *AREXIMAS*

Initialement développé par V. Legault [6], cet outil génère un banc de test à la façon de SPECViewPerf, mais en considérant spécialement les besoins du domaine de l'avionique, soit : utilisation d'*OpenGL ES 1.0* afin d'émuler les fonctionnalités offertes par *OpenGL SC*, utilisation des fonctionnalités de base de l'API utilisant un pipeline fixe et abstraction du matériel sous-jacent à l'API. L'outil génère une scène synthétique à paramètres variables : nombre de points, taille de la scène, taille des textures, taille du tampon de sortie (ou taille de l'écran), etc. dont la nature spatiale est très représentative d'une scène typique utilisée par des systèmes de vision synthétiques dans le domaine de l'avionique.

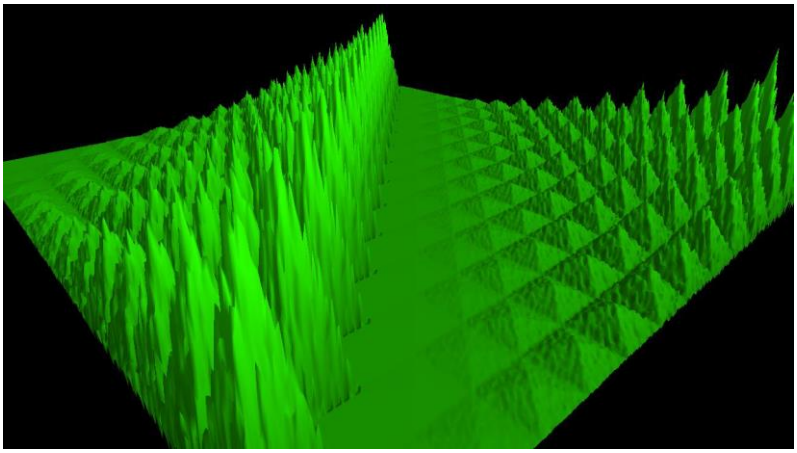

Figure 2-1: Scène synthétique générée par l'outil d'évaluation de performance du projet AREXIMAS. Auteur original: Vincent Legault. Source : [6]. Utilisé avec permission de l'auteur.

L'auteur suggère l'utilisation de scènes composées de plusieurs tuiles, chacune contenant un nombre égal de sommets, puisque cela mimique le comportement généralement réalisé par des applications graphiques en avionique. Il se base d'ailleurs sur une étude de cas venant de l'industrie et affirme au meilleur de ses connaissances qu'il s'agit d'un cas d'utilisation très repandu.

Un test est défini pour chaque valeur de paramètre puis, pour chaque test, une caméra parcourt toujours de la même façon la scène synthétique. Le parcours de la caméra est défini de sorte à afficher divers pourcentages de la scène totale afin d'évaluer l'influence sur le temps requis pour générer l'image. Durant ce parcours, pour chaque image générée, l'inverse du temps requis pour afficher cette image ou *FPS* instantané (*IFPS)* est récolté en fonction du pourcentage affiché de la scène. Par exemple, pour un test utilisant une scène contenant 100 000 points, le rapport généré subséquemment contiendra un tableau contenant l'*IFPS* de chaque image généré en fonction du pourcentage des 100 000 points affichés dans chaque image. Le pourcentage de points affichés est calculé comme suit $\frac{nombre\_de\_points\_affichés}{nombres\_de\_points\_total}$ où le $nombre\_de\_points\_affichés$ représente le nombre de points contenus dans le parallépipède du tronc de projection de la caméra et $nombres\_de\_points\_total$ représente la quantité de points totale dans la scène définie par la valeur de ce paramètre du test actuellement effectué. L'auteur original moyenne ensuite cet *IFPS* pour chaque valeur de paramètre testé (par exemple, 100 000, 200 000, 300 000, etc. points) afin de générer des graphes représentant le *FPS* moyen en fonction de la valeur des paramètres. Aucune prédiction de paramètre n'est effectuée par la suite. Dans le cadre de l'OG1 du présent travail de recherche, cet outil est d'un grand intérêt : toutes les considérations propres au domaine de l'avionique sont prises en compte, puis la première métrique soit l'*IFPS* en fonction du pourcentage affiché de la scène représente un échantillon très intéressant pour générer un modèle prédictif de performance. Plus de détails sur cet outil sont offerts dans l'article présenté au chapitre 4, toutefois l'article présente l'outil amélioré dont les lacunes suivantes ont été corrigées.

2.1.3.1 Ignore l'élimination des faces arrières et du z-test durant le calcul du nombre de points affichés

Le nombre de points affichés ne dépend pas uniquement du nombre de points contenus dans le parallépipède du tronc de projection de la caméra. Deux autres facteurs l'influencent, soit : l'élimination des faces arrières, l'élimination des faces occluses. Un autre facteur influence les performances en plus du nombre de sommets : le *z-test*. D'ailleurs, l'élimination des points sortant

du parallépipède du tronc de projection ne fait pas partie du standard d'*OpenGL*, mais cette méthode est tellement avantageuse que la majorité des pilotes de matériel graphique l'implémentent en-dessous du capot. L'élimination des faces arrières est une partie du programme du pipeline fixe défini par le standard d'*OpenGL*. Cette étape vérifie pour chaque point si la surface composée de ce point fait face ou non à la caméra. Si la surface ne fait pas face à la caméra, le point est subséquemment ignoré par le reste du pipeline. Il s'agit d'une méthode efficace pour ne pas traiter des points qui ne seront pas visibles selon la position actuelle de la caméra, mais qui sont tout de même dans le tronc de projection de cette dernière. *OpenGL* utilise l'ordre dans lequel les sommets d'un triangle sont parcourus pour définir si une surface fait face ou non à la caméra. Plusieurs autres techniques sont présentes dans la littérature [11, 12], toutefois la plus simple permettant de recréer celle utilisée par OpenGL consiste à analyser la différence d'angle entre la direction de la caméra et le vecteur normal de chaque surface.

L'élimination des faces occluses ne fait pas partie du standard d'*OpenGL* et n'est pas implémentée par les pilotes. Cette méthode doit être implémentée au niveau applicatif afin de réduire le nombre de données envoyées vers la carte. Ainsi, dans le cadre de l'étude des performances de matériel graphique, son utilisation n'est pas nécessaire puisqu'il suffit de modifier le nombre de données envoyées de façon manuelle. À titre informatif, la littérature présente quelques méthodes, principalement les *Potentially Visible Set*, les *Portals*, et les *Antiportals* (c.f. [13-16]).

Le *z-buffering* est performé vers la fin du pipeline graphique, bien après l'élimination des faces-arrières et est effectué dans le cadre de la génération de l'image en sortie. Étant donnée la nature tridimensionnelle des scènes, il est possible que plusieurs surfaces d'objets de la scène se « cachent » l'une-l'autre, malgré le fait que les surfaces des deux objets font face à la caméra et sont positionnées dans le tronc de projection de cette dernière. Lors de la génération de l'image, chaque surface d'objet est divisée en diverses petites unités de taille similaire à un pixel nommées fragments. D'ailleurs, les opérations *par-fragment* (telles l'ajout de lumière ou d'ombrage ou encore l'application de texture sur des surfaces) sont très coûteuse. Ainsi, si un fragment est caché par un autre fragment, il est optimal de ne pas effectuer d'opération *par-fragment* sur celui-ci. Pour chaque pixel de l'image générée on effectue l'opération *par-fragment* uniquement pour ceux représentant la surface en avant-plan (admettant que la surface n'est pas semi-transparente). Le standard d'*OpenGL* demande l'utilisation d'un tampon de profondeur qui permet d'associer une profondeur *z* pour chaque pixel de l'image de sortie. Si une surface demande l'exécution

d'opération *par-fragment*, le tampon de profondeur est interrogé pour vérifier s'il n'y a pas déjà eu une valeur de couleur associée à ce fragment pour une coordonnée moins profonde dans la scène. Si c'est le cas, alors l'opération est ignorée, sinon elle est exécutée et le tampon de profondeur est mis-à-jour avec la profondeur de la surface pour chaque pixel touché. Pour pouvoir recréer ce test de profondeur à l'extérieur du contexte d'OpenGL, il faut avoir la possibilité de demander à la mémoire graphique d'envoyer le tampon de profondeur vers la mémoire RAM applicative ou encore de reproduire une approximation externe du tampon de profondeur. Cette fonctionnalité n'est toutefois pas présente dans la version *Safety-Critical* d'OpenGL. Encore une fois, la littérature propose quelques autres méthodes pour effectuer ou optimiser cette technique d'élimination de données [17-19].

La figure 2-2 illustre les concepts présentés précédemment :

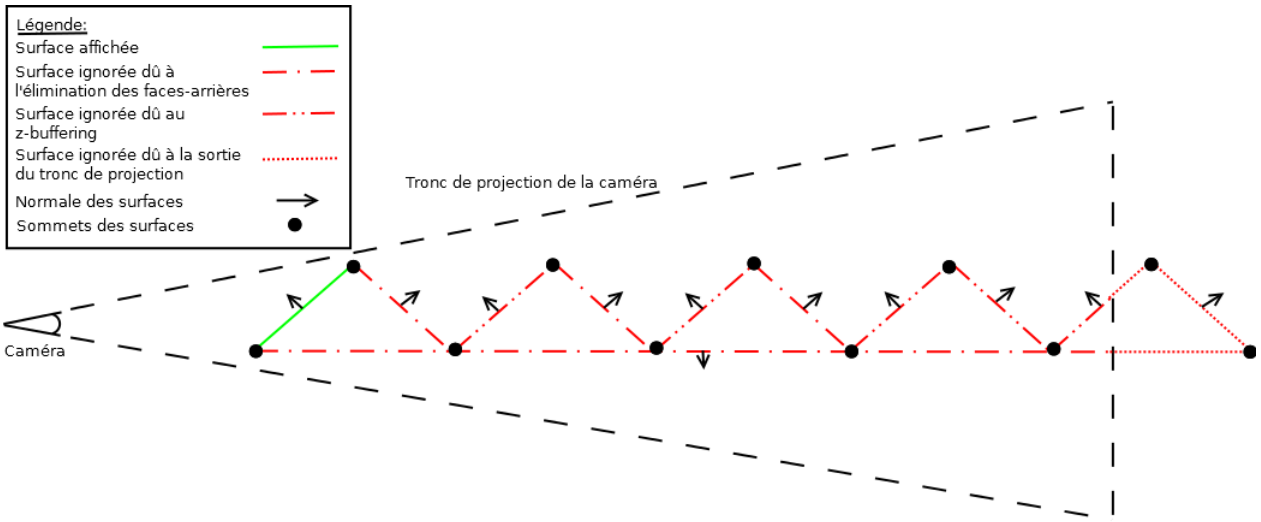

Figure 2-2: Démonstration du nombre de points et de surfaces effectivement affichés après l'élimination des faces arrière et sortantes du tronc de projection, ainsi que du z-test.

On remarque donc que des onze sommets présents dans la scène, seulement deux seront utilisés par le matériel graphique lors du rendu de cette scène, tandis que l'outil d'évaluation en aurait calculé neuf.

2.1.3.2 Haut taux moyen d'échec de cache

La majorité des pipelines de matériel graphique utilisent de la mémoire cache pour garder en mémoire les opérations récentes effectuées sur des sommets. Le matériel est donc sujet à des

baisses de performance reliées à l'absence de l'optimisation de cette antémémoire. Il est établi depuis déjà plus d'une dizaine d'année que le type de cache le plus optimal pour ce matériel est le FIFO [20]. Ainsi, en réorganisant l'ordre des données 3D avant de les envoyer vers la mémoire graphique, il est possible de réduire le taux moyen d'échec de cache permettant du coup d'améliorer les performances générales de l'application graphique. Ce présent outil d'évaluation de performance ne fait pas exception à cette règle et souffre donc de ce problème, puisqu'il ne prend pas soin d'effectuer ce prétraitement sur ses données. Les données 3D sont en général les sommets de la scène ainsi que le tableau d'index contenant tous les ensembles de trois sommets représentant tous les triangles composant les surfaces de la scène. L'optimisation consiste habituellement à réorganiser le tableau d'index de sommets afin de traiter le plus grand nombre de faces utilisant un même sommet avant de le décharger de la mémoire cache pour éviter d'avoir à faire repasser ce même sommet dans le pipeline plus tard. Depuis la preuve de la meilleure performance des caches FIFO, plusieurs auteurs ont développé des algorithmes permettant de réorganiser les données 3D en en tenant compte.

Ce problème initialement abordé dans [20] est résolus à l'aide de deux techniques : la technique *Greedy Strip-Growing* ordonne les données de façon avare en parcourant la chaîne d'index de triangle initial et en en reconstituant une nouvelle plus optimale. À chaque incrément durant l'itération du tableau d'index initial, il décide s'il est mieux d'ajouter la donnée à la nouvelle chaîne ou d'en recommencer une nouvelle. Pour ce faire, il procède similairement aux algorithmes de parcours de graphes en attribuant à chaque somme un indicateur *parcouru* ou *non-parcouru*. Il démarre la nouvelle chaîne avec l'index d'un sommet qui possède un petit nombre de sommets voisins, puis il parcourt de voisin en voisin chaque sommet en les indiquant comme étant *parcouru*. Si après une itération il n'existe plus de points voisins et que tous les points n'ont pas été parcourus, alors il recommence une nouvelle chaîne. L'autre technique présentée dans l'article est la technique *Local Optimization* qui tente d'améliorer la chaîne extraite par l'algorithme avare à l'aide de trois perturbations pouvant être effectuées à chaque paire de trois sommets (triangle) *x* et *y* de la chaîne : le reflet (inversion d'une sous-séquence de donnée), l'insertion1 (ajout d'une donnée) ou l'insertion2 (ajout de deux données). Ces perturbations sont appliquées si la variation d'une certaine métrique de coût calculée avant et après la perturbation est plus petite que zéro. La sélection de la surface *x* est aléatoire, puis on sélectionne une surface *y* dont les sommets sont

adjacents à ceux de *x* soit dans la scène ou soit dans l'ordonnancement trouvés par l'algorithme avare.

Plus récemment, [21] brevète une technique basée sur la création de multiples groupes de données 3D. Ces groupes sont formés à l'aide d'une adaptation du célèbre algorithme *k-mean* permettant de générer des groupes de données en utilisant l'information de planéité des diverses surfaces (vecteurs normals). Un graphe de connectivité entre les divers groupes est ensuite généré à l'aide de leur méthode utilisant l'information d'occlusion qu'un groupe a sur un autre selon diverses positions de la caméra dans la scène. Les groupes sont par la suite ordonnés dans un vecteur à l'aide d'un tri topologique si le graphe de connectivité est acyclique, ou de l'heuristique *Minimum Feedback Arc Set* si le graphe est cyclique, pour estimer le tri optimal de ce problème NP-complet.

Dans [22], la solution se présente en effectuant plusieurs petits rendus partiels de la scène. Chaque sommet se fait attribuer un indicateur *blanc* (le sommet n'a pas encore été utilisé), *gris* (le sommet est en cours d'utilisation) ou *noir* (le sommet a été utilisé et ne sera plus nécessaire). Chaque sommet est initialement indiqué comme étant *blanc* et l'algorithme s'arrête quand tous les sommets sont indiqués comme étant *noir*. À chaque itération de l'algorithme un point *blanc* ou *gris* est sélectionné comme celui possédant le focus, puis chaque surface utilisant ce sommet est dessiné. Le sommet est ensuite indiqué comme étant *noir* puis une nouvelle itération débute.

La méthode *Cache-Oblivious Layouts* est présentée dans [23] comme une solution au problème d'optimisation de leur métrique *Cache-Oblivious* pouvant être mesurée à partir de la géométrie de la scène. Ils démontrent un lien étroit entre la minimisation du taux moyen d'échec de la cache et la minimisation de leur métrique. Le processus NP-difficile de minimisation se fait à l'aide d'une heuristique effectuant plusieurs minimisations récursives sur divers niveaux de densité de données.

Enfin, une des solutions les plus récentes est proposée dans [24]. L'algorithme nommé *tipsify* est selon l'auteur le premier algorithme résolvant ce problème en temps linéaire O(n) en fonction du nombre de données mises en entrée. Simplement résumé, un tableau d'adjacence point-triangle est d'abord créé, puis l'index d'un point 3D est sélectionné. Chaque triangle utilisant ce point est parcouru de façon aléatoire et les indices des points les constituant sont transférés vers le tableau de sortie. Ensuite, un point voisin à ce point initial, contenant des triangles non-parcourus, est sélectionné à partir du tableau d'adjacence et l'opération se répète. Si l'algorithme de parcours rencontre une impasse, c'est-à-dire qu'il n'y a plus de points voisins faisant part de triangles non-

parcourus, alors un point non-voisin contenant des triangles non-parcourus est choisi aléatoirement dans le tableau d'adjacence. Ainsi, l'algorithme se complète après avoir itéré qu'une seule fois le tableau d'index de points fournis en entrée.

2.1.3.3 Utilisation d'une moyenne arithmétique pour le calcul du FPS

Comme mentionné précédemment, l'auteur original moyenne les *IFPS* récoltés pour chaque valeur de paramètre testé (par exemple, 100 000, 200 000, 300 000, etc. points) afin de générer des graphes représentant le *FPS* moyen en fonction de la valeur de divers paramètres. Le problème avec cette technique est qu'elle n'est statistiquement pas significative, car la distribution des IFPS n'est pas normale. De plus, les IFPS récoltés contiennent beaucoup de données aberrantes. Cette moyenne est donc peu représentative de la tendance de performance que l'auteur tente de représenter. Les preuves de non-respect de la normalité et de présence de données aberrantes sont présentées dans l'article du chapitre 4, ainsi que dans le chapitre 5.

2.2 Méthode de prédiction de performance de matériel informatique

Les techniques de prédictions de performance ne sont pas restreintes qu'au matériel ou aux applications graphiques. Dans la littérature, on recense l'utilisation de ces méthodes pour la conception de microarchitecture, de processeurs reconfigurables et pour des systèmes parallèles, pour n'en nommer que quelques usages. Cette section énumère donc diverses méthodes utilisées pour la prédiction de performance de plusieurs types de matériel ou logiciel informatique pour lesquelles il existe un intérêt notable à être adapté à la prédiction de matériel graphique.

