GENERAREA DE MICRO-OPERATII DE VIRGULA FLOTANTA UTILIZATE IN GRAFICA, IMPLEMENTATE PE FPGA

Ovidiu SICOE

Coordonator:
Prof. Dr. Ing. Mircea POPA

Facultatea de Automatică și Calculatoare
Universitatea Politehnica Timișoara

Timisoara
2018

BUPT

BUPT

Mulțumiri

Aș vrea să mulțumesc îndrumătorului meu, domnului profesor Mircea Popa, pentru susținerea și înțelegerea de care a dat dovadă în toți acești ani. De asemenea, îi mulțumesc pentru ideile propuse și pentru revizuirea constructivă a acestei teze, precum și a rapoartelor de cercetare elaborate pe parcursul studiilor doctorale.

De asemenea, aș vrea să mulțumesc domnului profesor conferențiar Alexandru Amaricăi-Boncalo pentru ideile și clarificările oferite, precum și pentru discuțiile purtate pe subiectele acestei teze care m-au ajutat să avansez în cercetarea efectuată.

Aș vrea să mulțumesc doamnei profesor conferențiar Oana Amaricăi-Boncalo și domnului profesor Marius Marcu, parte a comisiei mele de îndrumare, pentru observațiile pertinente și constructive care m-au ajutat să progresez cu studiul aferent acestei teze, precum si cu structurarea ideilor.

Colegului meu Sergiu Nimară aș vrea să îi mulțumesc pentru împărtășirea experiențelor sale acumulate de-a lungul studiilor doctorale care m-au ajutat să înțeleg pașii pe care îi am de urmat pentru realizarea tuturor actelor de care am avut nevoie în această perioadă.

De asemenea, aș vrea să mulțumesc părinților pentru formarea oferită, care m-a ajutat să concretizez aceste studii. Totodată, aș vrea să mulțumesc și socrilor, cumnatului și viitoarei cumnate, precum și nașilor de cununie și finilor pentru ajutorul oferit de-a lungul acestor ani și nu numai, precum și pentru înțelegerea arătată.

În mod special, aș vrea să-i mulțumesc soției mele Marcela, pentru toată înțelegerea și susținerea pe care mi le-a arătat, pentru toată munca pe care a depus-o pentru a-mi acorda timpul și liniștea necesare studiilor aferente acestei teze și pentru finalizarea acesteia. De asemenea, aș vrea să mulțumesc și fiului meu Alexandru pentru înțelegerea de care a dat dovadă de la o vârstă atât de fragedă precum și pentru zâmbetul lui care mi-a descrețit fruntea de atâtea ori.

De asemenea, aș vrea să mulțumesc tuturor celor pe care îi cunosc și pe care nu i-am menționat anterior, dar care au fost alături de mine în toți acești ani.

Mai presus de toate, as vrea să-i mulțumesc lui Dumnezeu că mi-a dat răbdare, putere și sănătate să finalizez această teză, precum și să mă bucur de aceste zile.

Timișoara,
Februarie 2018

Ovidiu Sicoe

BUPT

Cuprins

1 Introducere **7**
1.1 Context . . . 7
1.2 Ținte și realizări proprii . . . 8
1.3 Organizarea lucrării . . . 8

2 State Of The Art **10**
2.1 Operatori aritmetici existenți . . . 10
2.1.1 Generalități . . . 10
2.1.2 Unități pentru înmulțire-adunare . . . 11
2.1.3 Unități pentru împărțire . . . 11
2.1.4 Performanțe comparative cu microprocesoarele de uz general 12
2.1.5 Implementări pe FPGA-uri . . . 13
2.1.6 Concluzii preliminare . . . 18
2.2 Operatori aritmetici imprecişi și metrici . . . 18
2.2.1 Prezentare generală . . . 19
2.2.2 Analiză a studiilor curente . . . 19
2.2.3 Concluzii preliminare . . . 30

3 Noțiuni Preliminare **31**
3.1 Pipeline grafic . . . 31
3.2 Reprezentarea obiectelor tridimensionale . . . 34
3.3 Transformări de Coordonate . . . 35
3.4 Transformări Geometrice Afine . . . 35
3.4.1 Cazul general . . . 35
3.4.2 Translația . . . 36
3.4.3 Scalarea . . . 38
3.5 Reprezentări de numere reale . . . 40
3.5.1 Virgulă fixă . . . 40
3.5.2 Virgulă mobilă . . . 40

BUPT

4 Operatori matriciali folosiți în pipeline-ul grafic 41
4.1 Prezentare generală 41
4.2 Justificare 42
4.3 Implementare 43
4.3.1 Operatorul de translație 43
4.3.2 Operatorul de scalare 48
4.4 Concluzii preliminare 53

5 Implementare calcule aproximative 55
5.1 Motivație 55
5.2 Mediul de rulare 56
5.2.1 Implementare OpenGL ES 1.1 utilizată 57
5.2.2 Colectarea obiectelor tridimensionale folosite pentru cercetare 58
5.2.3 Transformarea datelor de intrare 59
5.2.4 Aplicația propriu zisă 61
5.2.5 Compararea imaginilor obținute 62
5.2.6 Centralizarea, filtrarea și interpretarea datelor 65
5.3 Schimbarea formatului folosit la nivel de stagiu al pipeline-ului grafic 66
5.4 Procesarea datelor și obținerea rezultatelor 70

6 Rezultate calcule aproximative 73
6.1 Informații generale despre setul de obiecte tridimensionale folosit . 73
6.2 Imagini alterate. Evaluare subiectivă 75
6.3 Imagini alterate. Evaluare obiectivă: MSE și PSNR 77
6.3.1 Definire 78
6.3.2 Valori obținute 78
6.3.3 Grafice ale valorilor obținute 79
6.4 Introducerea unei metrici noi 80
6.4.1 Definire 80
6.4.2 Valori obținute 81
6.4.3 Grafice ale valorilor obținute 82
6.5 Analiză comparativă între MSE, PSNR și M% 83
6.6 Metrici agregate 84
6.7 Concluzii preliminare 85

7 Concluzii și perspective 86
7.1 Contribuții originale 86
7.2 Perspective de viitor 88

A Histograme caracteristici obiecte tridimensionale 90

B Suită de cadre alterate 16_16 94

BUPT

C Suită de cadre alterate 32_32 98

D Imagini Rezultate 1 102

E Imagini Rezultate 2 106

F Grafice valori MSE și PSNR 110

G Grafice valori M% 121

H Grafice valori medii metrici 127

I Grafice valori medii M% pentru obiecte 131

J Listă de publicații 135

J.1 Lucrări indexate ISI publicate în România 135

J.2 Lucrări indexate ISI publicate în străinătate 135

J.3 Altele 136

Listă de figuri 137

Listă de tabele 140

Bibliografie 141

BUPT

Capitolul 1

Introducere

1.1 Context

Într-o lume în care dispozitivele mobile sunt tot mai răspândite, și tot mai bogate în conținutul grafic pe care îl prezintă utilizatorilor, consumul de putere, viteza de calcul și miniaturizarea sunt puncte esențiale în evoluția acestora. De asemenea, reutilizarea resurselor hardware este un deziderat care, odată atins, scade semnificativ costurile de producție ale acestor dispozitive.

Pentru a rezolva acest deziderat, o soluție ar fi reutilizarea resurselor fizice specializate pentru rezolvarea unei anumite probleme prin "reprogramarea" hardware a unei noi versiuni a algoritmului, pe măsură ce acesta evoluează și este eficientizat. Astfel de dispozitive hardware care suportă reconfigurarea resurselor fizice pentru a putea implementa un nou algoritm sunt FPGA-urile(en: Field Programmable Gate Arrays). În ultima perioadă, ele reprezintă o soluție viabilă mai ales în faza de prototipizare, dar și în faza de producție.

În același context al dispozitivelor mobile, procesoarele grafice au avut o evoluție și o miniaturizare spectaculoasă, astfel încât ele pot fi ușor încorporate într-un astfel de dispozitiv. Pentru a ajunge la această realizare, pe lângă miniaturizarea fizică a resurselor hardware, au fost făcute și o serie de optimizări din punct de vedere al algoritmilor folosiți și al micro-unităților de operații aritmetice folosite.

Astfel, se poate ajunge chiar să fie folosite unități aritmetice imprecise care să furnizeze un rezultat grafic utilizabil și acceptabil de către utilizatorul uman. Extrapolând această idee se pot utiliza și algoritmi imprecişi sau chiar formate imprecise de reprezentare a numerelor. Toate aceste compromisuri în precizie pot aduce câștiguri semnificative în consumul de putere sau cantitatea de resurse fizice necesare pentru implementarea unui astfel de dispozitiv.

BUPT

1.2 Ținte și realizări proprii

În contextul prezentat anterior, teza curentă propune o serie de idei, studii și soluții care să ajute la dezvoltarea și îmbunătățirea procesoarelor grafice dedicate dispozitivelor mobile.

Astfel, am conceput o serie de operatori matriciali corespunzători unor transformări geometrice afine, implementați pe FPGA. Acești operatori matriciali sunt utilizați în pipeline-ul grafic, în prima fază, cea de transformare a coordonatelor. Principala caracteristică a acestor operatori va fi posibilitate de a-i configura înainte de sinteză, prin specificarea dimensiunii reprezentării numerelor folosite, gradul de paralelizare dorit sau frecvența țintă.

În continuare, ne-am propus să studiem influența utilizării multiplelor formate de numere reale în cadrul aceluiași pipeline grafic asupra rezultatului final. Astfel, am folosit o implementare software proprie a specificației Open GL ES 1.1, pe care am adaptat-o astfel încât să se poată schimba la execuție formatul folosit pentru reprezentarea numerelor reale.

Pentru a putea realiza acest lucru, am creat o bibliotecă numeroasă, care să conțină obiecte tridimensionale variate care să acoperă o plajă cât mai largă de valori de intrare pentru un astfel de pipeline grafic hibrid.

De asemenea, am proiectat și implementat o serie de module software prin care să putem crea un proces sistematic și bine definit, având în vedere faptul că vor fi de procesat un număr mare de date de intrare, care vor produce un număr și mai mare de imagini la ieșire, dar și o mulțime de valori care vor trebui centralizate, filtrate și interpretate.

Nu în ultimul rând, am analizat rezultatele astfel obținute prin utilizarea unor metrici deja existente pentru compararea de imagini alterate relativ la o imagine de referință. În plus, am definit o nouă metrică cu o semantică a valorilor sale care să fie ușor de înțeles și interpretat.