2.2.1 Outils d'analyse de performance disponibles sur le marché

Les grands joueurs du marché de matériel graphique ont tendance à offrir des outils d'analyse de performance propres à leur matériel. Il y a entre autres l'outil Nsight [25] pour les cartes graphiques Nvidia et l'outil GPU PerfStudio [26] pour les cartes graphiques AMD. Bien que très au point, ces outils sont mal adaptés au domaine de l'avionique, car ils ne prennent pas en compte les divers aspects de la problématique propre à l'avionique tels l'utilisation d'*OpenGL SC* ou d'un pipeline fixe. De plus, ces outils n'existent que pour du matériel graphique utilisant des processeurs graphiques. Il est toutefois intéressant de poursuivre la revue de la littérature en étudiant les diverses méthodes qui pourrait être utilisées par ces outils.

2.2.2 Prédiction par modélisation analytique et simulation

Plusieurs techniques de prédiction de performance se basent sur la création d'un modèle mathématique du matériel graphique. Les bornes de temps d'exécutions du matériel sont approximées à l'aide de fonctions. L'exemple le plus simple est présenté dans [27] : la performance d'un processeur graphique est une fonction du nombre de noyaux dans le processeur (*CC*), de la fréquence d'horloge du processeur (*F*), du nombres d'instructions par cycle (*IPC*) et du nombre d'instruction dans le programme (*IC*). $Performance = \frac{CC}{\alpha} * F * \frac{IPC}{IC}$ où α est un facteur de mise à l'échelle. Il utilise sa métrique pour récolter quelques valeurs, puis en crée un modèle à l'aide d'une fonction linéaire. Dans [28], le modèle analytique est élargi en incluant une analyse de la mémoire partagée et locale à chaque thread exécuté par le processeur. Puis [29], pousse beaucoup plus loin le modèle analytique du processeur graphique en y incluant plus d'une vingtaine de métriques qui utilisent des détails reliés à la mémoire et à l'organisation des threads. Il porte d'ailleurs une grande attention à la modélisation du parallélisme au niveau de la mémoire et des échecs de cache. Son modèle est utilisé surtout pour la simulation de cartes graphiques, mais il pourrait être possible de l'utiliser pour faire de la prédiction de performance de matériel graphique en général. L'article [30] reprend une technique similaire en créant trois modèles distincts de la carte graphique, toutefois ils modélisent trois flots de données typiques à l'utilisation de ce matériel au lieu de l'architecture du circuit numérique. Ces dernières méthodes ne prennent pas en compte les délais provenant du pipeline interne du processeur dans leurs modèles, tels les délais de pipeline lors d'instructions SIMD, les conflits reliés aux instructions de lecture/écriture mémoire ou encore les flots de contrôle divergeant. Les auteurs de [31] proposent donc une méthode basée sur la modélisation du flot de contrôle à l'aide d'un graphe cyclique directionnel pour tenir compte de ces facteurs. Plusieurs autres outils ont été développés afin de faire l'analyse des programmes envoyés vers les processeurs graphiques et d'en extraire des métriques : soit en transformant le code source vers un programme C exécutable par un processeur multicoeur tout-usage [32] ou encore à l'aide de simulateurs exécutant sur un processeur tout-usage le code compilé destiné aux processeurs graphiques [33-35]. D'autre part, [36] utilise l'analyse par composantes principales et l'analyse par composantes indépendantes pour évaluer la relation entre les performances et les composantes internes d'un processeur tout-usage. Un dernier auteur effectue une méta-analyse de plusieurs outils d'évaluation de performance d'applications graphiques et tente d'extraire un

modèle en corrélant les données de chacun [37]. Lors des prédictions de performance, il utilise les données fournit par le sous-ensemble d'outils d'évaluation de performance qui représente le mieux l'application graphique étudiée.

Les modèles analytiques sont aussi utilisés dans le cadre de prédiction de performance pour applications parallèles et distribuées. D'abord, [38] utilise les caractéristiques de l'équipement informatique et des caractéristiques de l'application pour créer son modèle. Dans [39], l'analyse des instructions du processeur au niveau des opérations de mémoire et des opérations arithmétiques est utilisée pour la création d'un modèle mathématique. Dans [40], le modèle de performance est créé sur une plate-forme de référence en exécutant itérativement un programme de façon partielle par incrément d'une certaine durée de temps et en mettant à jour un modèle à chaque itération. Le modèle est ensuite validé sur une autre plate-forme.

Le problème majeur avec ces techniques est qu'elles nécessitent l'accès au programme exécuté dans la carte graphique. Toutefois, ceci n'est pas possible dans un contexte avionique étant donné que le pipeline est fixe, c'est-à-dire que le programme s'exécutant sur la carte graphique ne peut être fourni par le développeur de l'application graphique étant donné qu'il est intégré dans la boîte noire par le fournisseur du matériel et qu'il est donc immuable et légalement inaccessible. De plus, la littérature semble indiquer que ce genre de modélisation est difficile à réaliser, car il est difficile de bien identifier tous les facteurs influençant les performances.

2.2.3 Prédiction de performance par modélisation paramétrique

Les modèles paramétriques sont réalisés habituellement en trouvant les coefficients permettant de plaquer un modèle mathématique (droite, plan, etc.) sur un ensemble de données en minimisant la distance moyenne entre chaque point et le modèle. Plusieurs méthodes peuvent être utilisées telles les moindres carrées ou RANSAC, quoique leur démonstration sort du cadre de ce travail. La revue de la littérature prend d'abord ici une vision plus large en recensant les méthodes de modélisation de performance de matériel informatique en général. Il est intéressant de remarquer la présence d'utilisation de régression paramétrique pour modéliser les performances de microarchitectures [41, 42] et de processeurs tout-usage [43] dans le cadre d'exploration d'espace de conception. D'autre part, la prédiction de performance au niveau applicatif est réalisée dans [44] en performant une analyse d'abord statique du code, puis en faisant une analyse dynamique pendant

l'exécution du programme afin d'en extraire des données permettant de générer un modèle par régression paramétrique. Une autre technique consiste en la génération de plusieurs modèles de régression pour diverses étapes d'exécution d'un programme [45].

Par contre, ces méthodes sont contraintes par les lacunes de ce type de régression : difficulté d'expliquer la variance pour des modèles complexes, fortes influence négative des données aberrantes et non-linéarité probable des modèles qui sont tentés d'être expliqués. D'ailleurs, ces méthodes paramétriques sont difficiles à appliquer à la prédiction de performance de matériel graphique dans un cadre avionique, car l'information utilisée pour créer le modèle (entrées/sorties uniquement) n'explique que partiellement les performances obtenues, de sorte qu'un grand volume de données aberrantes est présent.

2.2.4 Prédiction de performance par apprentissage automatique

D'autres méthodes de modélisation des performances utilisent des algorithmes d'apprentissage supervisé provenant du domaine de l'intelligence artificielle. Il existe plusieurs familles algorithmiques dans ce domaine permettant la création de modèle de régression dits non-paramétriques, c'est-à-dire que la fonction permettant de prédire la performance (FPS) selon divers prédicteurs (nombres de points, etc.) est inobservable et qu'il faut donc l'approximer. Dans le cadre de la prédiction de performance à partir de données provenant d'un outil comme celui découlant du projet AREXIMAS, il faut choisir la ou les familles permettant d'effectuer de la fouille de données en prenant en compte une grande présence de données aberrantes : le volume de données envoyé ou sortant du matériel graphique est un prédicteur plus faible que des prédicteurs représentant les détails architecturaux internes du matériel, tel que présenté dans les méthodes décrites précédemment. Ainsi, deux familles algorithmiques semblent être utilisées couramment dans la littérature : les arbres de régressions (arbre de régression simple, forêt d'arbres décisionnels ou *boosting* d'arbres de régressions aussi appellée *MART Multiple Additive Regression Trees*) et les réseaux neuronaux artificiels sans récursion (*Feedforward*). Avant de poursuivre, il est important de différencier les algorithmes d'apprentissage supervisé et non-supervisé. L'apprentissage supervisé permet de bâtir une base de règles permettant d'expliquer la relation inconnue entre une ou des variable(s) dite d'entrée X_i et une variable de sortie Y. Cela nécessite donc qu'il y ait une base de donnée d'entraînement possédant plusieurs observations associant plusieurs valeurs de X avec celle Y correspondante. Il s'agit donc d'une méthode intéressante pour

de l'étude régressive ou de la classification. D'autre part, l'apprentissage non-supervisé cherche à trouver des relations cachées permettant de regrouper les divers X_i de la base de données d'apprentissage sans qu'il n'y ait nécessairement de Y observés ou disponibles. Cette technique permet de créer des grappes de données afin d'en extraire des caractéristiques communes. Il est intéressant de noter l'existence de l'apprentissage par renforcement pour la modélisation de systèmes d'états-transitions-actions à l'aide d'une méthode de récompense/punition, mais cela dépasse la portée de ce travail. Pour plus de détails sur ces concepts, l'aide-mémoire disponible dans Wikipédia présente bien la base de ces domaines de recherche distinct : [46].

2.2.4.1 Prédiction de performance par arbres de régressions

Les arbres de régressions simples, connus sous le nom *CART (Classification and regression tree)* sont des modèles prédictifs en forme d'arbre permettant de modéliser des fonctions non-linéaires, c'est-à-dire pour lesquels on ne peut trouver de formule mathématique expliquant la relation entre son entrée et sa sortie. Étant un domaine de rechercher distinct en soi, un aide-mémoire présentant la littérature de base reliée aux *CART* est disponible dans [47]. Il existe deux natures à ces arbres selon qu'ils retournent une valeur discrète (arbre de classification) ou une valeur continue (arbre de régression). Ces arbres sont créés ou entraînés à l'aide de certains algorithmes d'apprentissage supervisé dont l'explication dépasse le cadre de ce travail. Le problème principal avec les arbres de décision simples est qu'ils peuvent être parcourus de sorte qu'un branchement est emprunté menant à un extremum local de la fonction non-linéaire inconnue qui est modélisée, ce qui l'empêcherait d'atteindre les branches contenant l'extremum global recherché. De plus, une fois entraîné, chaque niveau de profondeur de l'arbre est interprété comme une question booléenne (exemple : âge > 9.5?), puis chaque branche représente une réponse à cette question. Pour obtenir une prédiction, on fournit une valeur du domaine en entrée, puis on parcourt l'arbre en suivant les branches répondant à cette valeur. Pour entraîner l'arbre, il faut fournir un vecteur contenant diverses données pour lesquelles on cherche à expliquer la valeur de sortie. Par exemple, dans le cas de la prédiction de performance de matériel graphique, on cherche à expliquer le *FPS* selon le nombre de points dans la scène, la taille de la scène, la taille de l'écran, etc. La figure 2-3 présente un autre exemple provenant de wikipédia démontrant la relation entre divers facteurs démographiques des passagers du Titanic et la classe « Survivant » vs « Mort ».

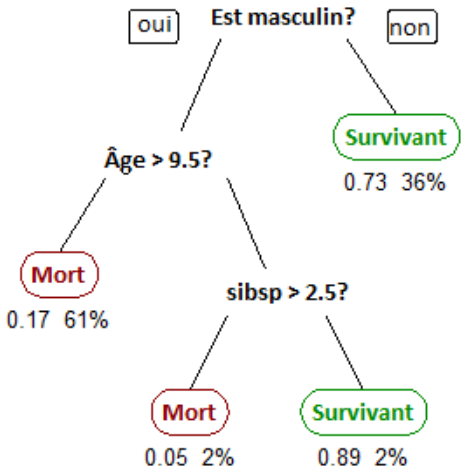

Figure 2-3: Exemple d'arbre de classification *CART*.
Auteur : Stephen Milborrow. Traduis de l'anglais par Simon Rivard-Girard.
Source: http://commons.wikimedia.org/wiki/File:CART_tree_titanic_survivors.png
Licence: CC-BY-SA 3.0 http://creativecommons.org/licenses/by-sa/3.0/

D'autre part, les forêts d'arbres décisionnels sont des ensembles d'arbres simples créées à partir du même ensemble de donnée initial toutefois en appliquant la méthode d'aggrégation par *bootstrap*, aussi connue sous le nom de *bagging* qui permettent d'augmenter la précision des prédictions. Le *bagging* consiste à échantillonner aléatoirement à plusieurs reprises un sous-ensemble des données d'entraînement pour par la suite effectuer un même traitement statistique sur chacun. Dans notre cas, l'ensemble de donnée utilisé pour l'entraînement de l'arbre est un vecteur dont chaque entité possède la forme :

$$[valeur, [prédicteurs]] \ = \ [FPS, [nombre\ de\ points, taille\ de\ scène, \dots]]$$

La méthode d'agrégation par *bootstrap* sélectionne donc aléatoirement plusieurs sous-ensembles de données de ce vecteur. Par la suite, la forêt d'arbre décisionnelle est créée en entraînant un arbre de décision simple pour chacun des sous-ensembles. Pour effectuer une prédiction avec la forêt d'arbre, on effectue d'abord une prédiction à partir de chacun des arbres simples, puis la prédiction finale est calculée, entre autres, à l'aide de la moyenne arithmétique de l'ensemble de prédictions individuelles. Cette méthode permet d'affaiblir le problème des arbres simples quant au fait de rester coincer dans un extremum local, étant donné que cet extremum local n'est pas partagé par tous les autres arbres utilisés pour la prédiction. Encore une fois, étant un domaine de recherche en soi, un aide-mémoire assez exhaustif des concepts de base est disponible dans [48].

La dernière méthode, appelée *MART*, consiste à créer un très grand nombre (quelques milliers) de petits arbres simples dont chacun possède un très faible pouvoir prédictif. Puis, à l'aide d'une

famille métaheuristique nommée *boosting*, ces arbres seront ordonnés dans une chaîne ayant un fort pouvoir prédictif. Ainsi, pour effectuer une prédiction, il faut parcourir chaque arbre simple puis fournir la sortie de chacun à l'entrée du prochain dans la chaîne. Toutefois, l'étude des algorithmes de *boosting* sort du cadre de ce travail. Pour plus de détails, un aide-mémoire exhaustif de la base de ce domaine de recherche distinct est accessible dans [49].

D'abord, il est intéressant de recenser certaines méthodes utilisées pour la prédiction de performance dans un sens plus large en étendant la recherche à du matériel informatique qui n'est pas nécessairement graphique. Dans cette optique, les arbres de régression sont utilisés pour la prédiction de performance de processeurs tout-usage créant un modèle à partir de leurs caractéristiques internes (taille de cache, taille de pipeline, latence d'accès à la mémoire, etc.) à l'aide de *MART* [50] ou à l'aide d'arbres de régression simples [51]. Y. Zhang et al. [52] combinent un modèle analytique avec une forêt d'arbres décisionnels pour une prédiction plus précise des performances et de la consommation d'énergie du matériel graphique.

Les arbres de régression démontrent une bonne résistance aux données aberrantes puisque la probabilité que les branches décisionnelles représentant de telles données soient empruntées est très basse. D'ailleurs, ces branches peuvent être éliminées durant une phase de post-optimisation après la création de l'arbre.

2.2.4.2 Prédiction de performance par réseaux neuronaux artificiels

Les réseaux neuronaux artificiels modélisent le comportement neurobiologique des neurones du cerveau. Organisés sous forme de graphe cyclique ou acyclique, il existe trois types de nœuds disposés minimalement en trois couches : nœuds d'entrée à la première couche, nœuds internes de la deuxième à un nombre arbitraire de couches et enfin les nœuds de sortie à la dernière couche. La topologie du graphe, l'algorithme d'apprentissage utilisé et la fonction d'activation et de transfert utilisés déterminent la nature et la fonctionnalité d'un réseau neuronal. Il est appellé *feedforward* s'il possède une topologie acyclique, tandis qu'il est plutôt appelé *récurrent* s'il est cyclique. S'il possède plusieurs couches de nœuds internes, il est qualifié de *multicouches*. Si chaque nœud d'une couche *i* possède un lien vers chaque nœuds de la couche *i+1*, alors il est qualifié de *pleinement connecté*, sinon il est *partiellement connecté*. Quelques exemples de topologies classiques : dans sa forme la plus simple, on retrouve les *Perceptrons* qui sont des graphes acycliques pleinement connectés entraînés avec des algorithmes d'apprentissages

supervisés servant à faire de la classification dichotomique. Une autre version ayant la même topologie, mais utilisant un algorithme d'apprentissage non-supervisé sera plutôt appelée un *Self-Organizing Map* pour créer des grappes à partir des données d'entrée. D'ailleurs, le réseau neuronal le plus adapté à l'ajustement de courbe utilisé pour bâtir un modèle de performance est un réseau *feedforward* pleinement connecté avec un seul niveau de nœuds internes, entraîné de façon supervisée. Il reste toutefois à évaluer expérimentalement le nombre de nœuds adéquats à placer dans cette couche interne. Une des façons de modéliser une fonction inconnue *F(X)* est à l'aide d'une somme pondérée : il s'agit du modèle neuronal McCullock-Pitts. Selon ce modèle, chaque nœud i du graphe retourne en sortie f_i la somme de ses entrées $X_i = \{x_1, x_2, \dots, x_n\}$, chacune multipliée par un poids w_j puis passé dans une autre fonction appelée fonction d'activation $A(x)$. Cette fonction d'activation peuvent avoir plusieurs formes : sigmoïdale, logarithmique, tangente hyperbolique, gaussienne, pour n'en nommer que quelques-unes. La valeur de sortie du *i-ième* nœud est alors passée en entrée à tous les nœuds qui lui sont connectés, puis le même processus recommence récursivement jusqu'aux nœuds de sorties.

$$f_i(X_i) = A(\sum_{j=1}^{|X_i|} w_j \cdot G_i(X_{ij})) \text{ où } G_i(X_{ij}) = X_{ij} \text{ pour cette méthode.}$$

$$F(X) = A(\sum_i f_i(X_i)$$

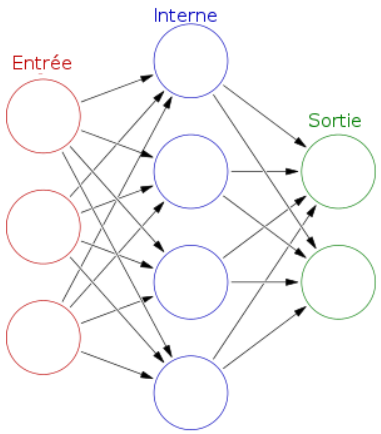

Figure 2-4: Réseau neuronal acyclique avec un seul niveau de nœuds internes.
Auteur : Glosser.ca. Traduis de l'anglais par Simon Rivard-Girard.
Source : http://commons.wikimedia.org/wiki/File:Colored_neural_network.svg
License : CC-BY-SA 3.0 http://creativecommons.org/licenses/by-sa/3.0/

Les algorithmes d'apprentissage peuvent être supervisés, non-supervisés ou par renforcement. Les algorithmes d'apprentissage ont pour but entre autres de trouver le poids de chaque nœud qui minimise l'erreur de prédiction, ainsi que le nombre de fois qu'il faut effectuer la modification de ces poids (communément appellé le nombre d'*époques*) pour obtenir une erreur de prédiction plus

petite qu'un seuil arbitraire. Les métriques les plus communes pour calculer l'erreur de prédiction sont l'écart quadratique moyen ou la somme d'erreur au carré entre la sortie du réseau neuronal et la valeur réelle dans l'ensemble d'entraînement de la donnée associée aux entrées fournies au réseau neuronal. La méthode la plus commune d'implémenter ces algorithmes se fait à l'aide de techniques de rétropropagation des erreurs. Plusieurs algorithmes existent pour accomplir ce problème de minimisation, tels *Levenberg-Marquardt, quickprop, irprop*, pour n'en nommer que quelques-uns. Il est aussi possible d'utiliser une méthode dite *en cascade* qui permet de trouver les connections optimales entre les nœuds pour un réseau neuronal partiellement connecté. Pour augmenter le pouvoir de généraliser une solution, un réseau neuronal peut être amélioré de trois façons. Premièrement, en ajoutant des contraintes d'arrêt précoce de l'entraînement, soit en donnant une limite maximale du nombre d'époques, en ajoutant une phase de vérification à chaque époque en vérifiant si l'erreur minimale est atteinte avant que l'algorithme de minimisation se complète, ou encore en vérifiant d'autres paramètres propres à chaque technique de rétropropagation. Cette technique donne de bons résultats, toutefois il est important de s'assurer de ne pas faire terminer l'entraînement trop précocement, surtout lorsque des algorithmes de minimisation déjà rapides tel *Levenberg-Marquardt* sont utilisés. Deuxièmement, en améliorant la métrique de calcul de l'erreur de prédiction en faisant une moyenne pondérée entre l'écart quadratique moyen des erreurs de prédiction et l'écart quadratique moyen des poids des neurones. Cette technique appelée *régularisation* performe particulièrement bien lorsque l'ensemble de données d'entraînement est petit. Toutefois il est difficile de paramétrer le poids associé à chaque écart quadratique moyen (c.f. [53]). Troisièmement, l'ensemble de données devrait être mis à l'échelle pour que chaque valeur de la variable dépendante soit dans l'intervalle [-1, 1]. Étant un domaine de recherche en soi, l'étude des réseaux neuronaux dépasse largement l'optique de ce travail. Pour plus de détails, un aide-mémoire assez complet est disponible dans [54].

D'autre part, leur utilisation pour la prédiction de performance est assez répandue dans la littérature. D'abord, les travaux d'une équipe de recherche [55, 56] présentent la création d'un modèle de prédiction pour applications parallèles en créant un réseau neuronal acyclique à multiples niveaux internes, à partir des vitesses d'exécutions de divers programmes et de la quantité de données qui leur est passé en entrée. Plusieurs programmes ont été évalués : *Multigrid* qui résoud des systèmes linéaires résultants d'équations différentielles, et *Linpack* qui résoud de denses systèmes linéaires de façon récursive. Le parallélisme est effectué à l'aide de multiples processeurs

embarqués sur des ASIC reliés ensemble sur un réseau local. Il est intéressant de voir qu'ils utilisent les métriques de pourcentage d'erreur et d'écart-type d'erreur de prédiction pour évaluer le pouvoir prédictif de leur modèle. Ils ont remarqués que les données utilisées pour entraîner les réseaux neuronaux contiennent beaucoup de bruit et que la fonction d'activation joue un rôle très important dans la précision des prédictions.

Les réseaux neuronaux sont aussi utilisés pour la prédiction de charge de travail en fonction des caractéristiques des composantes internes des processeurs d'usage général [57], des processeurs superscalaires [58] ou encore de microarchitectures [59].

2.2.4.3 Deep Learning

Une autre famille d'algorithmes par apprentissage automatique est utilisée dans la littérature : le deep learning. Cette famille algorithmique consiste à hiérarchiser la fouille de données : des features ou patterns de bas-niveau sont d'abord extraits. Par exemple : recherche de lignes dans une image. Puis, un modèle est entraîné sur ces ensembles de patterns. Par la suite, des patterns de plus haut-niveau sont extraits à partir de l'ensemble de patterns précédemment extraits. Par exemple : recherche de polygones à partir de l'ensemble de lignes extraites au niveau hiérarchique inférieur. Ainsi de suite, la hiérarchisation peut s'étaler sur plusieurs niveaux d'abstractions et un modèle est généré à partir de l'ensemble des patterns pour chaque niveau abstraction. La littérature présente au moins une méthode de prédiction de la consommation d'énergie de matériel informatique basé sur cette famille algorithmique : [60]. Toutefois, la prédiction de performance ne semble pas encore avoir été abordée.

2.2.4.4 Multicube Explorer

Cet outil unique en son genre permet la création de modèles de performance dans le but de faire de l'exploration architecturale. La méthode sous-jacente utilise des hypercubes et, ainsi que la théorie du *design of experiments* (c.f. [61]). Les prédictions de performance extraites par cet outil seraient intéressantes à comparer à celles générées par les modèles entraînés à l'aide de la contribution de ce mémoire.

2.2.5 Détermination du seuil acceptable d'erreur de prédiction

L'erreur de prédiction est souvent quantifiée soit à l'aide de la moyenne absolue ou relative de l'erreur de prédiction, ou soit à l'aide de l'écart quadratique moyen. Toutefois, la méthode préférable à utiliser est encore un débat ouvert dans la littérature [62]. Pour certains, les moyennes sont préférées à l'utilisation de l'écart quadratique moyen (MSE) entre la prédiction et la valeur réelle, car cette dernière donne un plus grand poids aux valeurs aberrantes par sa nature à mettre au carré chaque erreur de prédiction. Toutefois, l'avantage d'utiliser la métrique MSE est qu'elle permet d'obtenir l'équivalent de l'écart type de l'erreur de prédiction (RMSE) en faisant la racine carrée du MSE lorsque la distribution des erreurs est normale. D'ailleurs, lorsque le RMSE est donné dans un article et que la distribution d'erreur est normale, il est possible pour le lecteur de créer un intervalle de confiance d'erreur de prédiction suivant les règles statistiques standard si la moyenne de la distribution d'erreur est également fournie avec la même unité que l'écart type (pourcentage vs unité quantitative), ce qui semble rarement être le cas. Lorsque l'unité de la valeur prédite est difficilement interprétable, certains auteurs préfèrent représenter l'erreur de prédiction en termes de pourcentage, ce qui est une technique intéressante. Ainsi, la littérature est plutôt partagée sur le choix de l'une ou l'autre de ces métriques permettant la meilleure évaluation des modèles de prédiction. Le RMSE est certainement à prioriser lorsque la distribution est normale. Toutefois, il est rare d'obtenir une telle distribution avec de l'apprentissage automatique. L'erreur moyenne relative ou absolue est aussi significative pour une distribution normale mais peut retenir de la significativité lorsque la distribution n'est pas normale. La méthode d'évaluation de la tendance centrale de la distribution à utiliser dans un cas non-normal serait donc plutôt la médiane, toutefois on la retrouve peu dans la littérature. En fait, aucun auteur ne semble donner leur distribution d'erreur, et chacun lance une métrique de tendance centrale dont il est difficile d'interpréter la significativité statistique. Le tableau 2-1 présente quelques exemples de ces valeurs selon certains articles recensés précédemment.

Tableau 2-1: Pourcentage d'erreur moyen de prédiction et écart type d'erreur de prédiction de performance dans la littérature.

Article	**Pourcentage moyen d'erreur de prédiction (%)**	**RMSE (%)**
[42]	Entre 0,2 et 8,8	N/A
[50]	Entre 2,74 et 13,25	N/A
[51]	Environ 7.83	N/A
[55]	Environ 4,9	Entre 4,4 et 8,4
[57]	Environ 5	N/A
[58]	Environ 2,8	Environ 2,14
[59]	Entre 2,5 et 6,4	Entre 1,8 et 5,2

2.2.6 Prédiction du temps de transfert entre la mémoire centrale et graphique

Le matériel graphique possède la plupart du temps une mémoire auxiliaire qui ne fait pas partie de la mémoire vive du système computationnel central. Dans ce cas, l'opération la plus coûteuse en termes de temps d'exécution est l'envoi des données de la mémoire centrale vers la mémoire graphique via un bus, souvent PCI ou PCI Express. Par contre, certains systèmes supportent une architecture avec mémoire partagée et DMA, de sorte qu'ils n'en sont pas affectés. Pour les autres systèmes, il est toutefois important de tenir compte de ce délai lors de la modélisation des performances graphiques. Des travaux récents [63] démontrent qu'il est possible de modéliser le temps de transfert *T* en fonction linéaire de la taille en octets du transfert *d* en tenant compte du temps α pour envoyer le premier octet et du temps β pour envoyer les $d - 1$ octets suivants.

$$T(d) = \alpha + \beta d$$

2.3 Simulation de l'environnement *OpenGL SC*

Étant donné le très haut coût du matériel graphique avionique, il est souvent d'intérêt d'utiliser un simulateur de l'environnement sur lequel l'application graphique va s'exécuter en utilisant du matériel moins dispendieux. Ainsi, il est intéressant de fournir une interface *OpenGL SC* à une application graphique avionique, mais d'en faire l'exécution sur un ordinateur de bureau

possédant des pilotes *OpenGL*, ou encore *Direct3D* s'il s'agit d'un système d'exploitation Windows. La littérature présente ce genre d'outil [64]. Toutefois, le code source ou l'exécutable n'est pas librement disponible. Une alternative consiste à remplacer l'interface du simulateur *OpenGL SC* par *OpenGL ES 1.x* qui techniquement offre les mêmes fonctionnalités et qui utilise également un pipeline fixe et pour lequel il existe un simulateur d'environnement librement disponible. Il s'agit d'un outil fournie dans la suite *PowerVR SDK* de la compagnie *Imagination Technologies Limited* © disponible gratuitement : [65]. Toutefois, il faut faire attention lors de l'implémentation de l'application graphique afin de n'utiliser que des fonctions qui sont disponibles dans *OpenGL SC* et d'éviter l'utilisation des quelques fonctions supplémentaires disponibles dans *OpenGL ES 1.x*. C'est d'ailleurs cet outil qui est utilisé par V. Legault dans la réalisation de son évaluateur de performance pour applications graphiques en avionique, présentée à la section 2.1.3. Il s'agit en fait d'une méthode très intéressante pour l'accomplissement de l'OS3.

La revue de la littérature a permis d'identifier les principales méthodes utilisées dans la littérature pour faire de la prédiction de performance de matériel graphique et plus largement de matériel informatique. Les algorithmes d'apprentissage automatique présenté, soit : les forêts d'arbres décisionnels, MART et les réseaux neuronaux sont donc les meilleurs candidats à être adaptés à l'avionique et sont ceux qui seront utilisés dans ce travail. La revue de la littérature a aussi permis d'établir les métriques d'erreurs, soit l'écart-type et le pourcentage d'erreur de prédiction. De plus leur valeur seuil étant considéré comme « acceptable » est établit entre 1 et 10.

CHAPITRE 3 PRÉSENTATION DE L'ARTICLE

L'article en soi n'utilise pas la notation d'objectifs et d'hypothèses telle que retrouvée dans ce mémoire (OS#, OG#, H#). Toutefois, de façon indirecte, l'article présente l'outil réalisé dans le cadre de l'OG1, l'analyse des divers algorithmes de génération de modèles non-paramétriques réalisé dans le cadre de l'OS1, l'outil d'évaluation de performance amélioré réalisé dans le cadre de l'OS2, la méthode de simulation d'environnement *OpenGL SC* réalisée dans le cadre de l'OS3, ainsi que les résultats permettant de confirmer ou infirmer l'hypothèse H1. Un lien entre les conclusions élaborées dans l'article et la notation d'objectifs de ce mémoire est présenté au chapitre 6. L'article en soi regroupe deux travaux de recherche, le premier étant l'outil d'évaluation des performances de V. Legault [6] et le second étant le présent travail. Toutefois, des améliorations sont portées à l'outil d'évaluation de performance dans le cadre de l'OS2, et l'article présente cet outil en tenant compte de ces améliorations. Ainsi, l'analyse de l'influence de ces améliorations sur l'outil initial ne fait pas partie de l'article et est plutôt présentée dans le chapitre 5 de ce mémoire en guise de résultats complémentaires. D'autre part, l'auteur principal de l'article est le même étudiant qui rédige ce mémoire. L'article présente principalement les résultats de recherche reliés à l'outil de prédiction de performance en considérant l'outil d'évaluation de performance comme étant une section parmi le flot de données général de la prédiction. La division de la charge de travail pour l'ensemble de l'activité de rédaction de l'article est pondérée comme suit : 50% contenu, 40% rédaction, 10% révision. La charge de travail est donc divisée entre les deux auteurs S.R.-Girard et V.Legault comme suit :

TÂCHE	PONDÉRATION S. R.-GIRARD (%)	PONDÉRATION V. LEGAULT (%)
CONTENU (50%)	30	20
RÉDACTION (40%)	40	0
RÉVISION (10%)	10	0
TOTAL	80	20

En plus des deux auteurs ayant généré du contenu expérimental, les deux directeurs du projet CRIAQ AVIO509 sont ajoutés à la liste d'auteur : M. Guy Bois, professeur et docteur à l'École Polytechnique de Montréal, et M. Jean-françois Boland, professeur et docteur à l'École de Technologie Supérieure. L'article est en processus de révision pour le journal *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS).*

CHAPITRE 4 ARTICLE 1 : AVIONICS GRAPHICS HARDWARE PERFORMANCE PREDICTION WITH MACHINE LEARNING

4.1 Abstract

Within the strongly regulated avionics engineering field, conventional graphical desktop hardware and software API cannot be used because they do not conform to the DO-254 and DO-178B certifications. We observe the need for better avionic graphical hardware, but system engineers lack system design tools related to graphical hardware. The endorsement of an optimal hardware architecture by estimating the performance of a graphical software, when a stable rendering engine does not yet exist, represents a major challenge. There is also a high potential for development cost reduction, by enabling developers to have a first estimation of the performance of its graphical engine at a low cost. In this paper, we propose to replace expensive development platforms by a predictive software running on a desktop computer. More precisely, we present a system design tool that helps predict the rendering performance of graphical hardware based on the OpenGL SC API. First, we create non-parametric models of the underlying hardware, with machine learning, by analyzing the instantaneous frames-per-second (FPS) of the rendering of a synthetic 3D scene and by drawing multiple times with various characteristics that are typically found in synthetic vision applications. The number of characteristic combinations used during this supervised training phase is a subset of all possible combinations, but performance predictions can be arbitrarily extrapolated. To validate our models, we render an industrial scene with characteristics combinations not used during the training phase and we compare the predictions to those real values. We find a median prediction error of less than 4 FPS.

4.2 Introduction

In recent years, there has been an increased interest in the avionics industry to implement high performance graphical applications like augmented or synthetic vision systems (AVS, SVS). This has promoted the advent of faster graphical processing hardware. Because it is a highly regulated field, conventional desktop and embedded graphics hardware could not be used because they do not conform to the DO-254 certification and their application programming interface does not conform to the DO-178B certification [Dutton 2010; Hilderman 2007]. Considering the need of avionic hardware with higher performance, we observe that graphical application development

tools and hardware benchmarks and simulators available for conventional embedded or desktop graphical applications seem still to be missing for avionics applications. This fact is made especially clear when a quick search through the specifications of the most renowned tools such as Nvidia NSight [Nvidia 2015], AMD Perfstudio [AMD 2015] or SPECViewPerf [Corporation 2007] lead to the same conclusions. Even though, there are some wysiwyg ("What You See Is What You Get") GUI toolboxes available for the ARINC-661 standard [Presagis 2015; Wang 2014], it seems that there is no performance benchmark, performance prediction tool or performance-correct simulator availables for avionic graphical hardware. The interest in having such tools is especially significant because most development process include a design phase before the actual implementation. Taking example on the classical v-method, designers must make choices in regard to the purchase or the in-house development of graphical hardware. But as they want to evaluate the performance of such hardware relating to the choices made, they need some kind of performance metrics and benchmarks. This benchmarking tool should be provided by the software development team but, as the project is still in the design phase, they not yet necessarily have implemented a graphical engine to enable performance testing. Performance prediction can be useful to: 1) further extrapolate the benchmark performance results for any volume of graphical data sent to the hardware and 2) reduce the number of benchmarks required to evaluate various use cases. Going further, the performance models generated can then be used to develop a performance-correct hardware simulator that developers can use on their workstation, in order to have a general preview of the efficacy of their software, before executing it on the real system. This should reduce the development costs because less hardware has to be purchased.

In this work, we propose a set of two tools that can be used as a pipeline to evaluate and then predict the performance of graphical hardware. The first tool is a benchmark that can generate and then render custom procedural scenes according to a set of scene characteristics such as number of vertices, size of textures and more. It evaluates and outputs the number of frames generated per seconds (FPS). The second tool takes the output of a certain number of executions of the benchmark and generates a non-parametric performance model, by using machine learning algorithms on the performance data. Those performance model can then extrapolate predictions of performance for any dense 3D scene rendered on this piece of hardware. We evaluated the distribution of prediction errors experimentally to find that most prediction errors will not exceed 4 FPS.

In the rest of the paper, Section 2 presents the main problems which make inadequate the existing aforementioned tools for the avionics industry. It also presents the work related to the various algorithms and methods used by those standard tools. Section 3 presents the first contribution which is the avionic graphical application benchmarking tool. Then, Section 4 presents the second contribution which is the performance model generation tool. Section 5 present the experimental method used to evaluate the prediction power of these models. Finally, Section 6 presents, analyses and discusses the achieved prediction error distributions.