1.3 Organizarea lucrării

Astfel, am structurat această teză după cum urmează. În Capitolul 2 prezentăm stadiul actual al cercetării în subiectele atinse pe parcursul acestei teze, precum și exemple concrete de algoritmi sau implementări deja existente și chiar utilizate până în prezent.

În continuare, Capitolul 3 prezintă succint o serie de noțiuni preliminare necesare pentru înțelegerea subiectelor ce vor fi dezvoltate pe larg în această lucrare. Astfel, sunt introduse pe scurt noțiuni de grafică tridimensională și de reprezentări ale numerelor reale.

Capitolul 4 descrie o serie de arhitecturi ale unor operatori matriciale corespunzători transformărilor geometrice afine de translație și scalare, precum și

BUPT

detalii de implementare ale acestora. De asemenea, prezintă modul de validare a acestor operatori, precum și o serie de date obținute în urma sintezei acestor operatori pentru diferite configurații.

Capitolul 5 prezintă modulele software dezvoltate și folosite pentru a implementa procesul prin care am colectat o varietate de obiecte tridimensionale, le-am proiectat, folosind un pipeline grafic, pe o suprafață bidimensională obținând o serie de imagini de referință și, respectiv, alterate. Apoi prezintă și modulele utilizate pentru compararea imaginilor obținute și, la final, descrie întreg procesul folosit.

Capitolul 6 prezintă biblioteca de obiecte tridimensionale folosite, cu o scurtă analiză a distribuției caracteristicilor acestora. În continuare este prezentată o suită de imagini obținute, precum și graficele create pe baza valorilor metricilor aplicate comparațiilor acestor imagini. De asemenea, se descrie o nouă metrică pentru evaluarea diferențelor între două imagini similare, propusă în această teză. De asemenea, se efectuează o comparație cu metricile deja existente.

La final, Capitolul 7 prezintă o serie de concluzii desprinse din cercetarea întreprinsă pentru elaborarea acestei teze, precum și o serie de acțiuni ce pot fi desfășurate pentru continuarea acesteia.

BUPT

Capitolul 2

State Of The Art

2.1 Operatori aritmetici existenți

2.1.1 Generalități

Procesoarele grafice actuale au evoluat masiv înspre procesoare multicore, capabile să execute masiv operații în paralel [1]. Pentru a realiza acest lucru, au fost adaptate și îmbunătățite arhitecturile deja existente sau au fost dezvoltate altele noi. O altă măsură care a ajutat la realizarea acestui obiectiv a fost încorporarea unităților hardware specializate în realizarea unor operații aritmetice compuse. Un bun exemplu în acest sens ar fi $A \times B + C$. Aceste unități de înmulțire-adunare sunt denumite FMA(fused multiply-add) [2].

Într-un pipeline grafic, principalele operații efectuate sunt înmulțirile matriciale sau de matrici cu vectori. Aceste operații sunt necesare pentru a trece datele de intrare, care reprezintă coordonatele punctelor, din spațiul lumii în coordonatele ferestrei în care se afișează conținutul grafic. Aceste înmulțiri sunt necesare pentru a realiza proiecția obiectelor și pentru a facilita anumite operații, cum ar fi decuparea la fereastra de vizualizare. În [3] se prezintă un design și o implementare pentru FPGA a unui astfel de operator de înmulțire a unei matrici cu un vector. Autorii pretind o reducere a spațiului de stocare a matricilor și a lățimii de bandă necesare operațiilor de până la 43%, prin introducere unui nou format de reprezentare a matricilor cu valori împrăștiate, față de formatele deja existente.

De asemenea, în ultima perioadă se pune tot mai mare accent pe reconfigurare hardware. Un bun exemplu este oferit în [4], unde se prezintă o arhitectură reconfigurabilă dinamic a preciziei operanzilor de virgulă flotantă. Pentru a sprijini această idee, [5] prezintă o arhitectură reconfigurabilă pentru sistemele care includ comunicații fără fir, în care componentele ce asigură această facilitate sunt implementate pe FPGA.

BUPT

2.1.2 Unități pentru înmulțire-adunare

Unitățile pentru înmulțire-adunare implementate hardware îmbunătățesc viteza de execuție a operațiilor efectuate nu doar în procesoarele grafice ci și în procesoarele multimedia, procesoarele de imagini, DSP-uri precum și alte procesoare specializate [6–10]. Ele sunt capabile să execute și operații de împărțire și extragere a rădăcinii pătrate [1], precum și operații simple de înmulțire sau adunare [2].

Principalele avantaje ale unităților FMA de virgulă flotantă sunt:

- Nu este nevoie de rotunjiri separate pentru operația de înmulțire respectiv pentru cea de adunare. Este suficientă o rotunjire finală, după efectuarea ambelor operații [2].

- Nu este nevoie de despachetări din formatul IEEE 754 în formatul intern utilizat de unitate și invers, din formatul intern în formatul IEEE 754, pentru fiecare operație în parte, ci este suficient să se realizeze o singură data pentru operația compusă.

- Reducerea costurilor de implementare prin folosirea acelorași componente atât pentru operația de înmulțire, cât și pentru cea de adunare [2].

În implementarea oricărui algoritm pe FPGA se are în vedere minimizarea resurselor folosite și obținerea unor frecvențe de lucru cât mai ridicate. Astfel, [11] propune o arhitectură pentru implementarea unităților de înmulțire-adunare pe FPGA-uri tip insulă. Totodată, [11] prezintă și implementarea aferentă, precum și măsurătorile efectuate pe aceasta, reieșind o îmbunătățire de aproximativ 50%, atât ca și timp de execuție, cât și din punct de vedere al resurselor utilizate, relativ la implementările existente.

2.1.3 Unități pentru împărțire

Împărțirea nu este una dintre cele mai folosite operații aritmetice în procesoarele grafice și totuși, toate au unități hardware dedicate pentru împărțire. Unul dintre cei mai implementați algoritmi pentru împărțire din procesoarele deja existente este SRT(denumit după Sweeney, Robertson și Tocher) [12]. Unul dintre principalele motive ar fi că algoritmul SRT pentru împărțire tratează fiecare cifră pe rând si este foarte potrivit pentru implementări VLSI [13]. Din această cauză, o mare parte din cercetare a fost axată pe îmbunătățirea performanței algoritmului și pe minimizarea suprafeței utilizate de siliciu. O parte din ideile care susțin această afirmație pot fi găsite în [14–19].

Cu toate acestea, problema împărțirii la o generație mai veche a procesoarelor Pentium este binecunoscută și a scos în evidență că algoritmul SRT este susceptibil la erori de implementare, în special din cauza tabelelor de căutare

BUPT

implementate eronat; cu toate acestea, nu a fost desfășurată o cercetare aprofundată în această direcție [12]. [12] este unul dintre puținii care oferă o verificare matematică pentru corectitudinea implementării algoritmului SRT general, de rază 2^n.

Un anumit grad de paralelism în executarea algoritmului SRT este oferit de ideile descrisă în [15] și [16]. Autorii propun calcularea restului parțial probabil în paralel cu selectarea cifrei câtului, precum și transformarea câtului din forma redundantă în forma convențională. Ca să fie posibilă efectuarea concomitentă a acestor operații, restul parțial trebuie să fie limitat la un interval restrâns: $|R_j| < d_{min}$ [15].

În plus, pe lângă paralelism, performanța algoritmului este influențată de baza de numerație aleasă pentru reprezentarea numerelor și de factorul de redundanță al mulțimii de cifre aleasă pentru reprezentare [19]. Cu cât baza de numerație este mai mare, cu atât sunt necesari mai puțini pași, dar mulțimea de cifre posibile are mai multe elemente, complicând logica de selecție.

Pentru minimizarea consumului de energie, conversia cifrelor câtului din forma redundantă în forma normală poate fi substituită cu un sumator folosit în ultimele stagii ale algoritmului [17]. Această îmbunătățire, din punct de vedere al costului, duce la o penalizare în performanța algoritmului, întârziind calculul rezultatului final.

O altă alternativă care merită luată în considerare pentru diminuarea costurilor și a consumului de putere sunt unitățile hibrid pentru împărțire și calculul rădăcinii pătrate. [18] și [20] sunt două lucrări care promovează acest concept. Aceste două operații aritmetice sunt candidate foarte bune pentru acest gen de optimizare datorită asemănării algoritmilor lor de calcul.

În afară de algoritmul clasic SRT, mai sunt disponibili și alți algoritmi care obțin pe rând câte o cifra a rezultatului final, atât pentru împărțire cât și pentru calculul rădăcinii pătrate [13, 20–22]. Fiecare algoritm prezentat are propriile beneficii și neajunsuri în ceea ce privește performanța, consumul de putere și costurile de implementare.

2.1.4 Performanțe comparative cu microprocesoarele de uz general

S-a arătat că diferiți algoritmi pentru operații aritmetice efectuate în virgulă flotantă, implementați pe FPGA pot să fie mai rapizi decât implementările din microprocesoare de ultimă generație datorită posibilității masive de paralelizare a diferiților pași din algoritm [23]. O analiză axată pe aceste performanțe este oferită de [24]; în acest articol se subliniază faptul că nu doar prin paralelizare se obțin aceste îmbunătățiri ci și prin accesul mai rapid la memorie cât și prin cantitatea mai mare de date care poate fi procesată. Astfel, în [24], se arată că o

BUPT

implementare pe FPGA (Xilinx VIrtex II) poate fi de până la 47 de ori mai rapidă decât o implementare pe un microprocesor de uz general (Intel Pentium III).

În [25] se prezintă o modalitate de paralelizare a sub-operațiilor necesare efectuării operațiilor de adunare repetată în virgulă flotantă. De aici reiese că mici îmbunătățiri se pot aduce algoritmilor deja implementați în microprocesoarele existente, rezultând o scădere a timpilor de execuție ai algoritmului de până la șapte ori. De asemenea, [11] prezintă o modalitate de implementare a unităților de înmulțire-adunare pe FPGA-uri astfel încât se obține o reducere a resurselor folosite de 55% și o creștere a frecvenței medie de tact folosite de 40.7% față de arhitecturile existente la momentul respectiv.