4.3 Background

Among the differences between conventional (consumer market) and avionics graphic hardware development, three are denoted as especially standing out. The first is the use of OpenGL SC instead of OpenGL ES or the full OpenGL API to communicate with the hardware [Khronos 2009]. The second is the use of a fixed graphics pipeline instead of letting the possibilities of using shaders or custom programs that can be sent to the graphics hardware to modify the functionalities of certain areas of the rendering pipeline [Cole 2005]. Finally, there is the research interest in the development of DO-254-complient graphical hardware, in the form of software GPUs, FPGAs and CPU/GPU on-a-chip to name a few [Dutton 2010]. The nature of the hardware is then not necessarily a processor, and certain metrics specific to that nature cannot be applied, such as instruction count. Also, avionics graphical hardware is usually a very secured black box that cannot be intruded to actually perform the instruction count metrics on the internal programs. Thus, performance benchmark and prediction tools in an avionics context should account for these specificities. By only using the functions available in OpenGL SC to evaluate the performance of the hardware, we make sure of two points: 1) to use the standard fixed pipeline that accompanies this version of the API and 2) to not be dependent on the nature of the underlying hardware beyond that interface.

It is then interesting to look over the literature to find the methods that have been used in a conventional desktop and embedded context. It is also interesting to widen this review to general computer hardware and microarchitecture, as well as graphical hardware performance prediction. Numerous benchmarks for graphics hardware exists in the conventional context, such as SPECViewPerf or Basemark [Corporation 2007; Rightware], to name a few. Even if they do not satisfy the special problematic of the avionics needs, their workflow can be a source of inspiration.

For example, SPECViewPerf allows users to create a list of tests, each varying different characteristics of the scenes or the render state, such as local illumination models, culling, texture filters, simple or double buffering. It then returns the average FPS attained during the rendering of the scene for each test. The use of the average FPS might be more significant for the user, but because the FPS distribution is not normal it loses a lot of statistical significance. As for the performance prediction tools, they tend to be made available by the graphic hardware manufacturers such as the NVIDIA® Nsight™ [Nvidia 2015] or the AMD GPU PerfStudio [AMD 2015]. The problem with these tools is that they are only available for the desktop and embedded domains and are not adapted to the needs of avionics, as explained previously. It is still interesting to review the literature to better understand how those profiling tools might work internally. There are three main approaches to generate models: analytical modeling, parametric modeling and machine learning. Analytical models attempt to create mathematical models that represent performance as a set of functions describing the hardware. They often use metrics such as instruction count and properties such as frequency clock of the processor [Baghsorkhi et al. 2010; Hong and Kim 2009; Kanter 2011; Yao and Owens 2011]. The main issue with these methods is that they require a good understanding of the hardware's inner workings, which is difficult in an avionics context because they are secured black box entities. Also, because of the fixed pipeline of the graphics card, system engineers cannot obtain the inner programs operating the pipeline and thus cannot use analytical metrics such as instruction count. Also, the literature seems to indicate that it is very difficult to truly identify all factors influencing performance and thus to mathematically model them. However, there is one analytical model that has the potential to be used in an avionics context. It is a function that estimates the transfer time of data from the main to the graphic memory [Boyer et al. 2013].

The creation of parametric models implies the use of parametric regressions such as linear or polynomial regressions. It has been used for microarchitecture and CPU design-space exploration [Joseph et al. 2006; Lee and Brooks 2006; Lee and Brooks 2007; Marin and Mellor-Crummey 2004]. But, because we only have access to the interface of the hardware, it is hard to identify all the factors influencing the performance. Thus, these methods would have difficulty to explain most of the variance of the performance data and can then perform poorly. This is usually solved by using non-parametric regression models created with machine learning.

There are four machine learning algorithms that are mainly used throughout the literature to generate performance models: Regression Trees, Random Forest, Multiple Additive Regression Trees (MART) and Artificial Neural Networks. Performance and power consumption prediction in the case of design-space exploration of general purpose CPUs has been achieved with Random Forests [Zhang et al. 2011] and MART [Li et al. 2009]. Regression Trees [Ould-Ahmed-Vall et al. 2007] have been used for performance and power prediction of a GPU. Artificial Neural Networks have successfully been used for performance prediction of a parallelized application [Lee et al. 2007], but also for workload characterization of general purpose processors [Yoo et al. 2006], superscalar processors [Joseph et al. 2006] and microarchitectures [İpek et al. 2006]. Regression Trees are usually less accurate, and its more robust version, the Random Forest, is usually preferred.

4.4 Avionics Graphic Hardware Performance Benchmarking

There are two main steps in the creation of our performance models. First, a benchmark must be executed to gather performance data for various scene characteristics. The GPU benchmarking consists in itself in the generation of a customizable synthetic 3D scenes and in the analysis of the render time of each frame. Second, performance models are generated with machine learning from the performance data obtained in the first step. For our experimental purposes, we add a model validation phase to evaluate the predictive power of those performance models by comparing the predictions with the render time of a customizable and distinct validation 3D scene. Fig. 4-1 presents this dataflow. The remainder of this section will present the requirements and the implementation of our proposed avionics GPU benchmarking tool.

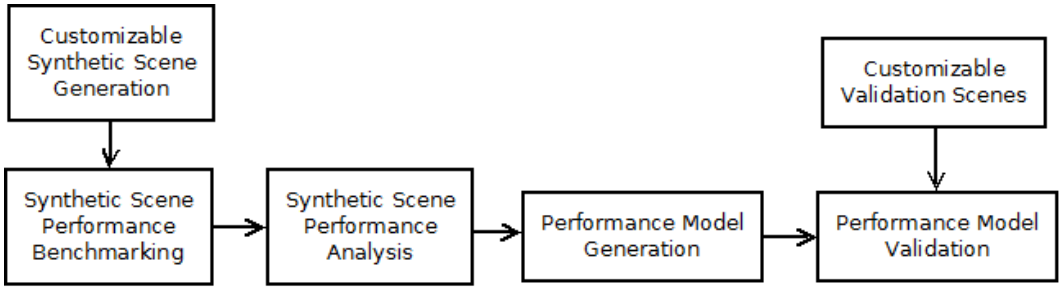

Figure 4-1: Dataflow of the proposed tool in an experimental context.

Performance data acquisition for a piece of hardware is achieved with our benchmarking tool as follows. First, a synthetic scene is generated according to various parameters. Then, the scene is rendered and explored by following a specific camera movement pattern. Finally, a last analysis step is performed to evaluate for each frame the percentage of the number of vertices of the scene that has been rendered. We use a study case from an industrial partner to enable us to enumerate the various characteristics of graphical data that has an impact on the rendering performance of an avionic graphical application. The study case was a SVS using tile-based terrain rendering.

The various factors found were:

1. Number of vertices per tile
2. Number of tiles per scene
3. The size of the texture applied on the tiles
4. The local illumination model, either: per-vertices or per-fragment
5. Presence or absence of fog effect
6. Dimension of the camera frustum
7. The degree of object occlusion in the scene

From these factors, we divide a tile-based synthetic scene that would evaluate rendering performance based on these factors. The benchmark tool takes a list of “tests” as input. Each test influences the generation of the procedural 3D scene by manipulating a combinations of those factors. The output of the benchmark tool is a file with time performance according to the input characteristics. Each test is designed to evaluate the performance of the scene rendered by varying one of the characteristics and keeping fixed every other. Consider for instance the tile resolution test, for each value, the benchmark will be executed, and a vector of performance will be output. During this test, every other characteristic (e.g., the number of tiles or the size of textures) shall be fixed. It is important to mention that tile-based scenes are stored as height maps or even dense 3D scenes, removed the need for analysing the number of triangles or faces, because it can always be derived or approximated from the number of vertices.

4.4.1 Synthetic Scene Generation

Each tile of our synthetic scene contains a single pyramidal shaped mesh. We used this shape because it can model various ground topography by varying the height of the pyramid. Furthermore, it enables the possibility to have object occlusion when the camera is at a low altitude and it is oriented perpendicularly to the ground. Also, this shape is easy to generate from a mathematical model. The remaining of this subsection presents how the visual components of the procedural 3D scene are generated and how they help to produce more representative performance data.

Tiles Dimensions: Tiles have a fixed dimension in OpenGL units, but the number of vertices it can contain can vary depending on the corresponding benchmark input value. To simplify the vertices count of our models, we use a per-dimension count c for the square base of the pyramids, meaning that each tile has a resolution of c^2 vertices. When the perspective distortion is not applied, each vertice of the pyramid is at equal distance of its neighbours in the XZ plane (see Fig. 4-2).

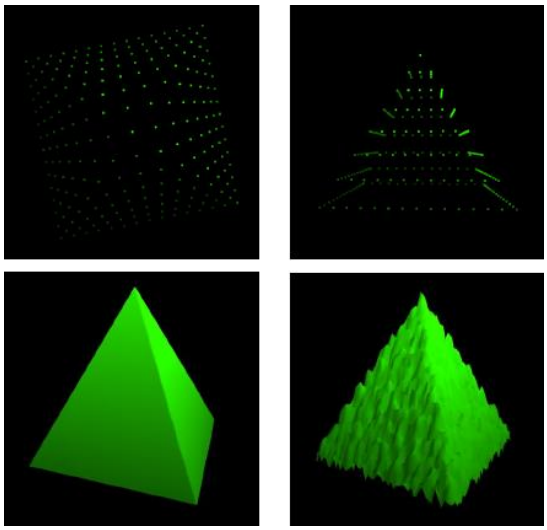

Figure 4-2: Pyramid vertices generated with a c-by-c dimension top-facing (top-left) and front-facing (top-right). Pyramid rendered mesh without added noise (bottom-left) and with added noise (bottom-right).

Noise: To further reproduce a realistic ground topography, we add random noise to the pyramid faces to unsmooth them. The quantity of noise applied is more or less 10% of the height of the pyramids, and is only applied to the Y coordinates (attributed to the height) as shown in Fig. 4-3. This proportionality helps to keep the general shape of the pyramid, regardless of its height.

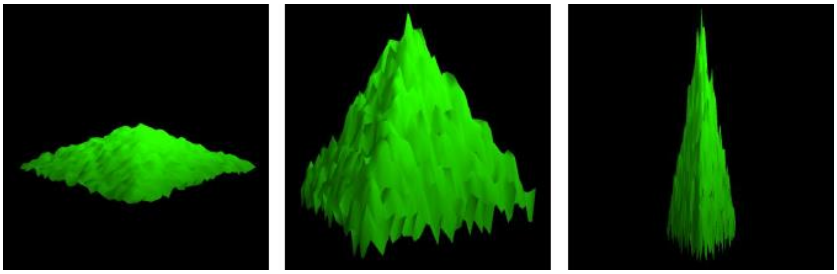

Figure 4-3: Various intensity of noise depending on the height of the pyramid. Low noise amplitude (left) to high noise amplitude (right).

Grid Generation: The grid of tiles is generated according to the corresponding benchmark input value. As for the tile resolutions, the grid size is measured as a per-dimension value v, meaning that the total grid size is v^2 , and thus that the grid has a square shape. In a real context, a LOD functionality is usually implemented, making farthest tiles load at a lower resolution and nearest tiles at a full resolution. However, because we evaluate the worst case execution performance of the hardware, every tile has full resolution.

Pyramid Height: The height of the pyramids varies from tile to tile, depending on their position in the tile grid, but the maximum height will never exceed the quarter of the length of the scene. This constraint enables the possibility to have various degrees of object occlusion for the same scene, depending on the position and orientation of the camera. Because of the positioning of the camera and the movement pattern (explained previously), the bigger the tile grid is, the higher the pyramids are. To obtain consistent scene topologies for each benchmark test, the pyramids height is calculated from the index of the tile in the grid (see Fig. 4-4) and is always a factor of two from the maximum pyramid height.

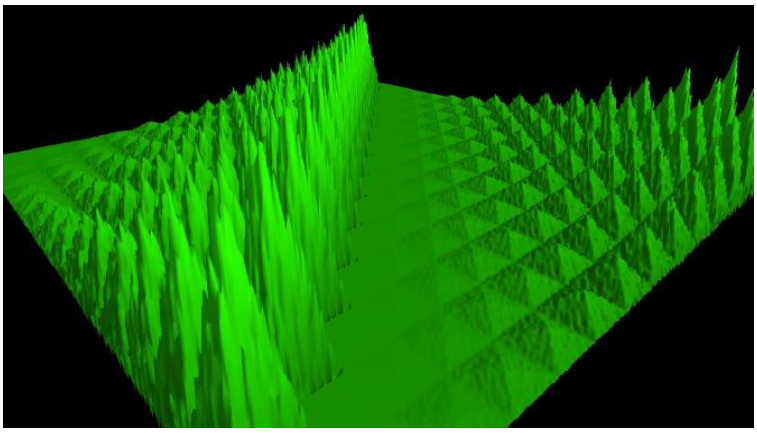

Figure 4-4: Overall generated synthetic scene with pyramids height varying according to their position in the grid.

Texture Generation: The OpenGL SC API requires the use of texture dimensions that are powers of two. For simplicity, we create RGB-24 procedural textures which consist of an alternation of white and black texels. For each vertice of the tile, the texture itself and the texture coordinates are computed before the frame rendering timer starts. In real cases, these informations are normally already available in some kind of database and not generated in real time, so it should not be taken into account by the timer measuring the period taken to draw the frame.

4.4.2 Camera Movement Pattern

According to the case study, there are three typical use cases for the camera movement and position patterns:

1. Low altitude: a small percentage of the scene is rendered with the possibility of much object occlusions;
2. Mid-range altitude: about half of the 3D objects are rendered with possibly less object occlusions;
3. High altitude: the whole scene is potentially rendered with low chances of object occlusions.

For each test of a benchmark, the camera position goes through each of these use cases. To achieve this, the camera always starts at its maximum height over the tile at the middle of the grid. The camera then performs a 360 degrees rotation in the XZ plane, while also varying its inclination over the Y axis, depending on its height. After each 360 degrees rotation, the camera height is reduced and there are eight possible values for each test. The inclination angle over the Y axis is not constant throughout the various height taken by the camera, in order to cover the highest possible number of viewpoints of the scene. At the maximum height, the inclination leans towards the edges of the grid, and at the lowest height the camera points perpendicularly towards the ground. The camera inclination for every camera height is calculated with a linear interpolation between the inclinations, at maximum and minimum heights. Overall each 360 degrees rotation of the camera will yield 32 frames for a total of 320 frames for each benchmark run.

The camera frustum is created to mimic the one used by the study case SVS. It implements a 45 degrees horizontal field of view and a vertical field of view corresponding to the 4:3 ratio of the screen, according to the OpenGL standard perspective matrix. Also, to maximize the precision

of the depth buffer, it is desirable to show a maximum of vertices with the smallest frustum possible. The last important parameter is to define the maximum height of the camera. We set this limit to the value of the length of the scene in OpenGL units, because the scene will be smaller than the size of the screen passed that length. Thus, the far plane of the frustum must carefully be chosen in regards of the scene length, as the maximum depth of the scene will most likely vary accordingly.

4.4.3 Loading Data to the Graphic Memory

To help reduce the randomness of the performance of the graphics hardware, and due to its internal properties, we apply the tipsify algorithm [Sander 2007] to the vertex index buffer before sending it to the graphics pipeline. This should reduce the average cache miss ratio of the standard internal vertex program of the fixed pipeline. All the 3D data is loaded to the graphic memory before beginning the rendering and the performance timer. Because the scene is static, no further data needs to be sent to the hardware, so the loading time does not influence the overall performance. This would not be the case in a real context but, as presented in the Section 2, the literature presents at least one method to estimate the influence of data loading during the rendering process. If needed, the benchmark tool can return the time required to load this static data from the main memory to the graphics memory.

4.4.4 Analysis of the Percentage of Scene Drawn

The data sent to the graphical hardware for rendering usually contains 3D objects that could be ignored during the rendering process because they are either unseen or hidden by other 3D objects, due to their spatial positioning. Thus, culling methods are commonly used to avoid the rendering of such objects. These methods are: 1) the frustum culling which ignores the rendering of triangles outside of the camera frustum, 2) the back-face culling which ignores the rendering of triangles that are facing away from the camera orientation and 3) the z-test which ignores the per-fragment operations such as the smooth lighting.

As mentioned earlier, the final step of the benchmarking process is the analysis of the percentage of the vertices of the scene that were used during the rendering process. To do this, we count the number of vertices that are in the camera frustum and also that are part of front-facing surfaces. We used a geometric approach by comparing the position of each vertex with the six

planes of the frustum in order to determine if the vertex is inside it or not. If it is, the next step is to determine if it is front or back-facing. To do this, we first transform the normal of the surface, of which the vertex is a part, from world-space to camera-space. Then, we compare the angle between the normal and the camera-space eye vector, which is a unit vector pointing in the Z-axis. If the angle is between 90 and 270 degrees, then the vertex is considered to be front-facing. Finally, we can evaluate the percentage of vertices drawn as the size of the set of vertices that pass both tests, divided by the total number of vertices in the scene.

4.4.5 Rendering Performance Metrics

The performance metric returned by the benchmark is the instantaneous frame per second (IFPS), which is measured for each frame by inversing the time it took to render the frame. It is more desirable than a moving-average FPS over multiple frames because we can then apply more specialized smoothing operations to eliminate further any outliers. On the other hand, the benchmark uses a vertical sync of two times the standard North American screen refresh rate: 120 Hz. We found that not using vertical sync yields a very high rate of outliers for IFPS greater than 120. Also, using a vsync of 60 FPS may create less accurate models as most of the scene characteristics will yield the maximum framerate. Finally, to ensure the proper calculation of the IFPS for each frame, we use the *glFinish* command to synchronize the high resolution timer with the end of the rendering.

4.5 Avionics Graphic Hardware Performance Modeling

The first step in the creation of a performance model is to evaluate which of the benchmark scene characteristics best explains the variance of FPS. In preliminary tests, we ran the benchmark by varying the values of one characteristic, while keeping fixed every other. This is repeated for each characteristic until all of them has been evaluated. We concluded that the size of the grid of tiles and the resolution of vertices in each tile are the most significantly well predicted characteristics by the machine learning algorithms. The size of the screen and the size of the texture also contribute to the variation of the FPS, but they were hard to model using our method. To predict IFPS in terms of texture sizes and screen sizes, a distinct performance model must be generated for each of their combinations, by training it with a subset of every possible combinations of grid size, tile resolution and percentage of vertices rendered. This limitation is being worked on

as a future contribution. To generalize tile-based scenes to dense voxelized scenes without fixed resolutions in each voxel, the tile resolutions can be replaced by the mean number of vertices per voxel.

Besides, another key factor in the creation of good models is the fact that they can be generated efficiently without the need to feed the FPS obtained for every possible combinations of scene characteristics to the learning algorithms. This number has been calculated to be about 58 million combinations of grid size and number of vertices. Compared to this number, the number of combinations of texture sizes (10) and of screen sizes (5) is relatively small. Generating a distinct performance model for each combination of texture and screen sizes would be a more trivial task, if the number of combinations of grid size, tile resolution and percentage of vertices rendered could be reduced. To organize those performance models, we create a three levels tree, where the first two levels represents the combinations of texture and grid size. The third level contains a leaf pointing to a non-parametric model trained with machine learning that predicts IFPS in terms of grid sizes and tile resolutions. Those non-parametric models are created by feeding the machine learning algorithms with only a small percentage of the performance of the whole combinations of grid sizes, tile resolutions and percentage of vertices, while keeping fixed the texture and screen size characteristics according to the leaf parents. The choice of the subset of total grid sizes and tile resolutions combinations to use is chosen by selecting those inducing the worst-case rendering performance. We run the benchmark only twice by running the tile resolution and grid size variation tests and by concatenating the output performance vectors of each. The characteristics evaluated by the benchmark for the training dataset is shown in Table 4-1.

Tableau 4-1: Values used for the tile resolution grid size tests.

Variation test	Values of Tile Resolution	Values of Grid Size
Tile resolution	7;9;13;17;21;25;31;37;45	25
Grid size	25	7;9;13;17;21;25;31;37;45

4.5.1 Machine Learning Algorithms Configuration

As stated in the Section2, three machine learning algorithms are of special interest for the task of performance prediction: Random Forest, MART and Artificial Neural Networks. We offer the comparison between the predictive powers of non-parametric models trained with each of these

algorithms in order to determine which one is the most suited for this application. Each of these algorithms has to be configured before its use: the number of hidden layers and the number of nodes per layers in the artificial neural networks, the number of bootstrapped trees in the case of Random Forest or the number of weak learners in the case of MART. Most of the time, there is no single optimal parameter. It usually takes the form of a range of values. These ranges were found during preliminary experimentation and are given in Table 4-2. In the case of artificial neural networks, we used a multilayer feedforward perceptron trained with backpropagation based on the Levenberg-Marquardt algorithm. We also try to improve the problem generalization and reduce overfitting by using early stopping methods and scaling the input performance values in the range [-1, 1].

Tableau 4-2: Parameters for the machine learning algorithms used

Algorithm	Parameter nature	Optimal range
Artificial Neural Network	Number of hidden layers	1
	Number of nodes in the hidden layer	[5; 15]
MART	Number of weak learners	[500; 1000]
Random Forest	Number of bootstrapped trees	[50; 150]

Because of the random nature of the optimality of a parameter value, we create three performance models for each machine learning algorithm. The first model uses the lowest parameter value, the second model uses the middle-range one and the last model uses the upper one. During the validation phase, we retain the model which yields the lowest prediction error.

4.5.2 Performance Data Smoothing

The performance data output by the benchmark itself is randomly biased, because it cannot explain some of the variance of IFPS, which can vary even for scenes with similar characteristics. The fact that we only analyse the input and output of a graphical hardware blackbox, partially explains the variance, because many internal factors can influence the output for the same input: cache miss ratio, processor instruction pipeline and instruction branching unpredictability to name a few. Because the program running on the hardware is fixed, we can assume that these factors are not enough random to make the analysis of input/output unusable for performance prediction. To reduce this noise in the benchmark output data, we apply an outlier-robust locally weighted scatterplot smoothing. This method, known as LOESS, requires fairly large datasets which is the

case here (around 5760 data points). Similar to the moving average method, this smoothing procedure will average the value of a data point by analyzing its k-nearest points. In the case of the LOESS method, for each point of the dataset, a local linear regression of a one or two degree polynomial is computed with their k-nearest points, by using a weighted least square giving more importance to data points near the analysed initial point. The analysed data point value is then corrected to the value predicted by the regression model. More information is available in [Cleveland and Devlin 1988]. In our case, the k-nearest points correspond to the IFPS of the 6 frames preceding and the 6 frames following the analysed frame. The use of the k-nearest frames is possible because the characteristics of the scene between adjacent frames are spatially related. This can be generalised to most graphical applications because the movement of the camera is usually continuous.

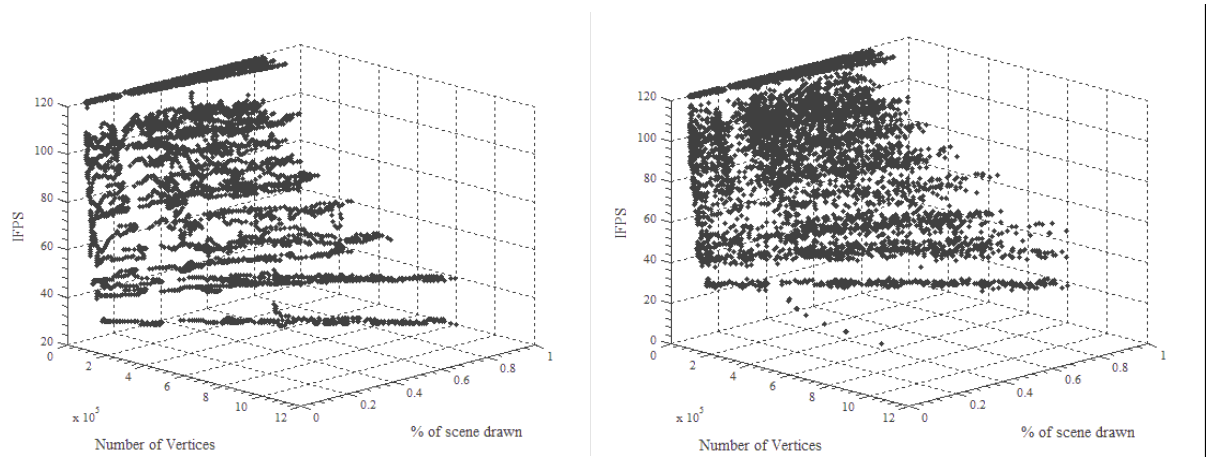

Figure 4-5: Comparison between smoothed (left) and unsmoothed (right) performance data.

4.5.3 Quantifying Scene Characteristics Equivalency

To further help the machine in the performance modeling, we transform the output format of the benchmark (IFPS *in terms of* number of points, scene or grid size, and percentage of scene drawn), into a format that is more similar to the scene characteristics that will be given by the system designer (IFPS *in terms of* number of points and scene or grid size). Also, the tool can be more easily used if the percentages of scene drawn input parameter could be omitted: it might lead to confusion to have to choose a percentage of scene drawn when querying the tool for predictions. Thus, it is necessary to internally find a proportionality factor that can help evaluate the equivalency of performance between various points in the training data. The basic assumption is that the IFPS

of a scene drawn with a certain set of characteristics ($IFPS_1$) will be somewhat equivalent to the IFPS of the same scene drawn with another set of characteristics ($IFPS_2$) iff the characteristics of both scenes follow a certain proportionality. The first characteristic in this case is the size of the scene without accounting for depth: (v_1 for the first set of characteristics and v_2 for the other) either in OpenGL units or in the size of the grid of voxels or tiles in the case of tile-based applications. The other characteristic is the tile resolutions in each tile or voxel c_1 and c_2. As mentioned earlier, those concentration and those sizes in the case of our benchmark are expressed as a per-dimension value. Thus, they are always equal in 2D (length and width). For simplicity we use the following notation:

$$c_i = {c_{width}}_i^{\,2} = {c_{depth}}_i^{\,2} \tag{1}$$

And

$$v_i = {v_{width}}_i^{\,2} = {v_{depth}}_i^{\,2} \tag{2}$$

It implies that :

$$IFPS_1 \approx IFPS_2 \Leftrightarrow \frac{c1}{c2} \propto \frac{v1}{v2} \tag{3}$$

Considering that a scene drawn at a certain percentage p_1 with a set of characteristics, then v_1 represents the fraction of the total v_2 area that is drawn as:

$$v_1 = p_1 * v_2 \tag{4}$$

In this case, since v_1 and v_2 are taken from the same scene, we have $c1 = c2$. Therefore:

$$\frac{v1}{v2} = p_1 * \frac{c1}{c2} \tag{5}$$

where p_1 is the proportionality factor.

This example uses a single scene with a single set of characteristics to help find the proportionality factor, but the formula can be also used to compare scenes with different initial characteristics, which is a powerful metric for extrapolation. During the design of the typical use case of our tool, we assumed that the designer would want to request a performance estimation of the rendering of the scene when it is entirely drawn, and not just drawn at a certain percentage (worst-case scenario). A way had to be found to use the $p_1$factor during the training phase, but to remove the need to use it in the performance queries, once the model is generated.

From (5), we obtain:

$$p_1 * \frac{c_1}{v_1} = \frac{c_2}{v_2} \quad (6)$$

Then, we found that the machine learning can create slightly more precise models if p_1 is expressed in terms of the concentration of triangles instead of in terms of the concentration of vertices. Considering that in our tile-based application the number of *facesPerTile* is obtained as follows:

$$\#facesPerTile = \left(\sqrt{c} - 1\right)^2 * 2 \quad (7)$$

From the proportionality function (6) and from (7) we deduce:

$$p_1 * \frac{(\sqrt{c_1}-1)^2*2}{v_1} = \frac{(\sqrt{c_2}-1)^2*2}{v_2} = K \quad (8)$$

The left part of (8) that can be obtained by the scene characteristics, and the benchmark output performance metrics, provide a value K which, in turn, allows to find values for c_2 and v_2 . We are thus capable of obtaining approximately equal IFPS values between that scene rendered with c_2 and v_2 at 100% and the same scene drawn with c_1 and v_1 at p_1 percent. The machine learning algorithms are then trained with a vector containing a certain number of tuples: (IFPS *in terms of K* and the number of points drawn) where

$$\#pointsDrawn = p_1 * \#totalNumberOfPoints \quad (9)$$

And

$$\#totalNumberOfPoints = c_1 * v_1. \quad (10)$$

The designer can then create a prediction query by inputting K and the number of points he desires to render without having to mind about a percentage of scene drawn p_1. The tool would then output a IFPS prediction for the input parameters.

4.5.4 Identifying the Percentage of Space Parameters Evaluated

The best way to predict the performance would be to evaluate the rendering speed of the scene with every combination of characteristics. In this case, we wouldn't even need to create non-parametric models. But this could require the evaluation of millions of possibilities, ending in way too long computation times (about a year). This performance prediction tool thus evaluates a very

small subset of all those combinations in reasonable time (about half an hour). In the following, we determine the percentage of the number of combinations that our tool needs to evaluate. Given:

- $n_{screen} = \{640\text{x}480, 800\text{x}600, 1024\text{x}768, 1152\text{x}864, 1920\text{x}960\}$ the discrete number of studied screen sizes;
- $n_{texture} = \{x \mid x = 2^i \wedge 1 \leq i \leq 10 \wedge x \in \mathbb{N}\}$ the set of studied texture sizes;
- $n_{vertices} = \{x \mid 1 \leq x \leq 1{,}300{,}000 \wedge x \in \mathbb{N}\}$ the set of all possible quantity of points;
- $n_{grid} = \{x^2 \mid 1 \leq x \leq 45 \wedge x \in \mathbb{N}\}$ the set of studied tile grid sizes such that each tile has the same size in OpenGL coordinates.

We generalized the concept of tile-based scenes for any dense scene by removing the tile resolution concept and replacing it by the concentration of any number of vertices lower than 1,300,000 divided by any grid size in n_{grid}. We chose this maximum number of vertices and also this maximum grid size arbitrarily, as it should cover most data volumes in most of the hardware use cases. We can then evaluate the total number of combinations of characteristics influencing the density of points for every studied screen and texture sizes N_{Total} as:

$$N_{Total} = |n_{vertices}| * |n_{grid}| * |n_{screen}| * |n_{texture}| = 2{,}925{,}000{,}000$$

The tool then needs only to test a small fraction of all those combinations to produce adequate performance models. As mentioned earlier, the tool only needs to run two tests of the benchmark to construct adequate models for a fixed screen and texture size. Each test is configured initially to execute the benchmark with nine varying test parameters as show in Table 4-1.

Given:

- N_{Tool} the total number of combinations of characteristics analyzed by the tool, and
- $S_{TrainingSet} = 5760$ the size of any training dataset output by the benchmark for a grid size and tile resolution test with fixed texture and screen size which corresponds to the number of frames rendered for both tests of the benchmark.

We then suppose that each frame rendered during the benchmarking represents one unique combination of those billions and find the number of combinations tested by the tool N_{Tool} as:

$$N_{Tool} = S_{TrainingSet} * n_{screen} * n_{texture} = P_{TrainingSet} * 288{,}000$$

The tool is then guaranteed to train successful models by using only about 0,0098% of the total combinations of characteristics.

4.6 Prediction Error Evaluation

This section presents the experimental setup and also the experimental considerations used to validate the predictive power of the performance model.

4.6.1 Experimental Setup

We used a Nvidia QuadroFX570, a graphic card model which should have consistent performance with the current avionic hardware: it is a low-performance, low-energy consuming, entry level graphic card which is about ten years old and is compatible with older versions of OpenGL. Since OpenGL SC consists of a subset of the full OpenGL API's functions and that this subset of functions is very similar to the OpenGL ES 1.x specification, we worked around the absence of an OpenGL SC driver by using an OpenGL ES 1.x simulator which transforms the application's OpenGL ES 1.x function calls to the current installed drivers which is OpenGL. A meticulous care has been taken to make sure to use only functions available in the current OpenGL SC specification with the exception of one very important method, named *vertex buffer objects*. This method will be available in the next version.

The experiment begins in the training phase with the generation of the performance models as presented in Section 4. The prediction power of those models is then validated during the validation phase for which an industrial scene is benchmarked many times with varying characteristics. Those performances are then compared to the predictions generated by the models. The following sections explains this validation phase in details.

4.6.2 Performance Model Validation

To validate the performance model created, we used a 3D scene from the World CDB representing a hilly location in Quebec, Canada. The scene was subsampled to various degrees to enable the validation of the model at various resolutions. The models were first validated for their interpolation predictive power by comparing the predictions with the validation scene rendered with characteristics similar to the ones used to train the models. The models were then validated for their extrapolation predictive power with the same method but by rendering the validation scene with characteristics untreated during the model training. It is also important to select well those characteristics in order to produce scenes that are not too easy to render. Because, it is easier for

the models to predict the maximum V-synced FPS, the validation scene characteristics should be selected in a way that makes the rendering operations generate various percentages of frames drawn with maximum IFPS. We analysed the influence of having about 0%, 50% or 100% of frames in the dataset drawn with maximum IFPS. As with the synthetic scenes, the whole validation scene is loaded in graphic memory prior to rendering the scene. Therefore, the loading time is not taken into account during the FPS calculation.

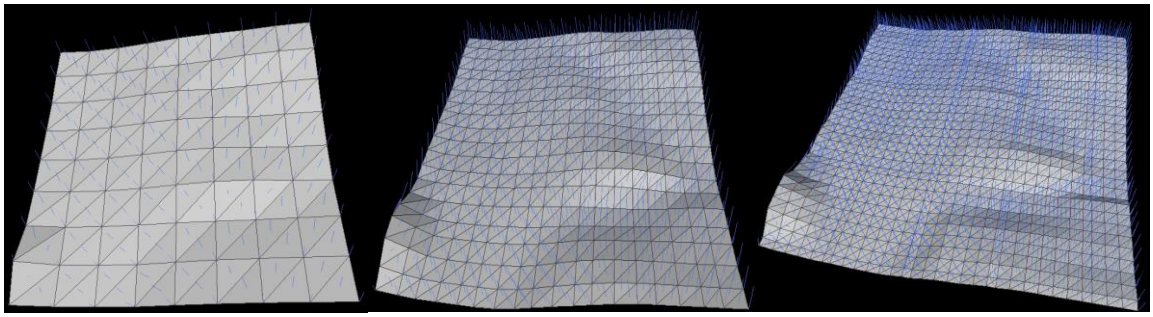

Figure 4-6: Mesh and normals of one tile of the validation scene sampled at a resolution of 9x9 (left), 19x19 (middle) and 31x31 (right) shown before applying the randomized texture.

To validate a model, the benchmark is executed, but instead of displaying the usual synthetic scene, it renders the one from the World CDB subsampled according to the various parameters shown in Table 4-3. The output of the validation dataset is then smoothed in the same way as the training dataset.

Tableau 4-3: Values of the World CDB scene characteristics

Dataset	Name	Varied parameter	Tile Resolution	Grid Size	% of frames with maximum IFPS
Validation (Cdb scene)	#1	Tile resolution	7;9;13;17;21;25;31;37;45	25	0%
	#2	Grid size	25	7;9;13;17;21;25;31;37;45	44.38
	#3	Tile resolution	7;9;13;17;21;25;31;37;45	17	100%
	#4	Grid size	17	7;9;13;17;21;25;31;37;45	44.51

We then compare the smoothed instantaneous FPS of each frame to the predicted ones with the following metrics.

4.6.3 Metric Choice and Interpretation

The choice of a metric to evaluate the prediction errors is still subject to debate in the literature [Chai and Draxler 2014]. Especially in the case of models generated with machine learning, the error distribution are rarely normal-like and thus more than one metric are commonly used to help understand and quantify the central tendency of prediction errors. We use the mean absolute prediction error *MAE* presented in (9) and also the rooted-mean-squared prediction error *RMSE* presented in (10). To conform to the literature, we also give the *MAE* in terms of relative prediction errors *PE* presented in (11). Because the error distributions will be most likely not normal, we also give the median error value which could yield the most significant central tendency. This last metric contribution in the understanding of the distribution is to indicate that 50% of the prediction errors are lesser than its value.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|\widehat{FPS}_i - FPS_i\right| \tag{11}$$

$$RMSE = \sqrt{\frac{1}{(n-1)}\sum_{i=1}^{n}(\widehat{FPS}_i - FPS_i)^2} \tag{12}$$

$$PE = \frac{1}{n}\sum_{i=1}^{n}\frac{|\widehat{FPS}_i - FPS_i|}{FPS_i} * 100\% \tag{13}$$

Where $\widehat{FPS}_i$ is the i-th prediction, FPS_i is the i-th measured value and n is the number of prediction/measurement pairs.