2.1.5 Implementări pe FPGA-uri

De-a lungul timpului s-a încercat realizarea unor biblioteci de operatori de virgulă flotantă implementați pentru FPGA-uri. Majoritatea acestor biblioteci conțin operatori pentru operațiile aritmetice de bază(adunare, scădere, înmulțire, împărțire sau extragerea rădăcinii pătrate) [26, 27]. De asemenea, dezvoltarea de noi operatori pentru operații aritmetice compuse poate să aducă îmbunătățiri semnificative, relativ la implementările care folosesc procesoare de uz general, care nu au instrucțiuni dedicate pentru aceste operații compuse [23].

Următorul pas firesc ar fi implementarea unor astfel de operații compuse. Astfel, [27] propun implementarea unor algoritmi de calcul al logaritmului si al ridicării la putere, cu operanzi numere flotante, și adaptează algoritmii astfel încât să faciliteze implementarea pe FPGA. Autorii tratează și cazurile de eroare specifice operațiilor de virgulă flotantă, astfel încât operatorii sunt compatibili cu rezultatele produse de o implementare software. De asemenea, au comparat implementările propuse, rulate pe o placă Virtex-II, cu funcțiile echivalente găsite în biblioteca `glibc`, rulate pe un procesor Intel Xeon la 2.4 GHz. Rezultatele au fost pozitive, obținând o îmbunătățire semnificativă, de aproximativ 10 ori, a performanței, din punct de vedere al numărului de operații efectuate într-o secundă. De asemenea, din punct de vedere al resurselor folosite, implementările lor au fost mai eficiente comparativ cu singura implementare pe care au găsit-o la acea vreme.

Mai recent, [28] prezintă un operator pentru calculul ridicării la putere axat pe resursele plăcilor FPGA actuale(memorie RAM încorporată, blocuri DSP cu unități pentru înmulțire sau adunare). Autorii estimează implementarea algoritmului pentru Virtex-6, pentru operatori în virgulă flotantă, de dublă precizie, la maxim 60 GFPexp/s(giga ridicări la putere în virgulă flotantă / secundă). Comparativ, o implementare software de mare performanță(Intel Vector Math Lib), rulată pe un procesor Itanium-2 sau Core i7, poate atinge maxim 4 GFPexp/s.

BUPT

Operațiile trigonometrice fac parte dintr-o altă categorie de operații des folosite în diferite domenii. Funcțiile trigonometrice de bază, cu ajutorul cărora, prin compunere, se pot calcula și celelalte funcții trigonometrice sunt *sinusul* și *cosinusul*. Astfel, [29] propune o implementare a operațiilor trigonometrice de bază pentru numere reprezentate în virgulă flotantă. Ca și în cazul operatorilor pentru calculul logaritmului, respectiv al ridicării la putere, autorii propun o un algoritm adaptat special pentru implementarea pe FPGA. O parte costisitoare a implementării, în special din cauza resurselor utilizate, o constituie partea de reducere a gradelor la valori în intervalul $[0, \pi/4]$ [29]. O altă optimizare propusă de aceeași autori o constituie calcularea în paralel a sinusului și cosinusului pentru o valoare dată, astfel încât modulul de reducere va fi folosit o singură dată. Această optimizare este benefică în special în aplicațiile grafice, la calculul valorilor matricii de rotație, unde trebuie calculat atât sinusul cât și cosinusul aceluiași unghi [29]. De asemenea, Detrey et al. propune tratarea erorilor specifice și preciziei definite de standardul IEEE 754. Astfel, implementarea este compatibilă în întregime cu echivalentul software. Implementarea a fost rulată pe o platformă Virtex-II-4, rezultând o valoare calculată la fiecare 10 ns. Pentru o înțelegere a îmbunătățirii aduse, au fost evaluate și funcțiile sinus, respectiv cosinus din biblioteca `libm`, pe un procesor Pentium la 2.4 GHz. Acesta din urmă a reușit să producă un rezultat la fiecare 200 de cicli de clock, adică un rezultat la fiecare 80 ns [29]. Aceste rezultate indică o viteză mai mare de calcul a implementării pe FPGA. De asemenea, și resursele utilizate sunt mai mici, relativ la alte implementări existente la momentul respectiv [29].

Un alt articol care tratează funcțiile trigonometrice pentru operatori reprezentați în virgulă fixă este [30]. Diferența este că acest articol propune arhitecturi pentru familii de plăci FPGA mai noi, care dispun și de blocuri de memorie RAM precum și de blocuri DSP. Principala idee pe care o accentuează autorii este faptul că se pot realiza o mulțime de compromisuri între performanță(frecvența de lucru și latență), pe de o parte și consumul de resurse(LUTs, blocuri DSP și blocuri de memorie) pe de altă parte.

Alte operații des folosite sunt adunarea repetată și repetițiile de înmulțire urmată de adunare. În special ultima operație este des folosită într-un pipeline grafic la înmulțirile de matrici sau matrice cu vector. De asemenea, operația de înmulțire-adunare este folosită în calculul produsului scalar sau în alți algoritmi de procesare digitală a semnalelor. Astfel, Dinechin et al. [31] propune o implementare pentru FPGA a operațiilor de adunare repetată și de înmulțire-adunare, utilizând FloPoCo [32], un generator de cod VHDL. Și autorii acestui articol subliniază faptul că aceste operații compuse merită atenție, nu numai datorită domeniilor de aplicație ci și predispoziției spre pipeline-izare. Pentru operația de adunare repetată, se propune o arhitectură parametrizabilă, în care lungimea acumulatorului este configurabilă. De asemenea, încercând

BUPT

să îmbunătățească performanța operatorului, propun și o variantă de arhitectură în care folosesc sumatoare parțiale carry-save, ale căror lanț de carry are lungimea configurabilă. Cu implementarea propusă, au reușit o performanță similară, dar folosind mai puține resurse comparativ cu CoreGen [31].

Pentru a doua operație pe care s-au axat, folosesc operatorul de adunare repetată, la care au adus unele modificări [31]. Aceste modificări constau în principal în însumarea rezultatelor unor înmulțitoare exacte, care nu fac rotunjirile necesare standardului IEEE 754. Rotunjirile au loc doar pentru rezultatul final, reducându-se astfel resursele utilizate și timpul de obținere a rezultatului final. Cu toate acestea, rezultatele sunt mai slabe comparativ cu CoreGen [31].

FloPoCo [32] este un generator de cod VHDL care oferă suport pentru realizarea unor arhitecturi pipeline-izate [33]. De asemenea, ușurează realizarea operatorilor aritmetici pentru operanzi de diferite dimensiuni. Este scris în C++ și pentru a crea propriul operator este nevoie să se descrie semnalele de intrare/ieșire și logica acestuia utilizând funcțiile puse la dispoziție de această bibliotecă. Folosind FloPoCo, se înlocuiesc construcțiile parametrice din codul VHDL cu parametrii utilizați în codul C++ pentru descrierea operatorului. Astfel, dacă este nevoie de crearea unui sumator pentru operanzi pe 8 biți sau 16 biți, se descrie un singur operator cu parametru pentru dimensiunea operanzilor. În interiorul acestei descrieri se va specifica că este nevoie de x sumatoare pe un bit, unde x este dimensiunea operanzilor. De asemenea, FloPoCo oferă suport automat pentru pipeline-ul intern al unui operator [33]. În plus, se pot și sincroniza pipeline-urile interne a doi operatori, dacă unul este nevoit să aștepte după rezultatul celuilalt. Astfel, folosind această bibliotecă se reduce timpul de dezvoltare al operatorilor care au structuri interne repetitive, în special datorată arhitecturilor care implică pipeline-uri. Timpul economisit este cu atât mai mare, cu cât operatorul are mai multe astfel de structuri repetitive, potențial mai multe stagii pipeline, iar paralelizarea masivă este principala caracteristică prin care se îmbunătățesc performanțele algoritmilor implementați pe FPGA-uri. O altă funcționalitate oferită de FloPoCo este generarea automată a testbench-ului pentru operatorul implementat [33]. Este nevoie doar de specificarea valorilor semnalelor de intrare și de valorile așteptate la ieșire, iar framework-ul va genera cod VHDL sintetizabil pentru platforma dorită.

Dinechin et al. [34] subliniază diferența dintre modul de gândire pentru a dezvolta operatori pentru un procesor de uz general și operatori pentru FPGA. Principala diferență o constituie domeniul de utilizare al lor. Dacă pentru un procesor de uz general este necesar să fie capabil să ofere suport pentru operații din diferite domenii, în cazul FPGA-urilor se pot crea operatori specifici pentru o anumită aplicație sau care să satisfacă anumite particularități, simplificând astfel logica internă sau reducând costurile de implementare. Scopul FloPoCo [32] este de explora operatorii aritmetici exotici, care nu vor fi utilizați în viitorul

BUPT

apropiat pentru procesoarele de uz general [34], dar care sunt necesari pentru anumite tipuri de aplicații.

O altă operație des utilizată este înmulțirea cu o constantă. În [35] se propun o arhitectură dedicată pentru FPGA pentru această operație, implementarea sa, precum și o propunere de optimizare a algoritmului folosit. Autorii au comparat mai mulți algoritmi care folosesc reprezentări diferite ale unei constante întregi și ajung la concluzia că o reprezentare sub formă de graf orientat aciclic ajută la economisirea de resurse precum și la scurtarea căii critice. Astfel, cu arhitectura propusă, pentru numere reprezentate pe 50 de biți, implementată cu ajutorul FloPoCo, s-au obținut rezultate puțin mai bune, atât din punct de vedere al performanței cât și al resurselor utilizate, comparativ cu o implementare folosind reprezentarea lui Lefèvre [36], deja prezentată în alte articole similare. Pentru numere reprezentate pe mai mulți biți însă, performanțele au fost mai slabe, raportate la aceeași implementare.

O idee de îmbunătățire a performanțelor unităților aritmetice specifice unei clase particulare de aplicații, implementate pe FPGA este prezentată în [37]. În acest articol se propune un algoritm de sumator pentru numere reale sau complexe, reprezentate în sistem logaritmic. De asemenea, autorii prezintă și implementarea mai multor arhitecturi pentru FPGA a algoritmului propus, în funcție de precizia aleasă. Pe baza acestor implementări, s-a ajuns la concluzia că ele oferă performanțe similare cu alte implementări existente la acea vreme, însă folosind mai puține resurse.