4.7 Results

Fig. 4-7 to 4-10 show that the prediction errors do not follow a normal distribution, but even though there is some skewness in them, they still retain a half-bell like appearance. The most adequate central tendency metric is thus the median. Furthermore, the artificial neural network has a better prediction than the two other algorithms most of the time, followed closely by random forest. Table 4-4 shows that the gap between the median prediction errors of both of these algorithms never exceeds 1 FPS. On the other hand, the MART method performed poorly on all datasets with a gap of up to about 12 FPS. Also, the performance models made quite good predictions in an interpolation and extrapolation context, as shown by the central tendencies of errors of all validation datasets confounded. The central tendency gap between those two sets never exceeds 4 FPS in this experiment. By analyzing the maximum absolute prediction error of most

datasets, we see that there is a small chance that a very high prediction error is produced. These high errors can be as high as 43 FPS in the third dataset. Even though the general accuracy of the model is pretty good, the precision can be improved. Hopefully, the mode of each error distribution is always the first bin (range of values) of the histogram, which means that the highest probability of a prediction error is always the lowest error.

Tableau 4-4: Central tendencies of the prediction error distributions for each machine learning algorithm and for each validation dataset.

Validation Dataset Name	Supervised Learning Algorithm	RMSE (FPS)	Relative Prediction Error (%)	Mean Absolute Prediction Error (FPS)	Median Absolute Prediction Error (FPS)
#1	Random Forest	2.71	2.12	1.36	0.45
	MART	9.37	10.15	6.85	4.86
	Neural Net	2.29	1.99	1.26	0.50
#2	Random Forest	10.87	8.74	5.85	2.06
	MART	16.53	17.65	12.16	10.39
	Neural Net	7.90	8.50	5.16	3.51
#3	Random Forest	5.94	5.98	4.10	2.79
	MART	9.99	10.60	7.51	6.15
	Neural Net	2.97	3.03	2.39	2.13
#4	Random Forest	15.80	9.12	10.94	3.98
	MART	30.20	18.78	22.53	15.02
	Neural Net	4.02	2.58	3.10	3.01

Fig. 4-7 to 4-10 presents the error distribution of each non-parametric performance model for the four validation scenes.

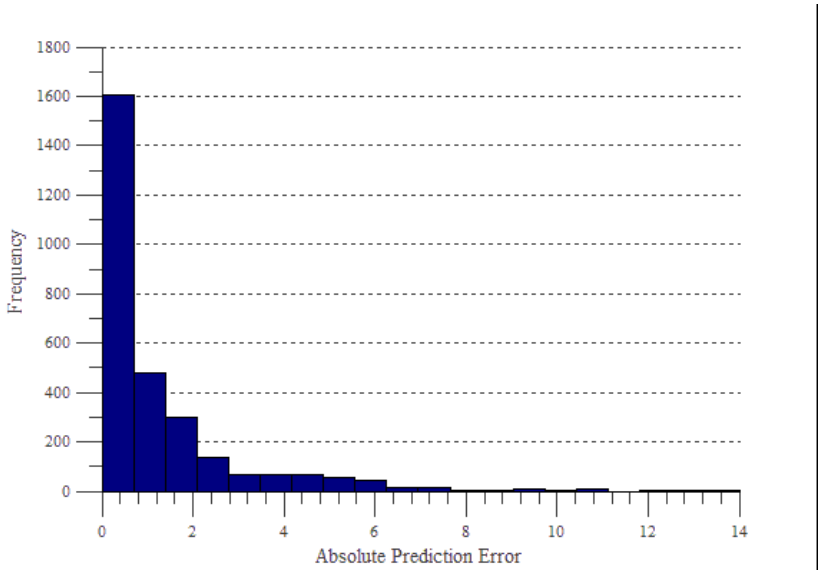

Figure 4-7: Prediction error distribution of the validation dataset #1 for the artificial neural network.

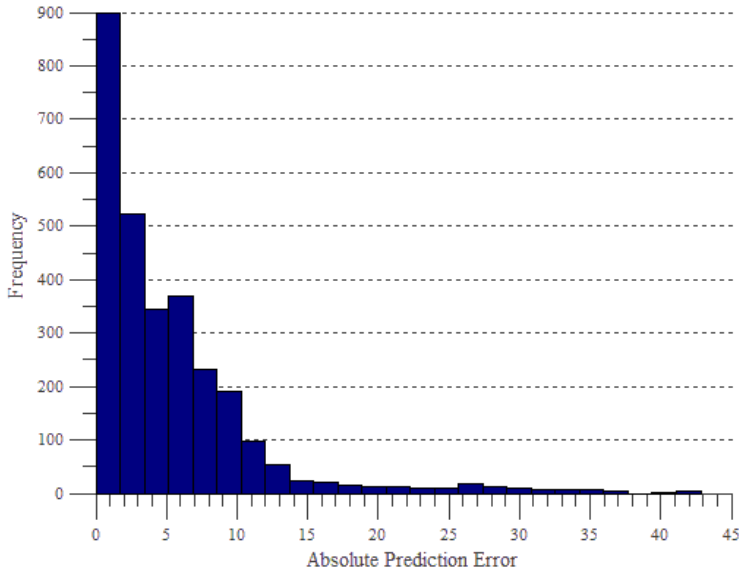

Figure 4-8: Prediction error distribution of the validation dataset #2 for the artificial neural network

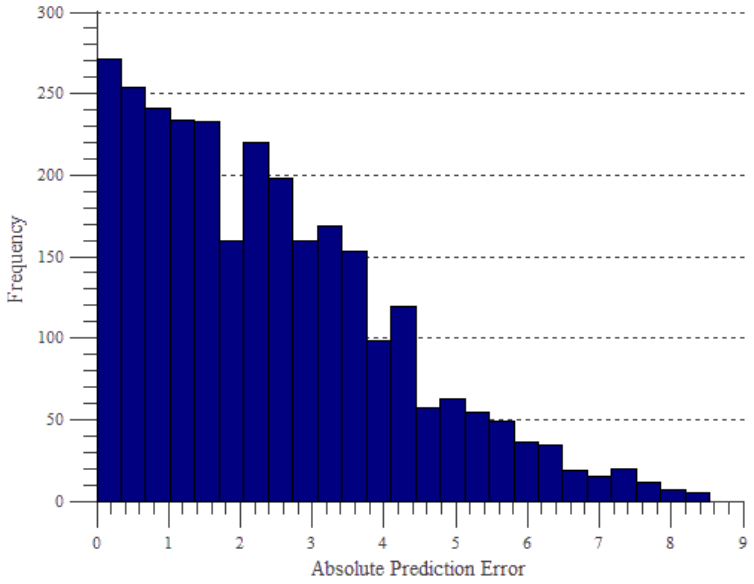

Figure 4-9: Prediction error distribution of the validation dataset #3 for the artificial neural network.

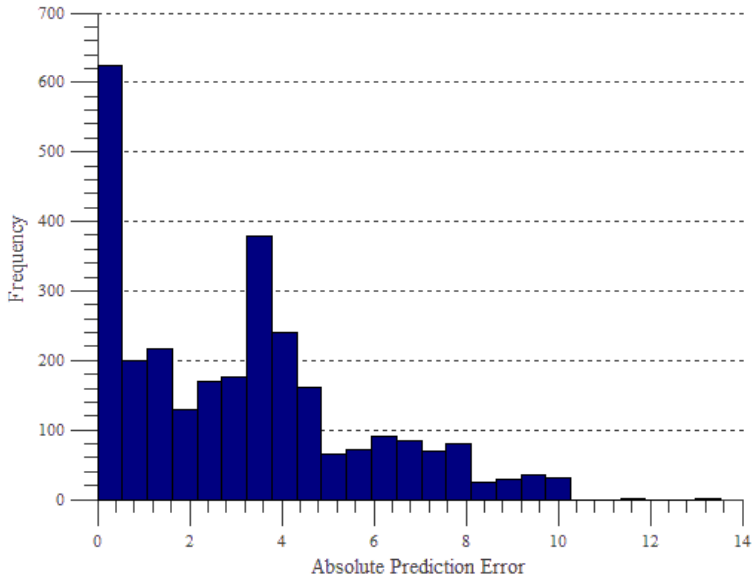

Figure 4-10: Prediction error distribution of the validation dataset #4 for the artificial neural network.

4.8 Discussion

Because there is a very small chance (less than 1%) that a prediction might have a high error, the final prediction offered by the tool for a combination of characteristics should be a weighted average or a robust local regression of a small set of performance with similar scene characteristics, in order to help reduce the influence of these prediction outliers. Also, the scene

used to train and validate our models are all dense, thus our experiment cannot imply any significance for sparse scenes. But, as most graphical application in an avionics context uses dense scenes this should not be a major issue. We also do not use fog effects in our tools which is a feature that could be used in a real industrial context, as this feature will be part of the next release of *OpenGL SC*. On the other hand, because of the high costs of avionics hardware, we had to abstract desktop graphic hardware behind an *OpenGL ES 1.x* environment to simulate an *OpenGL SC* environment which might weaken the correlation of our results to the ones that would be obtained with real avionics hardware. We also used only one validation scene subsampled into four distinct scenes. It could be of interest to reproduce the experiment with a bigger dataset of dense scenes. Considering our results as reproducible, then the prediction errors made by our tool would be low enough for industrial use. Otherwise, we confirm that the central tendency of our prediction error distributions are similar to the ones presented in the literature, when these methods are applied to other kind of hardware or software performance prediction.

4.9 Conclusion

We have presented a set of tools that enable performance benchmarking and prediction in an avionics context. These were missing or not offered in the literature. We believe that avionics system designers and architects could benefit from these tools as none other are available in the literature. Also, the performance prediction errors were shown to be reasonably low, thus demonstrating the efficacy of our method. Future work will include the development of a performance-correct simulator for avionics graphics hardware and also the addition of other scene characteristics like fog effects or antialiasing in the performance models. Also, it is of interest to evaluate the possibility in creating a single performance model including all scene parameters. The generalization of such a model might be the biggest challenge.

4.10 Acknowledments

Special thanks to CAE Inc. for having provided us with experimental material, industrial 3D scenes from the World CDB. We would also like to thank our other partners, contributors and sponsors CMC Electronics, CRIAQ, MITACS and NSERC.

4.11 References

AMD 2015. GPU PerfStudio. http://developer.amd.com/tools-and-sdks/graphics-development/gpu-perfstudio/

Baghsorkhi, S.S., Delahaye, M., Patel, S.J., Gropp, W.D. AND Hwu, W.-m.W. 2010. An Adaptive Performance Modeling Tool for GPU Architectures. *SIGPLAN Not. 45*, 105-114.

Boyer, M., Meng, J. AND Kumaran, K. 2013. Improving GPU Performance Prediction with Data Transfer Modeling. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, 1097-1106.

Chai, T. AND Draxler, R.R. 2014. Root mean square error (RMSE) or mean absolute error (MAE)? Arguments against avoiding RMSE in the literature. *Geoscientific Model Development 7*, 1247-1250.

Clevelan, W.S. AND Devlin, S.J. 1988. Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting. *Journal of the American Statistical Association 83*, 596-610.

Cole, P. 2005. OpenGL ES SC - open standard embedded graphics API for safety critical applications. In *Digital Avionics Systems Conference, 2005. DASC 2005. The 24th*, 8.

Corporation, S.P.E. 2007. What is This Thing Called "SPECviewperf®"? https://www.spec.org/gwpg/gpc.static/whatis_vp8.html

Dutton, M.a.K., D 2010. The challenges of graphics processing in the avionics industry. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, 5.A.1-1-5.A.1-9.

Hilderman, V.a.B., Len 2007. *Avionics Certification: A Complete Guide to DO-178 (Software), DO-254 (Hardware)*. Avionics Communications Inc., USA.

Hong, S. AND Kim, H. 2009. An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness. *SIGARCH Comput. Archit. New 37*, 152-163.

Joseph, P.J., Vaswani, K. AND Thazhuthaveetil, M.J. 2006. A Predictive Performance Model for Superscalar Processors. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, 161-170.

Joseph, P.J., Vaswani, K. AND Thazhuthaveetil, M.J. 2006. Construction and use of linear regression models for processor performance analysis In *International Symposium on High-Performance Computer Architecture - HPCA*, 99-108.

Kanter, D. 2011. "Predicting AMD and Nvidia GPU Performance", Real World Technologies. http://www.realworldtech.com/amd-nvidia-gpu-performance/

Khronos 2009. OpenGL SC Safety Critical Profile. https://www.khronos.org/openglsc/

Lee, B.C. AND Brooks, D.M. 2006. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. *SIGPLAN Not. 41*, 185-194.

Lee, B.C. AND Brooks, D.M. 2007. Illustrative Design Space Studies with Microarchitectural Regression Models. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, 340-351.

Lee, B.C., Brooks, D.M., de Supinski, B.R., Schulz, M., Singh, K. AND McKee, S.A. 2007. Methods of Inference and Learning for Performance Modeling of Parallel Applications. In *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Jose, California, USA, 249-258.

Li, B., Peng, L. AND Ramadass, B. 2009. Accurate and efficient processor performance prediction via regression tree based modeling. *Journal of Systems Architecture 55*, 457-467.

Marin, G. AND Mellor-Crummey, J. 2004. Cross-architecture Performance Predictions for Scientific Applications Using Parameterized Models. *SIGMETRICS Perform. Eval. Rev. 32*, 2-13.

Nvidia 2015. Nvidia Nsight. http://www.nvidia.com/object/nsight.html

Ould-Ahmed-Vall, E., Woodlee, J., Yount, C., Doshi, K.A. AND Abraham, S. 2007. Using Model Trees for Computer Architecture Performance Analysis of Software Applications. In *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, 116-125.

Presagis 2015. ARINC-661 Widget Creation, Presagis Inc. http://www.presagis.com/solutions/arinc_661_widget_creation/

Rightware Basemark ES 3.0. http://www.rightware.com/benchmarks/basemark-es-3-0/

Sander, P.V.a.N., Diego and Barczak, Joshua 2007. Fast Triangle Reordering for Vertex Locality and Reduced Overdraw. *ACM Trans. Graph. 26*.

Wang, Z.a.Y., Hui and Zhou, Xiuzhi 2014. A Simulation Method of Reconfigurable Airborne Display and Control System. In *Proceedings of the First Symposium on Aviation Maintenance and Management-Volume II*, J. Wang Ed. Springer Berlin Heidelberg, 255-263.

Yao, Z. AND Owens, J.D. 2011. A quantitative performance analysis model for GPU architectures. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, 382-393.

Yoo, R.M., Lee, H., Chow, K. AND Lee, H.-H.S. 2006. Constructing a Non-Linear Model with Neural Networks for Workload Characterization. In *Workload Characterization, 2006 IEEE International Symposium on*, 150-159.

Zhang, Y., Bin Li, Y.H. AND Pen, L. 2011. Performance and Power Analysis of ATI GPU: A Statistical Approach. In *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*, 149-158.

Ïpek, E., McKee, S.A., Caruana, R., de Supinski, B.R. AND Schulz, M. 2006. Efficiently Exploring Architectural Design Spaces via Predictive Modeling. *SIGARCH Comput. Archit. News 34*, 195-206.

CHAPITRE 5 RÉSULTATS COMPLÉMENTAIRES

Étant une fusion de deux travaux de recherche, l'article ne présente pas la contribution apportée dans le cadre de l'OS2. Il est donc nécessaire d'ajouter cette section de résultats complémentaires pour présenter et évaluer ces améliorations.

5.1 Vers une évaluation de performance plus exacte

En guise de rappel, la section 2.1.3. présentait les lacunes de l'outil d'évaluation de performance utilisé pour la récolte des données d'entraînement et de test desquelles on cherche à créer un modèle statistique des performances :

1. Le calcul du nombre de sommets utilisés pour l'affichage de la scène ne tient pas compte de l'élimination des faces arrières et du *z-buffering*.
2. Haut taux moyen d'échec de cache.
3. L'utilisation d'une moyenne arithmétique n'est pas significative, car les distributions de IFPS ne sont pas normales.

De plus, comme on peut lire dans l'article, les données extraites de l'outil d'évaluation de performance sont d'abord lissées à l'aide de la méthode LOESS, puis leur format est modifié à l'aide de la variable *K* avant d'être envoyées dans l'outil de modélisation des performances. Ces deux opérations ne sont pas présentes dans les travaux de l'auteur initial et font partie de la contribution de ce présent travail de recherche.

5.1.1 Calcul du nombre de sommets tenant compte de l'élimination des faces arrières

Lors du calcul du nombre de points utilisés pour l'affichage de la scène de l'outil original, chaque point est d'abord comparé aux plans du parallépipède du tronc de projection pour éliminer ceux qui en sortent. Dans le but de remédier à l'absence de la prise en compte de l'élimination des faces-arrières, l'angle entre la direction de la caméra et la normale associée à chaque point est analysé. Si l'angle est entre 90 et 270 degrés, alors on considère que le sommet fait partie d'une surface qui fait face à la caméra, sinon, le sommet n'est pas pris en compte. Pour ce faire, le pseudovecteur représentant la normale dans l'espace 3D du repère *monde* est d'abord transformé

vers l'espace 3D du repère de la *caméra* à l'aide de la transposée inverse de la matrice de vue. On effectue ensuite le produit scalaire de cette nouvelle normale avec la direction de la caméra (qui dans son propre repère est le vecteur unitaire $\overrightarrow{(0,0,1)}$. Si le résultat du produit scalaire est plus petit que zéro, alors l'angle est considéré comme étant entre 90 et 270 degrés.

$$\overrightarrow{N_{cam}} = ({M_{cam}}^{-1})^T * \overrightarrow{N_{monde}}$$

$cosAngle = \overrightarrow{N_{cam}} \cdot \overrightarrow{(0,0,1)}$ où $\cdot$ représente le produit scalaire.

Si $cosAngle < 0$ alors le sommet est considéré comme étant affiché, sinon comme étant ignoré.