Una din cele mai costisitoare operații aritmetice de bază este înmulțirea. Odată cu apariția unităților de înmulțire din DSP-urile aferente unei plăci FPGA, s-a pus problema folosirii lor pentru a obține înmulțitoare pentru numere reprezentate pe mai mulți biți decât dimensiunea intrărilor unui singur astfel de înmulțitor. Astfel, în [38], se propune o implementare a unor algoritmi optimizați pentru reducerea consumului de DSP-uri, încercând să obțină aceleași performanțe din punct de vedere al latenței. Articolul prezintă o implementare pentru algoritmul Karatsuba-Ofman, precum și două implementări ale unor adaptări ale acestui algoritm, pentru a reduce numărul de blocuri DSP utilizate. Una dintre tehnicile utilizate este împărțirea operanzilor în blocuri de dimensiuni corespunzătoare intrărilor blocurilor DSP și cascadarea liniară a blocurilor. Cea de-a doua optimizare constă în aranjarea blocurilor DSP într-o structură de formă pătrată, în care operația de înmulțire se desfășoară atât pe verticală, cât și pe orizontală. Folosind aceste optimizări, Dinechin et al. [38] a reușit sintetizarea unor operatori mai economici din punct de vedere al blocurilor DSP utilizate, comparabili ca și frecvență de lucru cu alte implementări existente, dar cu o latență chiar și de trei ori mai mică.

Pe măsură ce aplicațiile științifice au progresat și au avut nevoie de tot mai multă putere de calcul și de o precizie ridicată, a fost adoptat și standardul

BUPT

IEEE-754 pentru numere flotante de cvadruplă precizie. Astfel, în [39], se prezintă un înmulțitor pentru numere reprezentate după acest standard, cu mantisa pe 112 biți și care implementează și rotunjirile descrise în această specificație. Arhitectura descrisă folosește tot o așezare în formă de grilă și se folosește de resursele hardware oferite de platforma FPGA folosită. Pentru generaraea codului VHDL s-a folosit de asemenea biblioteca FloPoCo [32].

Marele avantaj al algoritmilor implementați pe FPGA este că ei se pot adapta la nevoile concrete ale aplicației pentru care sunt folosiți, rezultând astfel performanțe comparabile sau chiar mai bune decât folosind un procesor de uz general. În plus, ele se pretează a fi folosite pentru sisteme încorporate sau chiar mobile, care necesită și un consum scăzut de energie. Un alt avantaj oferit de FPGA-uri îl reprezintă posibilitatea de a crea arhitecturi cu multe stagii de pipeline, scăzând astfel latența obținerii a două rezultate consecutive. Susținând această idee de paralelizare a algoritmilor prin stagii pipeline alcătuite după nevoile aplicației, în articolul [40] se prezintă arhitecturi și implementări de operatori aritmetici de o complexitate ridicată, pipeline-izați, pentru care s-au obținut rezultate notabile din punct de vedere al vitezei de execuție și a resurselor folosite. Pentru implementarea acestor arhitecturi s-a folosit biblioteca FloPoCo [32]. Autorii prezintă modalitatea de utilizare a acestei biblioteci pentru a crea astfel de operatori, utilizând exemple practice, iar la final compară rezultatele obținute în urma utilizării acestor noi operatori cu cei oferiți deja de alți autori. Rezultatele sunt îmbucurătoare, obținând latențe chiar și de patru ori mai mici și utilizând cu pana la de două ori mai puține resurse. Principalul avantaj al folosirii acestei biblioteci pe care autorii [40] îl scot în evidență este managementul automat al pipeline-ului. O altă caracteristică care scurtează timpul de dezvoltare al operatorilor este generarea automată a codului pentru testbench-ul arhitecturii implementate [33].

Un alt articol care subliniază că FPGA-urile sunt pretabile pentru implementări pipeline-izate este [41]. În acest articol, autorii propun o arhitectură, precum și implementarea acesteia, pentru sumatoare de numere întregi reprezentate pe un număr mare de biți(128). De asemenea, pentru implementarea arhitecturii prezentate se folosește FloPoCo [32]. Pe lângă faptul că ajută la pipeline-izarea operatorului, autorii subliniază și abilitatea bibliotecii FloPoCo de a raporta latențele semnalelor de ieșire.

În [42], se descrie implementarea pe FPGA a unui algoritm pentru extragerea rădăcinii pătrate, folosind FloPoCo. Arhitectura operatorului este gândită pentru plăci FPGA care conțin înmulțitoare încorporate, precum și blocuri de memorie RAM. Autorii au încercat implementarea unui algoritm polinomial, pe care l-au comparat cu implementările precedente, implementări care folosesc algoritmi cu obținerea unui bit din rezultat la fiecare ciclu sau algoritmi bazați pe înmulțire. Concluzia a fost că soluția propusă aduce o îmbunătățire din

BUPT

punct de vedere al latenței și al frecvenței de lucru, comparativ cu ambele tipuri de algoritmi comparați. Cu toate acestea, se subliniază costul mare al implementării rotunjirii corecte conform standardului IEEE 754.

De asemenea, și autorii articolului [43] descriu arhitecturi de operatori aritmetici complecși, cum ar fi $\sqrt{1+x}$, $\log_2(1+x)$, $1-\cos(\frac{\pi}{4}x)$, $\frac{\pi}{4}-\frac{\sin(\frac{\pi}{4}x)}{x}$ și $\frac{\log(x+1/2)}{x-1/2}$. Operatorii sunt aproximați prin funcții polinomiale și sunt pentru numere în virgulă flotantă cu mantise de 23 de biți(simplă precizie), 36 de biți și 52 de biți(dublă precizie). Particularitatea acestor arhitecturi constă în faptul că ele sunt generate automat, având ca și date de intrare funcția dorită, precizia precum, gradul polinomului și alți câțiva parametrii, iar rezultatul final va fi codul VHDL al arhitecturii generate. Pentru generarea codului VHDL s-a folosit FloPoCo, iar pentru parametrizarea operatorului, procesul automat mai are două etape intermediare, înaintea celei de generare, de aproximare a funcție polinomiale, din care rezultă tabelul de coeficienți și de optimizare a evaluării, din care rezultă parametrii operatorului.

2.1.6 Concluzii preliminare

Din articolele analizate am putut deduce că s-a depus un efort semnificativ pentru optimizarea algoritmilor aritmetici adresați diverselor tipuri de aplicații, în special prin adaptarea la particularitățile specifice fiecărei aplicații în parte (mărimea semnalelor de intrare/ieșire, costuri, performanțe necesare, și altele). O altă direcție urmată pentru îmbunătățirea performanțelor operatorilor implementați pe FPGA, făcându-i astfel comparabili cu cei din microprocesoarele de uz general, este pipeline-izarea masivă a stagiilor interne necesare algoritmului. Astfel, FloPoCo [32] vine în ajutorul design-ului unor operatori aritmetici masivi pipeline-izați și puternic configurabili, punând la dispoziție și un generator de cod VHDL pentru sinteza operatorului dorit, precum și a codului VHDL aferent testbench-ului acestui operator, ținând cont de cazurile de test specificate, ușurând astfel crearea unor astfel de operatori aritmetici și nu numai.

2.2 Operatori aritmetici imprecişi şi metrici

Pentru a optimiza și mai mult diferiții operatori folosiți în diverse aplicații, pe lângă particularizarea acestora pentru problema țintă pe care o au de rezolvat, s-a trecut și la diminuarea preciziei, acolo unde aplicațiile permit acest lucru. Această diminuare a preciziei în calculele efectuate poate fi privită ca și o particularitate pe care nu toate aplicațiile o au.

Pentru a optimiza și mai mult operatorii matriciali creați, care vor fi prezentați în Capitolul 4, vom recurge la utilizarea operanzilor care folosesc diferite

BUPT

reprezentări, cu diferite precizii. Astfel, în continuare vom prezenta stadiul cercetării actuale pentru operatorii cu pierdere de precizie și unele metrici folosite pentru compararea de imagini, necesare pentru a putea evalua performanțele operatorilor creați din prisma rezultatelor produse.

2.2.1 Prezentare generală

Aplicațiile actuale folosesc din ce în ce mai multe calcule aproximative pentru a spori viteza de execuție, consumul de resurse sau consumul de putere, mai ales in acele domenii în care acest lucru este permis. De asemenea, există din ce în ce mai multe componente hardware care execută direct operații cu pierdere de precizie.

Un astfel de domeniu, dispus să accepte calcule aproximative, este domeniul aplicațiilor grafice. Pentru o parte a aplicațiilor grafice este permis să randeze reprezentarea bidimensională a obiectelor tridimensionale primite la intrare cu o anumită lipsă de precizie. În același timp, pentru a putea formaliza gradul de acceptabilitate al unei astfel de imagini obținute prin operații care implică calcule cu pierderi de precizie, s-a încercat formularea unor metrici prin care să se poată compara astfel de imagini.

Un alt domeniu care acceptă calcule aproximative este procesarea de semnale digitale. Aici sunt folosite în special procesoare digitale cu arhitecturi speciale, particularizate pentru o anumită aplicație, fiind foarte greu sa se folosească același procesor pentru a rezolva probleme diferite. Se acceptă acest lucru pentru a avea o viteză de execuție ridicată, oferită în primul rând de implementarea dedicată a hardware-ului, lucru care nu ar fi posibil prin utilizarea unei soluții generale de calcul. Soluțiile implementate cu FPGA-uri reprezintă o soluție prin care se pot programa diferiți algoritmi pe același suport hardware. Astfel se pot ușor implementa soluțiile particulare, scăzând costurile aferente hardware-ului, nefiind nevoie să se creeze câte un nou hardware pentru fiecare algoritm în parte.

2.2.2 Analiză a studiilor curente

Un studiu despre compromisul dintre energia consumată și precizie este prezentat în [44]. În acest articol se subliniază importanța câștigului de energie și implicit a duratei de viață a dispozitivelor mobile prin reducerea preciziei operațiilor efectuate de unitatea de procesare grafică. Autorii s-au concentrat pe stagiul de procesare al vertex-urilor din pipeline-ul grafic și au încercat să cuantizeze compromisul dintre câștigul de energie și fidelitatea imaginii astfel obținute. Autorii s-au axat pe aplicarea îmbunătățirilor gândite asupra operatorilor aritmetici de adunare și înmulțire cu operanzi întregi. Una dintre concluziile importante a fost că se pot obține câștiguri de cel puțin 60% prin scăderea

BUPT

preciziei operațiilor aritmetice din stagiul analizat. În continuare, au reușit să obțină o îmbunătățire și mai semnificativă a energiei consumate, însă cu o înrăutățire a calității imaginii obținute. Chiar dacă ei afirmă că înrăutățirea calității e puțin semnificativă, nu prezintă o cuantificare a acestei afirmații.

Pentru a-și proba teoria, au proiectat și implementat câteva sumatoare și multiplicatoare cu precizie variabilă, reglabile dinamic. Astfel au adaptat trei sumatoare pentru a le putea folosi cu precizie variabilă: sumator cu propagare în cascadă a carry-ului (En: ripple carry adder), sumator cu selecție a carry-ului (En: carry select adder) și sumator Brent-Kung [45]. Pe lângă sumatoare, au modificat și un multiplicator cu salvare a carry-ului (En: carry-save multiplyer).

În același timp, au creat un model matematic al energiei consumate pentru a putea prezice consumul de energie în diferitele scenarii pe care le-au gândit. Astfel, au introdus modelul lor matematic în ATTILA [46] și au obținut o estimare a consumului de energie pentru intrările date, intrări de diferite precizii.

Pentru a-și testa implementările și pentru a-și valida modelul, ei au randat câteva cadre din aplicații folosite pe scară largă, precum Doom 3, Quake 4 și altele [44]. Concluziile lor au fost că încărcarea suplimentară adusă de suprimarea de putere a sumatoarelor și multiplicatoarelor nu e costisitoare și că precizia trebuie redusă cu unu sau doi biți pentru a vedea îmbunătățiri în consumul de energie.

FPGA-urile precum și procesoarele grafice sunt utilizate pentru a implementa algoritmi numerici în diferite aplicații ce au nevoie de o viteză ridicată de execuție [47]. În [47], autorii prezintă un studiu despre influența hardware-ului asupra algoritmilor pentru operații care utilizează formate ne standard pentru reprezentarea numerelor reale. Pentru a-și testa teoria, au ales algoritmi pentru înmulțirile matriciale. Astfel, pentru matrici foarte de mari, au ajuns la concluzia că implementările pe plăcile grafice au o viteză de execuție mai mare decât implementările similare de pe FPGA-uri. Analog, pentru matrici de dimensiuni mai reduse, același tip de algoritm are viteză mai mare de execuție pe FPGA-uri decât pe plăcile grafice. Pentru acest studiu, au fost folosite reprezentări în virgulă flotantă de precizie dublă sau înjumătățită.

Hardware-ul folosit pentru experimentele din [47] au fost o placă grafică NVidia Tesla C1060 și un FPGA Xilinx Virtex-6. Arhitectura plăcii grafice este special concepută pentru calcule paralele de mare viteză și programare de uz general pentru plăci grafice. Ea are suport pentru simplă și dublă precizie a numerelor reale reprezentate în virgulă flotantă și conține 240 de procesoare care rulează la o frecvență de 1.296 GHz, implementate pe un suport de 55 nm [48]. De cealaltă parte, placa FPGA este implementată pe un suport de 40 nm] [49], iar pentru sinteza algoritmilor au folosit ca și frecvență țintă 250 MHz. De asemenea, pentru implementarea algoritmilor au utilizat reprezentările de virgulă flotantă oferite de LogiCORE [50] și FloPoCo [32].

BUPT

În final, rezultatele prezentate în [47] indică superioritatea plăcilor grafice față de FPGA-uri atunci când vine vorba de viteza de execuție pentru algoritmii de înmulțire între matrici de numere reale de dimensiuni mari ($> 64x64$). Pentru matrici de dimensiuni mai mici, FPGA-urile au fost superioare plăcilor grafice pentru toate preciziile de numere reale utilizate. În continuare, autorii sugerează alte criterii de comparație între plăcile grafice și FPGA-uri, cum ar fi acuratețea calculelor, consumul de putere și energie sau utilizarea altor algoritmi, pe lângă cel utilizat în lucrarea aceasta. Luând în considerare acest ultim criteriu, autorii consideră că plăcile grafice sunt optimizate pentru înmulțiri matriciale și astfel, susțin că pentru alți algoritmi numerici care nu implică operații cu matrici, FPGA-urile ar putea oferi rezultate superioare plăcilor grafice. Aceste concluzii întăresc faptul că o aplicație folosită în lumea reală are nevoie de o analiză profundă pentru a alege soluția hardware optimă astfel încât să se obțină cele mai bune rezultate.

În continuare, o altă aplicație care exploatează folosirea de numere reale cu reprezentări cu precizie variabilă este prezentată în [51]. Autorul prezintă un mod de reprezentare a cuvintelor de culoare care folosește precizie variabilă. El susține că folosind această reprezentare se obțin degradări mai mici la compresie, în special la variații constante și omogene ale culorii. Pentru a-și valida teoria, a folosit sute de texturi, iar rezultatele obținute au fost că prin folosirea acestui algoritm se obțin rezultate mai bune decât la folosirea S3TC, pentru aceeași rată de compresie sau, pentru RGBA, rezultate similare cu S3TC, însă la o rată dublă de compresie.

Pentru a cuantiza diferențele între imaginea originală și cea compresată, autorul a folosit media absolută a erorilor, precum și media pătratelor erorilor, împreună cu rădăcina pătratică a mediei pătratelor erorilor. De asemenea a folosit metrica pentru similaritatea structurală pentru a măsura calitatea vizuală percepută. În același timp a utilizat metrica de neasemănarea structurală pentru a păstra avantajele metricii de similaritate structurală, dar are avantajele metricilor de distanță: pozitivă, reflexivă și simetrică. Astfel, folosind aceste metrici, autorul își susține și validează afirmațiile inițiale asupra algoritmului propus pentru comprimarea texturilor prin utilizarea numerelor reale reprezentate în virgulă mobilă.

O altă idee propusă în [52] este utilizarea de operații cu precizie variabilă în transformările geometrice și iluminare pentru a accelera operațiile dintr-un pipeline grafic. Astfel, au făcut o legătură între distanța unui obiect țintă față de fereastra de vizualizare, poziția acestuia în spațiul de vizualizare și precizia necesară pentru a desena respectivul obiect, astfel încât să obțină o reprezentare cât mai exactă a obiectului. În final, sunt prezentate rezultatele obținute în urma rulării algoritmului descris anterior pe date din diferite domenii, cum ar fi scanere laser tridimensionale sau imagini CAD(computer aided design) obținute

BUPT

prin procedee mecanice sau procedurale. Principala lor concluzie este că abordarea lor poate aduce îmbunătățiri asupra vitezei de execuție, fără a degrada calitatea vizuală a imaginii desenate. Cu toate că au făcut această afirmație, nu este prezentată o cuantizare matematică a diferenței obținute, ci doar o evaluare vizuală subiectivă.

În [53], autorul prezintă arhitectura unităților grafice particularizate ale firmei Apple. Astfel, printre altele, el prezintă îmbunătățirile aduse pentru a avea un consum mai mic de putere și chiar un câștig de performanță. Pe lângă utilizarea unor registre mai mici care să acopere dimensiunea numerelor reale reprezentate în virgulă mobilă de jumătate de precizie(16 biți), au făcut disponibilă o interfață prin care se pot utiliza acești regiștrii și, odată cu ei, se poate implicit utiliza formatul de virgulă mobilă de jumătate de precizie, pe lângă cel deja existent de simplă precizie. Astfel, autorul sugerează că în acest procesor există un convertor între precizie simplă și înjumătățită și vice-versa gata inclus în calea principală a datelor. Deși această abordare necesită un consum hardware suplimentar pentru implementare, se subliniază câștigul de viteză, simplificarea compilatorului de shader-e și, nu în ultimul rând, câștigul de putere.

În continuare, [54] prezintă o arhitectură pentru sumatoare imprecise care pot să piardă lin din precizia cu care execută calculele, chiar în timpul execuției. Autorii susțin că sumatorul propus obține rezultate mai precise decât sumatoarele existente la vremea respectivă în calcularea imprecisă a transformatei cosinus discrete.

Sumatorul propus este compus din mai multe sumatoare standard, în care fiecare subunitate pentru adunare are același număr de biți. Primul nivel de reconfigurare constă în ajustarea lungimii operanzilor folosiți de fiecare dintre sumatoarele componente. Astfel, pentru fiecare sumator folosit au adăugat un multiplexor care selectează bitul de carry-in fie de la sumatorul precedent, fie de la o unitate de predicție a carry-ului atașată fiecărui sumator. Al doilea nivel de reconfigurare constă în selectarea numărului de biți de carry folosiți pentru predicție.

Pentru a testa îmbunătățirile aduse de sumatorul propus, au rulat un algoritm de transformată cosinus discretă asupra unor imagini, algoritm implementat pe rând cu diferite tipuri de sumatoare, inclusiv cel vizat. Totodată, validarea sumatorului din punct de vedere al corectitudinii funcționării au făcut-o prin furnizarea a un milion de valori aleatoare la intrări. Una dintre concluzii a fost că sumatorul propus obține rezultate precise, alături de sumatoare exacte, însă cu un consum mai mare de subcomponente și de putere. O altă concluzie, a fost că în comparație cu alte sumatoare imprecise se obține o precizie similară, dar cu mult mai puține resurse utilizate, însă cu o viteză de execuție puțin mai mare. De asemenea, autorii subliniază importanța curbei “calitate-efort” în

BUPT

conceperea circuitelor imprecise, reconfigurabile dinamic.

În mod asemănător, și în [55] se prezintă arhitecturi de sumatoare bazate pe porți logice XOR sau XNOR pentru calcule imprecise. Astfel, Yang și ceilalți motivează nevoia de circuite eficiente energetic din cauza limitării la care au ajuns circuitele CMOS. Autorii compară sumatoarele propuse din punct de vedere al energiei consumate, al întârzierii precum și al consumului de resurse fizice cu un sumator exact. Pentru a putea evalua arhitectura sumatoarelor imprecise, folosesc metrica pentru distanța erorilor. Principala optimizare propusă este reducerea unora dintre tranzistoarele folosite în sumatoarele precise. Prin renunțarea la unele tranzistoare, se reduce puterea dinamică și astfel se reduce consumul de putere, însă se pierde din precizie.

Pentru a putea compara sumatoarele propuse cu sumatorul precis, au simulat operațiile folosind Cadence NC-Sim. Astfel, per ansamblu, dintre toate cele patru sumatoare, cel precis a avut cel mai mare consum de energie, urmat de celelalte trei, în ordinea acurateții. Era de așteptat ca sumatorul precis să aibă cel mai mare consum, însă, deși unul dintre sumatoarele imprecise are o așezare mai simplă și mai mică a resurselor utilizate, acesta consumă mai mult ca și un alt sumator care folosește o suprafață mai mare. Concluzia finală este că sumatoarele imprecise obțin o îmbunătățire semnificativă a numărului de tranzistoare și, implicit, a puterii consumate.