5.1.2 Réduction du taux moyen d'échec de cache

L'algorithme *tipsify* est d'abord appliqué au nuage de points avant de démarrer le chronomètre calculant les temps d'exécution. Le nuage de points est ensuite envoyé vers la carte graphique et le rendu chronométré peut débuter. L'implémentation de l'algorithme provient de la bibliothèque open source *Open Asset Import Library v3.1.1*. [66]. Le choix de cet algorithme est explicable par le fait que son auteur assure une complexité linéaire en fonction du nombre de sommets, ainsi que par le fait que son implémentation est disponible gratuitement dans un contexte académique.

5.1.3 Utilisation du IFPS sans moyenne arithmétique

L'auteur original fait une moyenne arithmétique des IFPS en regroupant d'abord tous ceux qui partagent un même paramètre de tests (c'est-à-dire le même nombre de points total de la scène, le même nombre de tuiles, etc.). L'utilisation d'une moyenne arithmétique n'est en fait pas statistiquement significative puisque la distribution des IFPS n'est pas normale. Les figures 3-1, 3-2, 3-3 et 3-4 montrent la distribution des IFPS de l'ensemble de données utilisé pour entraîner les modèles de performance, d'abord avec les améliorations mentionnées précédemment, puis sans leur utilisation. Les distributions obtenues durant les autres tests possèdent une forme semblable et ne sont pas démontrés dans ce document aux fins de simplification, mais sont disponibles sur demande. Ainsi, un coup d'œil rapide sur ces histogrammes permet clairement de voir qu'il ne s'agit pas d'une courbe gaussienne prouvant ainsi la non-normalité. L'utilisation de FPS moyen tel que suggéré par la deuxième étape d'évaluation des performances de l'auteur original n'est donc pas pris en compte durant la création de modèles de performance.

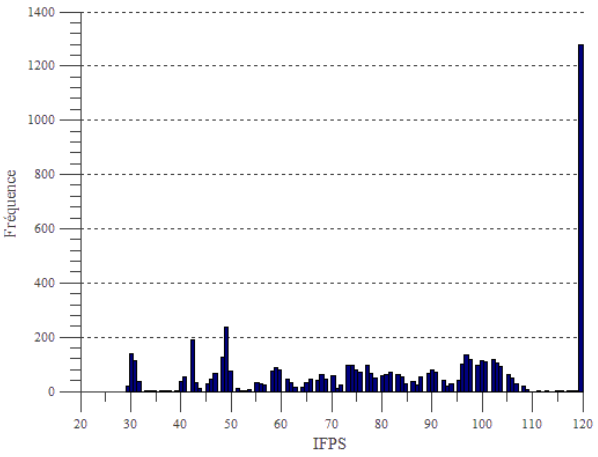

Figure 5-1: Distribution des IFPS pour l'ensemble de données d'entraînement – **avec** contribution de l'OS2.

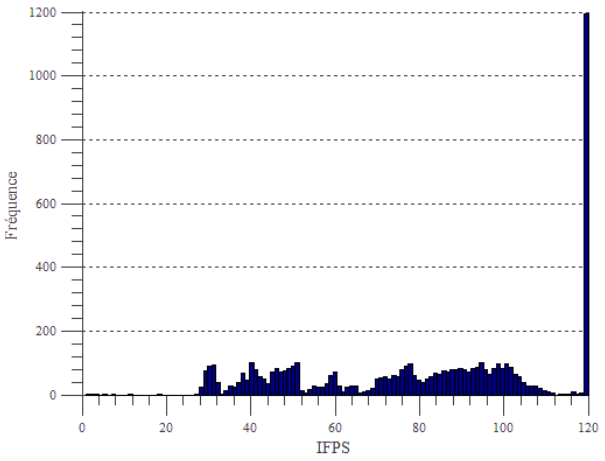

Figure 5-2: Distribution des IFPS pour l'ensemble de données d'entraînement – **sans** contribution de l'OS2.

5.2 Évaluation quantitative des améliorations

Malgré les améliorations portées à l'outil faisant la récolte des temps d'exécution, il est intéressant d'étudier leur influence sur la construction des modèles de prédiction. Le tableau 5-1 présente le pourcentage d'erreur, l'erreur moyenne absolue, la médiane d'erreur absolue et l'écart quadratique moyen de la distribution d'erreurs de prédiction entre l'évaluation de performance d'abord sans les améliorations présentées précédemment, puis avec toutes les améliorations proposées ainsi que l'opération de lissage et de transformation du format des données. Le pouvoir prédictif des modèles de performance sont évalués dans deux cas d'intérêt traités par les scènes de validation : interpolation et extrapolation. Dans le cadre de l'interpolation, on cherche à connaître le pouvoir prédictif des performances pour l'affichage d'une scène réelle possédant des paramètres (nombres de points, taille d'écran, etc.) similaires à ceux de la scène synthétique ayant créé le modèle. Dans le cadre de l'extrapolation, on cherche à connaître le pouvoir prédictif des performances pour l'affichage d'une scène réelle possédant des paramètres distincts n'ayant pas été utilisés lors de la création du modèle. Les résultats présentés de la prédiction de performance sont ceux associée à une carte graphique Nvidia QuadroFX570.

Tableau 5-1: Mesures de tendances centrales des erreurs de prédiction pour les tests d'évaluation de performance de la carte Nvidia QuadroFX570 – Comparaison entre les méthodes avec et sans améliorations pour chaque scènes de validation et pour chaque algorithme d'apprentissage.

Scène de validation	Algorithme d'apprentis-sage	Méthode d'évaluation	Écart quadratique moyen (FPS)	Erreur relative (%)	Erreur moyenne absolue (FPS)	Médiane d'erreur absolue (FPS)
#1	Random Forest	Selon [6]	11,85	17,04	9,70	8,72
		Contribution OS2	2.71	2.12	1.36	0.45
	MART	Selon [6]	11,84	17,02	9,65	8,69
		Contribution OS2	9.37	10.15	6.85	4.86
	Réseaux neuronaux	Selon [6]	11,91	17,03	9,78	9,33
		Contribution OS2	2.29	1.99	1.26	0.50
#2	Random Forest	Selon [6]	12,79	11,91	10,13	9,09
		Contribution OS2	10.87	8.74	5.85	2.06
	MART	Selon [6]	12,97	11,30	9,95	8,59
		Contribution OS2	16.53	17.65	12.16	10.39
	Réseaux neuronaux	Selon [6]	13,03	11,42	10,21	9,25
		Contribution OS2	7.90	8.50	5.16	3.51
#3	Random Forest	Selon [6]	13,16	17,18	9,60	7,81
		Contribution OS2	5.94	5.98	4.10	2.79
	MART	Selon [6]	13,20	17,28	9,66	8,06
		Contribution OS2	9.98	10.60	7.51	6.15
	Réseaux neuronaux	Selon [6]	12,79	16,74	9,13	6,93
		Contribution OS2	2.97	3.03	2.39	2.13
#4	Random Forest	Selon [6]	33,45	29,59	29,19	30,10
		Contribution OS2	15.80	9.12	10.94	3.98
	MART	Selon [6]	34,30	30,24	29,79	31,13
		Contribution OS2	30.20	18.78	22.53	15.02
	Réseaux neuronaux	Selon [6]	33,78	30,24	29,89	30,53
		Contribution OS2	4.02	2.59	3.10	3.01

On remarque d'abord que les améliorations présentées par la contribution de l'OS2 permettent de réduire grandement les erreurs de prédiction, dépassant souvent une réduction de la moitié de la valeur originale. La méthode MART toutefois fonctionne de façon similairement mauvaise avec et sans les améliorations de l'OS2. Comme mentionné dans l'article, les réseaux neuronaux performent en général mieux que les deux autres méthodes. Les figures 5-5 à 5-12 présentent donc la différence entre les distributions d'erreurs des modèles entraînés avec les réseaux neuronaux avec et sans les améliorations de l'OS2 pour les quatre scènes de validation. À noter que les figures 5-5, 5-7, 5-9 et 5-12 sont extraites de l'article et maintiennent la langue utilisée durant la rédaction de ce dernier.

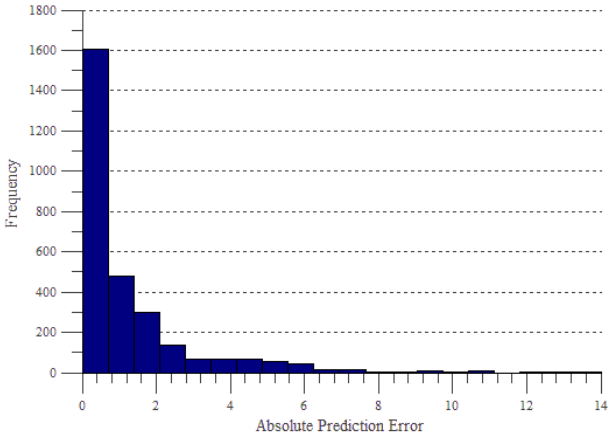

Figure 5-3: Distribution d'erreur de prédiction pour la scène de validation #1 pour les modèles entraînés par réseaux neuronaux **avec** la contribution de l'OS2.

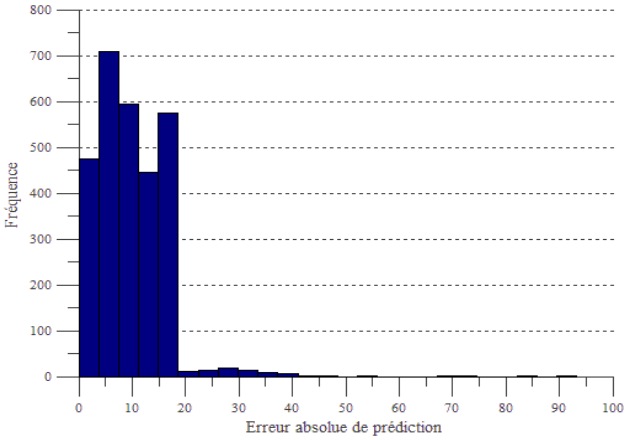

Figure 5-4: Distribution d'erreur de prédiction pour la scène de validation #1 pour les modèles entraînés par réseaux neuronaux **sans** la contribution de l'OS2.

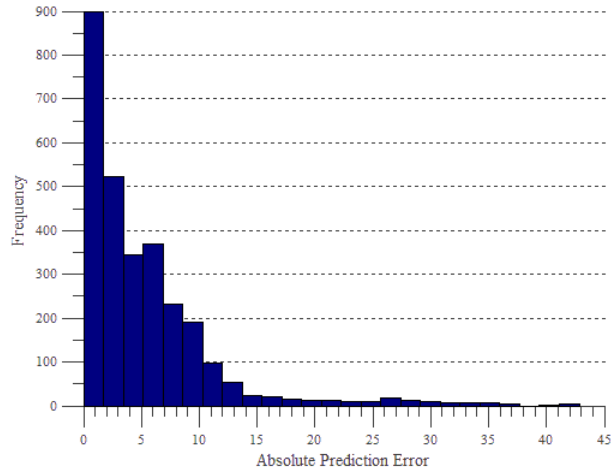

Figure 5-5: Distribution d'erreur de prédiction pour la scène de validation #2 pour les modèles entraînés par réseaux neuronaux **avec** la contribution de l'OS2.

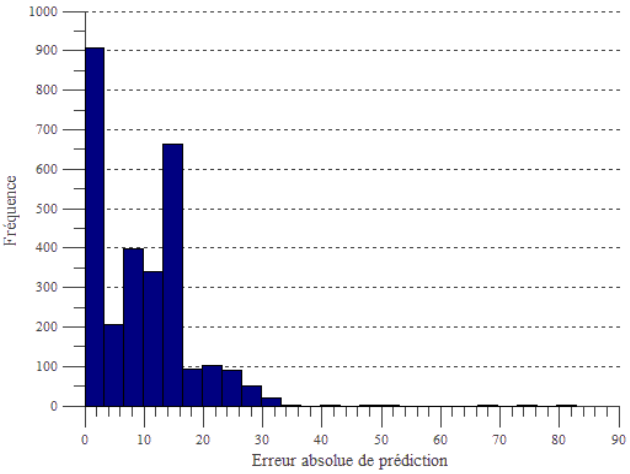

Figure 5-6: Distribution d'erreur de prédiction pour la scène de validation #2 pour les modèles entraînés par réseaux neuronaux **sans** la contribution de l'OS2.

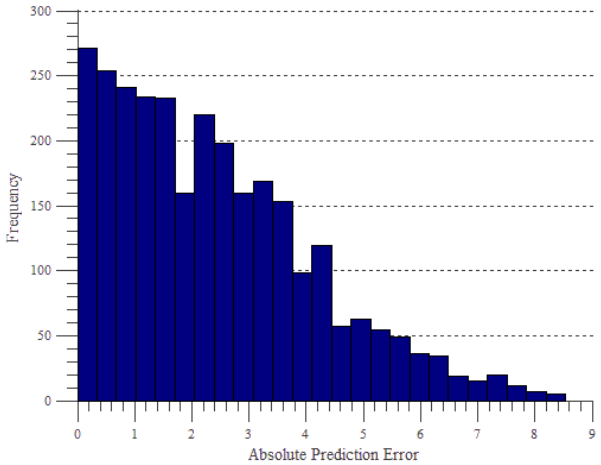

Figure 5-7: Distribution d'erreur de prédiction pour la scène de validation #3 pour les modèles entraînés par réseaux neuronaux **avec** la contribution de l'OS2.

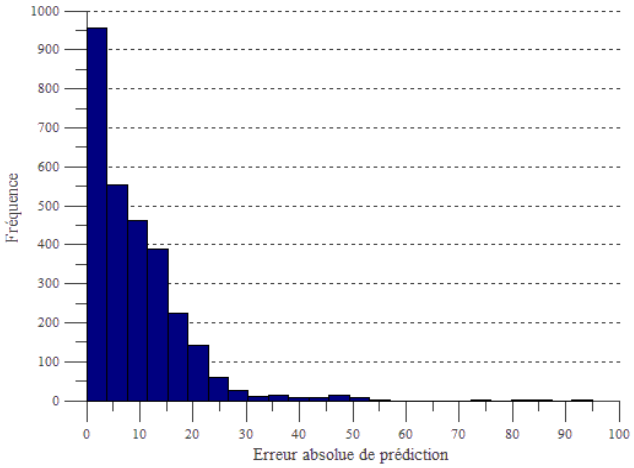

Figure 5-8: Distribution d'erreur de prédiction pour la scène de validation #3 pour les modèles entraînés par réseaux neuronaux **sans** la contribution de l'OS2.

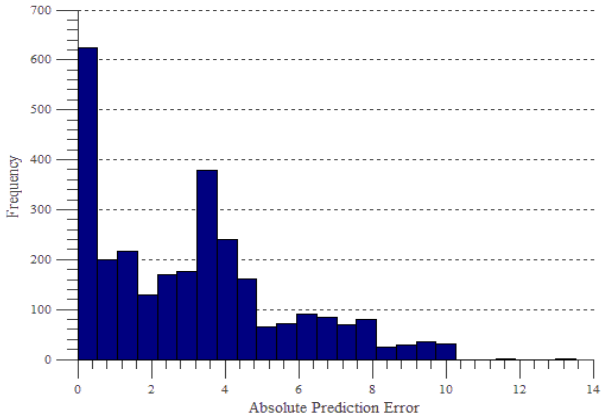

Figure 5-9 : Distribution d'erreur de prédiction pour la scène de validation #4 pour les modèles entraînés par réseaux neuronaux **avec** la contribution de l'OS2.

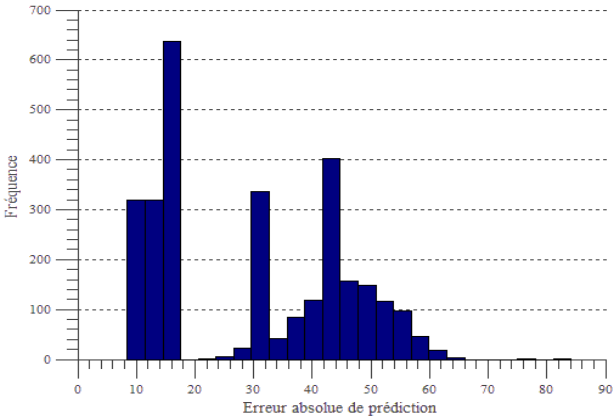

Figure 5-10 : Distribution d'erreur de prédiction pour la scène de validation #4 pour les modèles entraînés par réseaux neuronaux **sans** la contribution de l'OS2.

Selon les méthodes d'évaluation de prédictions présentées dans la section 2.2.4, l'étude quantitative de l'exactitude se fait soit à l'aide de l'écart quadratique moyen (RMSE), de l'erreur moyenne absolue (AE) et/ou relative ou encore à l'aide de la médiane dépendamment du type de distribution d'erreur. Comme on peut l'observer dans les figures 5-5 à 5-12, les distributions sont loin d'être normales. Le RMSE perd donc sa significativité statistique. L'exactitude est alors interprétable d'abord avec beaucoup de significativité comme étant inversement proportionnelle à la médiane puis à l'erreur moyenne et relative de prédiction. Le tableau 5-1 montre que la méthode originale d'évaluation des performances selon [7] permet la création de modèles beaucoup moins exacts pour tout algorithme d'apprentissage confondu, car la moyenne et la médiane d'erreur sont beaucoup plus grandes.

5.3 Discussion partielle reliée à l'OS2

On considère donc l'hypothèse H1 comme confirmée pour cette méthode puisqu'il a été prouvé expérimentalement que les erreurs de prédiction peuvent être réduites à des valeurs similaires à celles présentes dans la littérature. Les valeurs de tendances centrales obtenues initialement avec l'outil [6] démontrent que les données sont initialement inutilisables, mais la contribution de l'OS2 permet d'en extraire des données entraînant des erreurs de prédictions à tendance centrale beaucoup moins grande. En ne s'intéressant qu'aux réseaux neuronaux artificiels, on peut observer des écarts d'erreurs de prédiction de plus de 25 FPS dans la quatrième scène de validation entre l'outil original et l'outil amélioré dans le cadre de l'OS2. La comparaison de cette tendance par rapport à la littérature est présentée dans le chapitre 6. L'avantage de cette contribution est donc clairement relié à l'amélioration de l'exactitude de l'outil, toutefois son désavantage est qu'elle est plus difficile à implémenter.