Continuând aceeași idee, în [56] prezintă o arhitectură de sumator imprecis, configurabil în timpul rulării. De asemenea, și aici se subliniază faptul că prin utilizarea unui sumator imprecis se scade consumul de energie. În continuare, menționează că utilizarea de astfel de unități aritmetice imprecise este posibilă în special în contextul aplicațiilor care interacționează cu simțurile umane; acest lucru este favorizat de faptul că simțurile umane nu au nevoie de reprezentări extrem de precise pentru a putea percepe corect mesajul transmis. Autorii promit îmbunătățiri de aproximativ 30% față de sumatoarele convenționale, cu precizie totală. De asemenea, propun și o variantă pipelineizată a acestui sumator.

Autorii și-au propus să realizeze un sumator cu precizie reglabilă dinamic, astfel încât sa poată opera atât în mod imprecis cât și cu precizie totală. Ei motivează această țintă prin faptul că detecția și corectarea unei erori a rezultatului furnizat de un sumator imprecis nu aduce o încărcare suplimentară foarte mare, dar această dublă utilizare a aceluiași sumator salvează utilizarea de resurse necesare pentru un al doilea sumator.

Pentru a-și testa prototipul, au descris arhitectura folosind Verilog și l-au sintetizat pe o placă care folosește tehnologic TSMC 65GP. De asemenea, au efectuat simulări la nivel de poartă logică folosind Cadence NC-Sim. O parte din datele de intrare pentru teste au fost generate aleatoriu, iar cealaltă parte a fost preluată din benchmark-urile oferite în SPEC 2006 [57].

BUPT

Pentru a-și cuantiza rezultatele, autorii au folosit două metrici: rata erorii(ER), care reprezintă procentul de rezultate diferite față de rezultatele corecte din numărul total de execuții, precum și eroarea absolută(ES) care reprezintă modulul diferenței dintre dintre rezultatul obținut și valoarea corectă. Ca și criteriu de acceptabilitate al rezultatului au utilizat metrica propusă de [58]: $ES \times ER \geq acceptancethreshold$, unde $acceptancethreshold$ e specific aplicației. Ei afirmă ca această metrică e utilă în special în aplicațiile de procesare a imaginilor sau a sunetelor.

De asemenea, au ales să utilizeze sumatorul imprecis într-o aplicație reală: un filtru Gaussian de netezire. Pentru a putea compara rezultatele obținute folosind diferite sumatoare, au folosit metrica PSNR(en: Peak Signal-to-Noise Ratio) și au măsurat consumul de putere. Astfel, comparând rezultatele ei afirmă că sumatorul propus poate fi folosit în aplicații reale, furnizând o economie de aproximativ 50 % a puterii consumate, însă cu o mică pierdere a calității imaginii.

Concluzia finală a autorilor este că sumatorul propus oferă o îmbunătățire de aproximativ 24.6% a vitezei de execuție și de 37% față de un sumator precis convențional.În același timp, varianta pipeline-izată oferă o îmbunătățire de aproximativ 30% a puterii consumate față de un sumator precis, convețional, pipeline-izat. De asemenea, se susține că sumatorul imprecis propus asigură un compromis mai bun între performanță, puterea consumată și precizia rezultatelor față de alte sumatoare imprecise pentru aceleași cerințe de precizie.

Un alt concept al unui sumator imprecis pentru aplicații tolerante la erori din domeniul procesării semnalelor digitale este prezentat în [59]. Autorii susțin îmbunătățiri de până la 65 % al produsului putere-întârziere. Autorii precizează că un astfel de sumator imprecis reprezintă o soluție viabilă pentru a rezolva compromisul între puterea consumată și viteza de execuție a unui sumator precis. Ei adaugă că principala problemă a întârzierilor pentru un sumator precis este propagarea carry-ului de-a lungul căii critice, iar principala cauză a consumului ridicat de putere o reprezintă tranzițiile biților de carry în timpul propagării; astfel, susțin că dacă se scurtează sau se ajustează propagarea carry-ului se obține o îmbunătățire semnificativă a vitezei de execuție sau a consumului de putere.

Pentru a-și testa prototipul, autorii au ales să simuleze folosind HSPICE sumatorul propus, împreună cu alte patru sumatoare convenționale. Intrările au fost obținute prin generarea de 100 de seturi de numere aleatoare; aceste seturi de date au fost furnizate fiecăruia din cele cinci sumatoare și s-au urmărit puterea consumată, întârzierea, produsul putere-întârziere precum și numărul de tranzistoare folosite. Au putut astfel observa o îmbunătățire de cel puțin 66 % a produsului putere-întârziere față de sumatoarele convenționale și chiar un număr mic de tranzistoare utilizate; doar un sumator convențional folosea mai

BUPT

puține tranzistoare.

Pe lângă această simulare, au ales să implementeze și o transformată rapidă Fourier utilizând sumatorul imprecis propus, aplicată într-un algoritm de prelucrare a semnalelor digitale care reprezintă o imagine. S-a ales acest algoritm deoarece conține un număr foarte mare de adunări și înmulțiri; astfel, au înlocuit toate sumatoarele convenționale cu câte o instanță a celui imprecis, conceput de ei. Deși prezintă o comparație vizuală a imaginii rezultate cu imaginea originală, nu furnizează nici o metrică cu care să analizeze diferențele dintre cele două imagini.

În final, ca și concluzie, subliniază îmbunătățirile aduse de acest sumator imprecis ca și consum de putere și viteză de execuție, precum și posibilitatea de a-l utiliza în aplicații reale, fără o pierdere prea mare a calității produsului final.

Jianghang și ceilalți [60] propun o serie de metrici pentru evaluarea sumatoarelor imprecise. Autorii împart sumatoarele imprecise în două categorii: sumatoare probabilistice și sumatoare aproximative. Cele aproximative se obțin prin alterarea tabelei de adevăr a semnalelor de intrare, iar cele probabilistice se obțin prin utilizarea de circuite CMOS probabilistice, în special pentru biții mai puțini semnificativi ai sumatorului.

Având în vedere cele două categorii de sumatoare, propun metrici care să acopere ambele categorii. Astfel, ei propun metrica numită SPTMs(Matrici probabilistice și secvențiale de Transfer) care presupune calcularea unei matrici în care fiecare poziție (i, j) din matrice reprezintă probabilitatea ca vectorul de ieșire cu valoarea j să fie obținut de vectorul de intrare cu valoarea i. În continuare, prezintă eroarea absolută ca valoarea absolută a diferenței între o ieșire imprecisă și valoarea așteptată la acea ieșire pentru niște intrări date. Pe baza erorii absolute definesc media erorilor absolute și eroarea absolută normalizată. De asemenea, susțin că media erorilor absolute este relevantă pentru evaluarea sumatoarelor de dimensiuni diferite, în timp ce eroarea absolută normalizată este aproape constantă, nevariind cu dimensiunea operatorilor și, astfel, poate fi folosită pentru compararea diferitelor implementări. În final, afirmă că produsul dintre puterea consumată și eroarea absolută normalizată reprezintă o metrică bună pentru calcularea compromisului dintre putere și precizie.

Pe baza metricilor definite, autorii afirmă că sumatoarele aproximative sunt mai eficiente din punct de vedere al energiei consumate, dar cu o pierdere de precizie mai mare, pe când la cele probabilistice, situația e opusă: se obține o precizie mai bună, dar cu un consum mult mai mare de putere.

O altă abordare pentru îmbunătățirea vitezei de execuție a unor aplicații software este prezentată în [61]. Autorii prezintă o căutare stocastică de numere reprezentate în virgulă flotantă folosită pentru optimizarea aritmeticii cu numere în virgulă flotantă. Autorii susțin că au reușit să genereze implementări

BUPT

cu precizie redusă a bibliotecii numerice scrise manual de Intel, implementări care sunt de până la șase ori mai rapide și cu aproximativ 30 % mai rapide la rularea completă a algoritmilor de test; algoritmii aleși suportă rezultate imprecise.

Pentru implementare au extins STOKE [62], o bibliotecă software pentru optimizarea stocastică pentru secvențe fără bucle a programelor care conțin numere în virgulă flotantă. Pentru a-și valida și verifica implementarea, au ales să o compare cu biblioteca numerică `math.h`, să ruleze un algoritm numeric de simulare pentru motoare care au aprindere prin încărcare omogenă și compresie internă, precum și să ruleze un algoritm complet funcțional de ray tracing.

Concluziile autorilor se referă în primul rând la aplicabilitatea în aplicații reale a bibliotecii de optimizare propusă. În același timp, reiau performanțele care au fost descrise la început și mai apoi dovedite în conținutul lucrării.

În [63] s-a creat un procesor construit pe tehnologie CMOS de 65 nm, procesor care include o unitate de virgulă mobilă adaptată pentru calcule imprecise. Această unitate de virgulă flotantă e compusă din patru subunități: trei imprecise, fiecare implementând un alt algoritm, și una precisă. Cele patru subunități de virgulă flotantă pot fi activate pe rând, astfel încât fiecare poate fi folosită la un moment dat.

Pentru a-și testa procesorul creat, autorii au implementat un algoritm de procesare al imaginilor pe care l-au rulat folosind procesorul creat. Ei au rulat algoritmul cu fiecare unitate de virgulă flotantă activată la un moment dat, obținând astfel rezultate pentru fiecare în parte, în același context hardware. Pentru a compara imaginile, au folosit metrica de PSNR(en: Peak Signal-to-Noise Ratio), obținând valori între 76.4 dB și 127.3 dB pentru unitățile imprecise relativ la unitatea precisă. În același timp au folosit și diferența între fiecare canal de culoare al imaginii rezultate prin utilizarea unității precise de virgulă flotantă și al imaginilor rezultate folosind unitățile imprecise.

Măsurătorile efectuate în urma rulării algoritmilor folosind câte una din subunități au scos la iveală o reducere a consumului de putere cu până la 27 %, a suprafeței folosite cu până la 36 %, iar a produsului putere suprafață cu 53 %. De asemenea, au observat că diferențele între fiecare canal al fiecărui pixel rămân centrate pe 0 și au valori mici.

În continuare, Grigorian și ceilalți [64] prezintă o serie de metrici specifice fiecărei aplicații. Ei susțin că, spre deosebire de metricile care compară soluțiile precise cu cele imprecise, aceste metrici pot fi ajustate dinamic pentru analiza erorilor și corectarea lor. De asemenea, susțin că ele garantează o eroare maximă la nivel de aplicație. Aceste metrici se calculează dinamic, în timpul execuției aplicației.