CHAPITRE 6 DISCUSSION GÉNÉRALE

L'objectif général OG1 est de prouver la possibilité de porter certaines techniques de prédiction de performance de matériel informatique d'un contexte de développement conventionnel vers un contexte avionique. Pour considérer cet objectif comme atteint, l'hypothèse H1 doit être satisfaite : « La prédiction de performance de matériel graphique en avionique est considéré comme significativement utilisable si l'erreur de prédiction est similaire à celle des autres méthodes dans la littérature appliquées à tout domaine confondu. ». De plus, trois objectifs spécifiques découlent de l'objectif général : recension et analyse des méthodes de prédictions du contexte de développement conventionnel ayant le potentiel d'être utilisées dans un contexte avionique (OS1), recension et analyse des méthodes d'évaluation de performances dans un contexte avionique (OS2) et évaluation de la possibilité de généraliser le matériel graphique avionique en n'utilisant qu'un sous-ensemble de fonctionnalités du matériel graphique conventionnel afin d'éviter l'achat de matériel très cher (OS3).

D'abord, nous considérons l'hypothèse H1 comme confirmée, puisqu'il a été prouvé expérimentalement que l'outil de prédiction de performance réalisé dans le cadre de l'OG1 atteint des performances similaires à celles de littérature. L'erreur relative des réseaux neuronaux artificiels atteint durant l'expérience varie entre 1,99% et 8,5%, valeurs très similaires à celles indiquées dans le tableau 2-1 présentant les erreurs relatives de quelques articles de la littérature. L'objectif général #1 est donc considéré comme atteint. Les avantages et désavantages de l'outil développé sont présentés dans l'article, mais repris ici en français. D'abord, la légère imprécision de l'outil nécessite qu'une prédiction soit faite à l'aide d'un ensemble de prédictions pour réduire l'influence de la très faible chance d'obtenir une grande erreur de prédiction. Puis, notre outil utilise des scènes denses signifiant que la concentration de points est relativement uniforme dans toute la scène. Bien que la majorité des cas d'utilisation d'application graphique avionique utilise ce genre de scène, notre outil serait difficilement adaptable à des scènes clairsemées dont la concentration de sommets est peu uniforme. Ensuite, l'utilisation d'un simulateur d'environnement *OpenGL SC* permet d'utiliser les outils d'évaluation et de prédiction de performances sur du matériel graphique conventionnel à moindre coût, tout en restant dans un contexte avionique. L'avantage de cette méthode est donc la réduction des coûts reliés à ce projet de recherche, ainsi qu'à de potentiels clients de l'outil. Toutefois, cela réduit la capacité de tirer des conclusions par rapport aux

performances de l'outil pour du matériel graphique avionique véritable. Sachant ce défaut, du matériel graphique conventionnel vieux de plus de dix ans est utilisé durant l'expérimentation afin de réduire cet écart. D'autre part, l'utilisation d'un arbre de modèles prend plus de place en mémoire et nécessite un plus long temps d'entraînement que l'utilisation d'un simple modèle. Toutefois, l'utilisation d'un simple modèle contenant autant de variables dépendantes que de caractéristiques de scènes serait difficile à généraliser. Un autre désavantage de cette méthode est qu'elle n'est pas généralisable pour tout type de matériel graphique et ne fonctionne que pour du matériel à pipeline fixe. Enfin, certains effets graphiques présents dans la prochaine version d'*OpenGL SC* ne sont pas pris en compte par l'outil, tels les effets de brouillard ou l'anticrénelage.

D'autre part, à titre de conclusions par rapport à l'OS1, on peut affirmer avec preuve empirique que les réseaux neuronaux artificiels performent mieux que les forêts d'arbres décisionnelles ou que la méthode MART. Toutefois, les réseaux neuronaux sont assez difficiles à paramétrer et nécessitent une bonne connaissance de leur fonctionnement. Ainsi, en présence d'une personne qualifiée, ce serait la méthode à adopter pour de futures implémentations de l'outil. Toutefois, en absence de personne qualifiée, les forêts d'arbres décisionnelles sont plus faciles à implémenter et permettraient d'atteindre des résultats similaires quoiqu'un peu moins significatifs. Ceci est prouvé par expérimentation et permet donc de conclure l'OS1 comme atteint. De plus, d'autres remarques peuvent être faites quant à la configuration optimale du réseau neuronal pour le problème étudié d'approximation de fonction. D'abord, l'algorithme de rétropropagation Levenberg-Marquardt est particulièrement adéquat en termes de vitesse d'entraînement selon la topologie du réseau utilisé dans le cadre de ce travail. Puis, le seuil d'erreur minimale, qui constitue une des conditions d'arrêt de l'entraînement du réseau, n'est jamais atteint. Cette erreur, mesurée à l'aide de l'équart quadratique moyen et obtenue durant l'entraînement est rarement plus basse que 10 FPS. Cela signifie que le réseau neuronal n'explique pas totalement la variance des données de performance qui lui sont fournies. Toutefois, comme mentionné plus tôt, c'est un comportement attendu puisque l'on connait la haute présence de bruit dans les données d'entraînement, confirmant l'importance d'effectuer l'opération de lissage de données en prétraitement. Enfin, l'OS2 est démontré comme atteint dans la section 5.3, puis nous considérons l'OS3 comme partiellement atteint puisque les erreurs de prédictions de l'outil sont faibles, malgré l'utilisation d'un simulateur d'environnement *OpenGL SC*. Toutefois, il est difficile de conclure avec certitude qu'il est possible d'atteindre ces mêmes résultats dans un réel environnement avionique.

CHAPITRE 7 CONCLUSION ET RECOMMANDATIONS

Cet ouvrage présente les travaux de recherches reliés au transfert de connaissances de la prédiction de performance de matériel graphique, du domaine de développement embarqué ou pour ordinateur de bureau, vers le domaine de l'avionique. Une nouvelle méthode est présentée permettant d'effectuer de la prédiction de performance pour du matériel graphique avionique. Aucune autre méthode n'existe dans la littérature pour ce problème au niveau du domaine de l'avionique. Toutefois, les résultats obtenus sont similaires à ceux d'autres méthodes existantes pour le domaine d'ordinateur de bureau ou embarqué. Les travaux futurs porteront sur l'ajout de nouvelles fonctionnalités présentes dans la prochaine version d*'OpenGL SC*, tels les effets de brouillard ou l'anticrénelage. De plus, il est intéressant d'orienter l'utilisation de cette méthode de prédiction de performance vers un simulateur de matériel graphique avionique. Cet outil ajouterait une latence artificielle qui permettrait à des développeurs d'exécuter leur logiciel graphique sur leur station de travail possédant du matériel conventionnel, tout en obtenant un avant-goût des performances réelles qu'atteindrait leur logiciel sur le matériel avionique du système final. Il serait aussi intéressant de généraliser davantage les modèles de performance afin d'émettre des prédictions à partir d'un seul modèle contenant toutes les caractéristiques de scènes. Cela permettrait de réduire la taille prise en mémoire en remplaçant l'arbre de modèles par un unique modèle ainsi que de réduire le temps d'entraînement.

RÉFÉRENCES

[1] V. a. B. Hilderman, Len, *Avionics Certification: A Complete Guide to DO-178 (Software), DO-254 (Hardware)*, 1st ed. USA: Avionics Communications Inc., 2007.

[2] M. a. K. Dutton, D, "The challenges of graphics processing in the avionics industry," in *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, 2010, pp. 5.A.1-1-5.A.1-9.

[3] R. Moller, "State-of-the-Art 3D Graphics for Embedded Systems," in *Devices, Circuits and Systems, Proceedings of the 6th International Caribbean Conference on*, ed, 2006, pp. 339-343.

[4] Z. a. Y. Wang, Hui and Zhou, Xiuzhi, "A Simulation Method of Reconfigurable Airborne Display and Control System," in *Proceedings of the First Symposium on Aviation Maintenance and Management-Volume II*. vol. 297, J. Wang, Ed., ed: Springer Berlin Heidelberg, 2014, pp. 255-263.

[5] (9 Janvier). *ARINC-661 Widget Creation, Presagis Inc.* Available: http://www.presagis.com/solutions/arinc_661_widget_creation/

[6] V. Legault, "Méthodologie expérimentale pour évaluer les caractéristiques des plateformes graphiques avioniques," M. Sc. A., Département de génie informatique et logiciel, École Polytechnique de Montréal, Montréal, Canada, 2014.

[7] (2009). *OpenGL SC Safety Critical Profile*. Available: https://www.khronos.org/openglsc/

[8] P. Cole, "OpenGL ES SC - open standard embedded graphics API for safety critical applications," in *Digital Avionics Systems Conference, 2005. DASC 2005. The 24th*, 2005, p. 8.

[9] Rightware. (Janvier 2015). *Basemark ES 3.0*. Available: http://www.rightware.com/benchmarks/basemark-es-3-0/

[10] S. P. E. Corporation. (2007, Janvier 2015). *What is This Thing Called "SPECviewperf®"?* Available: https://www.spec.org/gwpg/gpc.static/whatis_vp8.html

[11] S. K. a. D. Manocha, "Hierarchical back-face culling," ed. Chapel Hill, NC, USA: Tech. Rep, 1996.

[12] H. Z. a. K. E. H., "Fast backface culling using normal masks," ed: SI3D, 1997, pp. 103-106.

[13] J. Gregory, "The Rendering Pipeline," in *Game Engine Architecture*, C. Press, Ed., 2nd Edition ed, 2014.

[14] D. M. H. Zhang, T. Hudson, and K.E. Hoff, "Visibility Culling Using Hierarchical Occlusion Maps " in *Proc. ACM SIGGRAPH '97* 1997, pp. 77-88.

[15] V. H. J. Bittner, and P. Slavik, "Hierarchical Visibility Culling with Occlusion Trees " in *Proc. Computer Graphics Int'l*, 1998, pp. 207-219.

[16] I. P. a. S. Tzafestas, "Occlusion culling algorithms: A comprehensive survey " *J. Intell. Robotics Syst*, vol. 35, pp. 123-156, 2002.

[17] H.-Y. K. a. C.-H. Y. a. L.-S. Kim, "A Memory-Efficient Unified Early Z-Test," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, pp. 1286-1294, 2011.

[18] D. K. C.-H. Yu, and L.-S. Ki, "An Area Efficient Early Z-Test Method for 3D Graphics Rendering Hardwar " *IEEE Trans.Circuits and Systems I* vol. 55, pp. 1929-1938, 2008.

[19] C.-L. W. Y.M. Tsao, S.-Y. Chien, and L.-G. Chen, "Adaptive Tile Depth Filter for the Depth Buffer Bandwidth Minimization in the Low Power Graphics Systems " in *Proc. IEEE Int'l Symp. Circuits and Systems* 2006, pp. 5023-5026.

[20] H. Hoppe, "Optimization of Mesh Locality for Transparent Vertex Caching," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 1999, pp. 269-276.

[21] D. N. a. J. B. a. P. V. Sander, "Triangle Order Optimization for Graphics Hardware Computation Culling " in *Symposium on Interactive 3D Graphics and Games*, 2006.

[22] G. a. Y. Lin, T.P.-Y., "An improved vertex caching scheme for 3D mesh rendering," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 12, pp. 640-648, 2006.

[23] S.-E. a. L. Yoon, Peter and Pascucci, Valerio and Manocha, Dinesh, "Cache-oblivious Mesh Layouts," *ACM Trans. Graph.*, vol. 24, pp. 886-893, 2005.

[24] P. V. a. N. Sander, Diego and Barczak, Joshua, "Fast Triangle Reordering for Vertex Locality and Reduced Overdraw," *ACM Trans. Graph.*, vol. 26, 2007.

[25] Nvidia. (2015, 3 Mars). *Nvidia Nsight*. Available: http://www.nvidia.com/object/nsight.html

[26] AMD. (2015, 3 Mars). *GPU PerfStudio*. Available: http://developer.amd.com/tools-and-sdks/graphics-development/gpu-perfstudio/

[27] D. Kanter. (2011, 19 Janvier). *"Predicting AMD and Nvidia GPU Performance", Real World Technologies*. Available: http://www.realworldtech.com/amd-nvidia-gpu-performance/

[28] Z. Yao and J. D. Owens, "A quantitative performance analysis model for GPU architectures," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, 2011, pp. 382-393.

[29] S. Hong and H. Kim, "An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness," *SIGARCH Comput. Archit. New*, vol. 37, pp. 152-163, 2009.

[30] L. Weiguo, W. Muller-Wittig, and B. Schmidt, "Performance Predictions for General-Purpose Computation on GPUs," in *ICPP 2007. International Conference on*, 2007, p. 50.

[31] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W.-m. W. Hwu, "An Adaptive Performance Modeling Tool for GPU Architectures," *SIGPLAN Not.*, vol. 45, pp. 105-114, 2010.

[32] J. Stratton, S. Stone, and W.-m. Hwu, "MCUDA: An Efficient Implementation of CUDA Kernels for Multi-core CPUs," in *Languages and Compilers for Parallel Computing*. vol. 5335, ed: Springer Berlin Heidelberg, 2008.

[33] S. Collange, M. Daumas, D. Defour, and D. Parello, "Barra: A Parallel Functional Simulator for GPGPU," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, 2010, pp. 351-360.

[34] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, Boston, MA, USA, 2009, pp. 163-174.

[35] A. Kerr, G. Diamos, and S. Yalamanchili, "Modeling GPU-CPU Workloads and Systems," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, Pittsburgh, Pennsylvania, USA, 2010, pp. 31-42.

[36] L. Eeckhout, R. Sundareswara, J. J. Yi, D. J. Lilja, and P. Schrater, "Accurate statistical approaches for generating representative workload compositions," in *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, 2005, pp. 56-66.

[37] S. Che and K. Skadron, "BenchFriend: Correlating the performance of GPU benchmarks," *International Journal of High Performance Computing Applications*, vol. 28, pp. 238-250, 2014.

[38] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive Performance and Scalability Modeling of a Large-scale Application," in *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, Denver, Colorado, USA, 2001, p. 37.

[39] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A Framework for Performance Modeling and Prediction," in *Supercomputing, ACM/IEEE 2002 Conference*, 2002, p. 21.

[40] L. T. Yang, X. Ma, and F. Mueller, "Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, 2005, p. 40.

[41] B. C. Lee and D. M. Brooks, "Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction," *SIGPLAN Not.*, vol. 41, pp. 185-194, 2006.

[42] B. C. Lee and D. M. Brooks, "Illustrative Design Space Studies with Microarchitectural Regression Models," in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, 2007, pp. 340-351.

[43] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "Construction and use of linear regression models for processor performance analysis " in *International Symposium on High-Performance Computer Architecture - HPCA*, 2006, pp. 99-108.

[44] G. Marin and J. Mellor-Crummey, "Cross-architecture Performance Predictions for Scientific Applications Using Parameterized Models," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, pp. 2-13, 2004.

[45] M. Faerman, A. Su, R. Wolski, and F. Berman, "Adaptive Performance Prediction for Distributed Data-intensive Applications," in *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, Portland, Oregon, USA, 1999.

[46] Collectif. (2015). *Machine Learning*. Available: http://en.wikipedia.org/wiki/Machine_learning

[47] Collectif. (2015). *Decision Tree Learning*. Available: http://en.wikipedia.org/wiki/Decision_tree_learning

[48] Collectif. (2015). *Random Forest*. Available: http://en.wikipedia.org/wiki/Random_forest

[49] Collectif. (2015). *Gradient Boosting*. Available: http://en.wikipedia.org/wiki/Gradient_boosting#Gradient_tree_boosting

[50] B. Li, L. Peng, and B. Ramadass, "Accurate and efficient processor performance prediction via regression tree based modeling," *Journal of Systems Architecture*, vol. 55, pp. 457-467, 2009.

[51] E. Ould-Ahmed-Vall, J. Woodlee, C. Yount, K. A. Doshi, and S. Abraham, "Using Model Trees for Computer Architecture Performance Analysis of Software Applications," in *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, 2007, pp. 116-125.

[52] Y. Zhang, Y. H. Bin Li, and L. Pen, "Performance and Power Analysis of ATI GPU: A Statistical Approach," in *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*, 2011, pp. 149-158.

[53] Mathworks. (2015). *Improve Neural Network Generalization and Avoid Overfitting*. Available: http://www.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html

[54] Collectif. (2015). *Artificial neural network*. Available: http://en.wikipedia.org/wiki/Artificial_neural_network

[55] E. Ipek, B. de Supinski, M. Schulz, and S. McKee, "An Approach to Performance Prediction for Parallel Applications," in *Euro-Par 2005 Parallel Processing*. vol. 3648, ed: Springer Berlin Heidelberg, 2005, pp. 196-205.

[56] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, "Methods of Inference and Learning for Performance Modeling of Parallel Applications," in *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Jose, California, USA, 2007, pp. 249-258.

[57] R. M. Yoo, H. Lee, K. Chow, and H.-H. S. Lee, "Constructing a Non-Linear Model with Neural Networks for Workload Characterization," in *Workload Characterization, 2006 IEEE International Symposium on*, 2006, pp. 150-159.

[58] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "A Predictive Performance Model for Superscalar Processors," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, 2006, pp. 161-170.

[59] E. Ïpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz, "Efficiently Exploring Architectural Design Spaces via Predictive Modeling," *SIGARCH Comput. Archit. News*, vol. 34, pp. 195-206, 2006.

[60] S. J. Tarsa, A. P. Kumar, and H. T. Kung, "Workload Prediction for Adaptive Power Scaling Using Deep Learning," in *IC Design & Technology (ICICDT), 2014 IEEE International Conference on*, 2014, pp. 1-5.

[61] V. Zaccaria, G. Palermo, F. Castro, C. Silvano, and G. Mariani, "Multicube Explorer: An Open Source Framework for Design Space Exploration of Chip Multi-Processors," presented at the *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on*, 2010.

[62] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)? Arguments against avoiding RMSE in the literature.," *Geoscientific Model Development*, vol. 7, pp. 1247-1250, 2014.

[63] M. Boyer, J. Meng, and K. Kumaran, "Improving GPU Performance Prediction with Data Transfer Modeling," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, 2013, pp. 1097-1106.

[64] N. Baek and G. J. Baeck, "Design of OpenGL SC emulation library over the desktop OpenGL 1.3," in *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, 2010, pp. 6.D.2-1-6.D.2-8.

[65] (2015). *PowerVR SDK*. Available: http://community.imgtec.com/developers/powervr/graphics-sdk/

[66] A. D. Team. (2009). *Open Asset Import Library*. Available: http://assimp.sourceforge.net/