Algoritmul prezentat are următorii pași:

1. utilizează acceleratorul imprecis

BUPT

2. calculează metricile propuse și determină calitatea serviciului
3. dacă constrângerile de calitate sunt îndeplinite, continuă
4. altfel, folosește o unitate precisă pentru a reprocesa intrarea

Autorii precizează de asemenea că partea de calculare a metricilor se efectuează o singură dată, algoritmul nefiind o buclă. De asemenea, afirmă că performanța per ansamblu a aplicației este influențată decisiv de performanța și acuratețea acceleratorului imprecis, de constrângerile de calitate, dar și de natura intrărilor care sunt procesate.

Pentru a-și testa algoritmul, autorii au instrumentat un algoritm de control al unui braț robotizat având trei articulații cu propriul algoritm. Astfel, au folosit o rețea neuronală, antrenată în prealabil cu 7500 de intrări-ieșiri. Pentru evaluare, au folosit 1024 de intrări diferite de cele din setul de învățare, obținând o eroare medie de 4.1 % cu o deviere standard de 3.6 %. În final, autorii afirmă că algoritmul lor reprezintă o soluție elegantă de analiză a erorii și de revenire în caz de nevoie, în timpul execuției.

Din analiza articolelor prezentate anterior, am putut deduce o serie de îmbunătățiri propuse pentru diferite tipuri de operatori imprecisi sau chiar la nivel de aplicație, însă nicăieri nu se propun metrici care să indice dacă o imagine obținută în urma utilizării unui algoritm imprecis este acceptabilă vizual față de o imagine de referință, obținută folosind un algoritm precis. Cu alte cuvinte să prezinte o metrică care să determine dacă un rezultat obținut în urma utilizării unui algoritm imprecis este acceptabil în locul unui produs al unui algoritm exact, în special produse care sunt evaluate de simțuri ale uni om. Nu se găsesc nici metrici din alte domenii care la prima vedere să poată fi folosite pentru rezolvarea problemei ridicată mai sus.

Vom continua cu analiza realizărilor din domeniul procesării de imagini, în care prezentăm, în special, metrici pentru compararea a două imagini similare.

Astfel, în [65], autorii propun o metrică de similaritate bazată pe valori proprii pentru compararea a două imagini în nuanțe de gri.

Ei prezintă modul de calculare a metricii pe baza unei matrici de corespondență la care îi asociază o matrice de covarianță și căreia îi calculează valorile proprii. Astfel, valoarea celei mai mici valori proprii dă valoare metricii de similaritate propusă.

Pentru a-și prezenta acuratețea metricii de similaritate propuse, au folosit imagini de 400x400 pixeli, cu câte opt biți per pixel. De asemenea, analizează influența dimensiunii ferestrei de vizualizare și a iluminării asupra valorilor metricii. Concluzia este că două imagini identice au matricea de similaritate diagonală, iar valoarea metricii de similaritate este aproape 0. Cu cât diferența este mai mare, cu atât forma matricii de similaritate deviază de la diagonale, iar valoarea metricii crește. În final, susțin că puterea de discriminare a matricii de similaritate propuse este superioară metricilor tradiționale deja existente și

BUPT

se pretează a fi folosită în detectarea defectelor de fabricație a produselor cu o suprafață complicată pe o bandă de producție, cum ar fi plăcile cu circuite imprimate.

Această metrică ar putea fi folosită pentru stabilirea acceptabilității unei imagini imprecise relativ la imaginea de referință prin stabilirea unui prag de acceptabilitate cu care să fie comparată valoarea rezultată a metricii pentru cele două imagini.

O altă metrică de evaluare a similarității dintre două imagini este prezentată în [66]. Autorii au dezvoltat o nouă metrică pornind de la măsura îmbunătățirii prin entropie(en: measurement of enhancement by entropy - EME). Fiind bazată pe EME, își denumesc metrica EME pentru similaritate(SEME).

Și ei ridică problema evaluării calității unei imagini alterate relativ la imaginea originală, însă afirmă că doar o evaluare umană, subiectivă poate hotărî calitatea unei astfel de imagini. Scopul prezentat de autori este definirea unor metrici care să poată prezice calitatea percepută de către un utilizator uman a unei imagini. Astfel, încearcă formularea unui algoritm de calculare al unei metrici care să țină cont de caracteristicile cunoscute ale sistemului vizual uman. Algoritmul vizat trebuie să fie consistent și rapid în același timp.

Pentru a-și valida metrica, autorii au încercat algoritmul propus pe mai multe seturi de date. Astfel, prezintă mai multe tipuri de alterare a imaginii, cum ar fi schimbări de contrast, compresie JPEG, încețoșare, și altele; pentru fiecare tip de alterare calculează patru metrici: MSE, SEME, SSIM și PSNR. Astfel, observă că MSE și PSNR au valori similare și aproape constante pentru toate tipurile de alterare. De cealaltă parte, SSIM și SEME ordonează clar imaginile în aproximativ aceeași ordine. Ca și bază de date pentru imagini, au folosit baza de date creată de Laboratorul de Inginerie pentru Imagini și Video al "University of Texas" din Austin [67]. Această bază de date conține 233 de imagini evaluate de utilizatori umani. Astfel, autorii au putut să-și compare valorile metricii pentru aceste imagini cu cele furnizate de utilizatorii umani.

În cele din urmă, pe baze celor demonstrate în articol, ei trag concluzia că algoritmul pentru calcularea metricii propuse este mai rapid decât cele existente la vremea respectivă de cel puțin două ori și este cel puțin la fel de precis comparativ cu metricile asemănătoare.

În [68] autorii analizează două metrici obiective, utilizate pe scară largă pentru aprecierea calității imaginilor: maximul raportului dintre semnalul util și zgomot (en: "Peak Signal to Noise Ratio" - PSNR) și metrica de similaritate structurală (en: "Structural Similarity Index Measure" - SSIM). Ei au reușit să ofere o relație matematică între cele două metrici prin referire la legătura cu MSE(en: Mean Squared Error).

Cu toate că au dedus o legătură matematică între cele două metrici, autorii afirmă că acest lucru nu indică faptul că o metrică este mai sensibilă decât cea-

BUPT

laltă la alterări ale unei imagini. Astfel, au calculat individual fiecare metrică pentru 76 de imagini din baza de date Kodak, imagini care au fost alterat utilizând algoritmi pentru zgomot Gaussian, încețoșare Gaussiană, compresie jpeg și respectiv jpeg2000. Imaginile din baza de date sunt formate din 512 x 768 sau 768 x 512 pixeli, fiecare cu 24 de biți, câte opt biți pentru fiecare canal de culoare. Astfel, fiecare imagine a trebuit transformată în nuanțe de gri înainte de a putea calcula cele două metrici.

Concluzia lor este că fiecare metrică este mai sensibilă decât cealaltă la diferite tipuri de transformări. Totodată, legătura matematică dedusă teoretic între cele două metrici a fost confirmată prin rezultatele practice.

O nouă metrică pentru evaluarea similarității a două imagini este prezentată în [69]. Autorii au denumit-o FSIM (din en: Feature Similarity Index Metric). Ei au conceput această metrică pe având în vedere că sistemul vizual uman percepe o imagine în special pe baza caracteristicilor de nivel scăzut. În continuare, afirmă că metrici ca PSNR sau MSE țin cont în mod direct de luminozitatea imaginii și nu o corelează bine cu evaluările subiective; de aceea au decis să creeze o metrică din domeniul aprecierii calitative a imaginilor. Metrica propusă o calculează pe baza matricii de congruență de fază, dată de maximul fazei componentelor Fourier și pe baza hărții mărimii gradientului.

Metrica FSIM este concepută în special pentru imagini în nuanțe de gri, dar a fost extinsă și pentru imagini cu trei canale de culoare(roșu, verde și albastru). Ei au folosit șase baze de date cu imagini folosite în comunitatea aprecierii calitative a imaginilor pentru a-și evalua metrica propusă. Astfel, au comparat rezultatele cu alte șapte metrici asemănătoare, folosite pe scară largă, printre care PSNR și SSIM.

Concluzia finală a autorilor este că atât FSIM, cât și varianta pentru canale de culoare ($FSIM_C$) prezintă o prestație robustă pentru toate cele șase baze de date, precum și pentru oricare tip de transformare a imaginii. De asemenea, $FSIM_C$ obține rezultate mai bune decât FSIM, pentru că toate imaginile din baza de date au canale de culoare.

În [70] se prezintă o enumerare și o analiză detaliată a celor mai folosite metrici pentru aprecierea vizuală a calității unei imagini. Astfel, autorii prezintă natura subiectivă a aprecierii calității unei imagini și fac o analiză a metricilor în funcție de diferitele transformări suferite de imaginea de referință. De asemenea, indică și influența schimbării mediului de vizualizare în calitatea percepută. Totodată prezintă și câteva baze de date publice cu imagini care pot fi folosite pentru verificarea și validarea metricilor din această categorie.

BUPT

2.2.3 Concluzii preliminare

Din articolele prezentate anterior, am putut deduce că cercetarea în domeniul calculelor imprecise s-a dezvoltat semnificativ. Au fost astfel concepute îmbunătățiri și soluții complete pentru consumul de putere și viteza de execuție prin renunțarea la precizia operanzilor folosiți, acolo unde aplicațiile tolerează rezultate imprecise. S-au propus foarte multe îmbunătățiri hardware, dar și îmbunătățiri la nivel de optimizare a codului compilat. În studiile menționate am identificat ca modalitate de îmbunătățire principală utilizarea unor operatori aritmetici deja existenți, care au fost modificați prin renunțarea la câțiva biți din precizia componentelor interioare sau prin renunțarea la unii tranzistori din componentele interioare, chiar dacă ofereau același număr de biți.

Acești operatori impreciși au fost validați și verificați în special pe aplicații din domeniul graficii pe calculator, în procesarea de imagini, datorită specificului acestor gen de aplicații, în care aprecierea rezultatului final se face de către un utilizator uman, fiind astfel subiectivă. De aceea, în foarte puține studii am găsit o evaluare matematică a pierderii de precizie a imaginii finale, rezultate în urma utilizării unor operatori matematici impreciși relativ la o imagine precisă, obținută în urma folosirii unor operatori convenționali, preciși.

Din această cauză, în continuare am căutat unele metrici din domeniul procesării de imagini, utilizate în special la algoritmii de prelucrare cu pierdere a imaginilor (de exemplu compresie/ decompresie cu pierdere) care să ajute la evaluarea degradării unei imagini față de o imagine inițială, de referință. Am găsit astfel o serie de metrici foarte utilizate, dar și unele recent propuse care să ajute la compararea a două imagini. Astfel, unele metrici încearcă să țină cont de caracteristicile sistemului vizual uman, iar altele utilizează distribuții ale erorilor.

în final, vom încerca în capitolele următoare să elaborăm o metrică proprie, care să fie simplu de calculat și care să aibă o semantică clară. Astfel, valorile ei vor putea fi rapid calculate pentru un volum mare de imagini alterate. De asemenea, prin semantica introdusă, ea va ajuta la înțelegerea gradului de modificare al imaginii relativ la imaginea de referință.

BUPT

Capitolul 3

Noțiuni Preliminare

3.1 Pipeline grafic

Un pipeline grafic constă dintr-o succesiune de stagii în care se prelucrează specializat datele de la intrare, astfel încât să obținem în final o reprezentare bidimensională, cât mai aproape de realitate a unui spațiu tridimensional. Fiecare stagiu aplică o serie de transformări geometrice afine asupra datelor primite la intrare astfel încât să producă imaginea finală, proces asemănător cu ceea ce se întâmplă într-un aparat de fotografiat.

Într-un pipeline grafic, se folosesc aceste transformări geometrice pentru a transforma puncte ale reprezentării tridimensionale în puncte bidimensionale care pot fi afișate pe o suprafață plană. Pe lângă acestea, există și informații de culoare care trebuie să facă parte din imaginea finală. Ca și date de intrare din care să rezulte culori se pot preciza texturile precum și informațiile de culoare asociate punctelor de intrare. Și pentru acestea există anumite transformări și calcule matematice ce sunt efectuate de unități specializate, în paralel cu transformările de coordonate. Pentru a înțelege mai bine reprezentarea datelor de intrare într-un pipeline grafic, figura 3.1 prezintă reprezentarea, sub formă de rețea de poligoane, a unui cangur.

OpenGL ES 1.1 [72] reprezintă specificația unei interfețe cu ajutorul căreia poate fi controlat un pipeline grafic. În această specificație, poligoanele prin care se poate specifica un obiect sunt triunghiurile; aceste triunghiuri se descriu ca o succesiune de vârfuri, iar coordonatele de poziție ale fiecărui vârf sunt reprezentate printr-un vector tridimensional $v_i = (x, y, z)$. Pe lângă poziție, un vârf poate avea atașate și alte informații, cum ar fi culoare, coordonatele vectorului normal sau chiar coordonate de textură. În continuare ne vom referi doar la coordonatele de poziție, precum și la transformările geometrice aplicate acestora.

BUPT

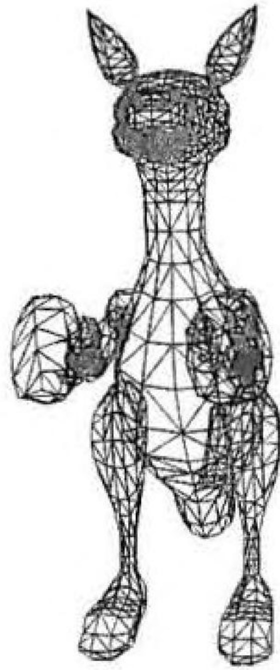


Figura 3.1: Rețea de poligoane [71]

Pentru stagiile pipeline-ului grafic aferente transformării coordonatelor de poziție ale triunghiurilor suprafeței de intrare în puncte proiectate pe o suprafață bidimensională, specificația OpenGL ES 1.1 prevede următoarele transformări, prezentate în figura 3.2.

Astfel, primele două stagii presupun înmulțiri cu matrici ale datelor de intrare, pe când celelalte două stagii implică doar operații aritmetice de bază precum adunare, scădere, înmulțire și împărțire.

O primă idee de optimizare a acestor stagii o reprezintă utilizare mixtă a unor formate de reprezentare a numerelor reale. Abstractizând, fiecare astfel de stagiu are două intrări și produce o ieșire:

→ I_0: coordonatele vârfurilor triunghiurilor reprezentând suprafața dată la intrare, respectiv datele produse de stagiul precedent

→ I_1: matricile, respectiv parametrii folosiți în fiecare stagiu pentru proiecția coordonatelor tridimensionale inițiale în planul bidimensional final

← O: datele prelucrate pe baza celor două intrări prezentate anterior

Având în vedere că intrarea I_0 este conectată la ieșirea O a stagiului precedent, putem stabili două reprezentări diferite ale numerelor reale folosite la

BUPT

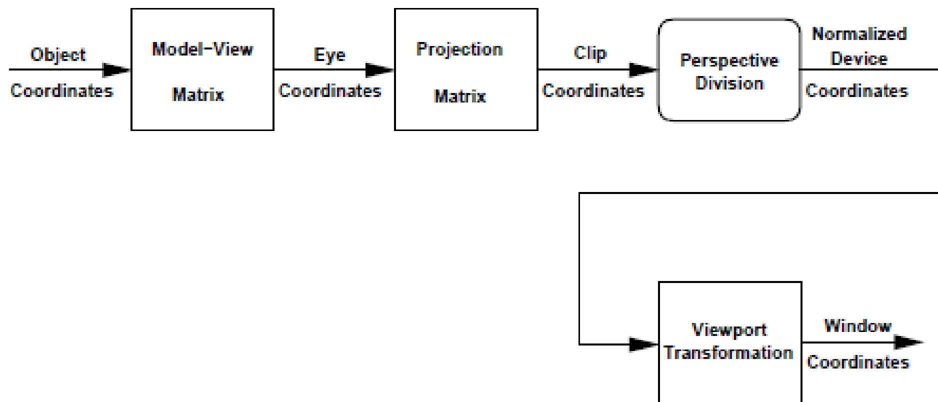


Figura 3.2: Transformări de Coordonate [72]

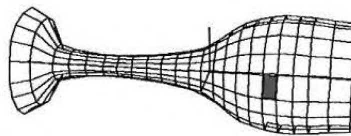


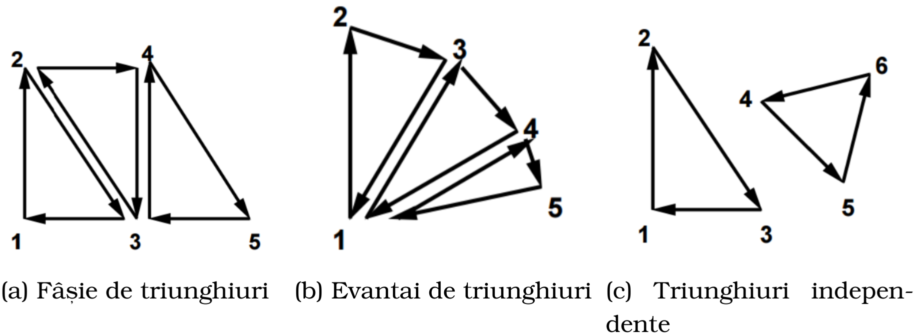
Figura 3.3: Suprafață descrisă print-o rețea de poligoane [71]

intrările I_0 și I_1. Utilizând interfața specificației OpenGL ES 1.1, putem afirmă că parametrii de configurare vor avea o reprezentare, iar datele vor folosi o altă reprezentare.

Ceea ce ne propunem este să studiem cum vor influența rezultatul final diferite combinații și variații ale multiplelor formate de reprezentare ale numerelor reale în cadrul acestor stagii. Pentru a putea urmări această variație, definim imaginile de referință ca imagini obținute în urma folosirii unor formate cu precizie ridicată. Toate celelalte imagini obținute în urma utilizării unor alte formate vor fi comparate cu cele de referință obținute pentru aceleași date de intrare precum și aceiași parametrii de configurare ai pipeline-ului.

În secțiunea următoare vom prezenta câteva formate folosite pentru reprezentarea numerelor reale de-a lungul pipeline-ului grafic utilizat pentru realizarea studiului.

BUPT



(a) Fâșie de triunghiuri (b) Evantai de triunghiuri (c) Triunghiuri independente

Figura 3.4: Rețea de triunghiuri [72]

3.2 Reprezentarea obiectelor tridimensionale

Pentru a modela obiecte tridimensionale, în grafica digitală, una dintre cele mai răspândite metode este reprezentarea suprafeței obiectului printr-o rețea de poligoane [71]. Astfel, rețeaua de poligoane este o aproximare a suprafeței obiectului de interes, cu atât mai precisă cu cât numărul de poligoane este mai mare.

Pentru a reprezenta o astfel de rețea de poligoane, există mai multe abordări, dar cea folosită de specificația OpenGL ES 1.1 este reprezentarea poligoanelor prin enumerarea explicită a vârfurilor, muchiile fiind implicite. Poligonul ales este triunghiul și astfel putem avea mai multe moduri pentru a specifica o suprafața printr-o rețea de poligoane, conform acestei specificații [72]:

- printr-o fâșie de triunghiuri: Fig. 3.4a. Un triunghi este definit de câte trei vârfuri consecutive, fiecare vârf făcând parte din câte trei triunghiuri.

- printr-un evantai de triunghiuri: Fig. 3.4b. Un triunghi este definit de primul vârf din listă și de câte alte două vârfuri consecutive.

- prin triunghiuri independente: Fig. 3.4c. Fiecare triunghi este definit de câte trei vârfuri care fac parte doar din triunghiul pe care îl descriu.

Toate aceste moduri de reprezentare a suprafețelor prin rețele de triunghiuri au în comun lista de vârfuri care trebuie procesată pentru a obține proiecția pe o suprafața bidimensională care poate fi afișată pe un display. Pentru a transforma aceste vârfuri se folosesc din spațiul tridimensional al obiectului în spațiul bidimensional al ferestrei de afișare se folosesc transformări geometrice afine.

BUPT

3.3 Transformări de Coordonate

OpenGL ES 1.1 [72] este o specificație a unui pipeline grafic fix, simplificat pentru domeniul sistemelor încorporate. Pentru acest pipeline grafic, datele de intrare sunt vârfurile ce alcătuiesc rețeaua de triunghiuri care descrie suprafața unui obiect vizualizat. Aceste vârfuri sunt prelucrate prin aplicarea unor transformări parametrizate de anumiți factori de vizualizare, rezultatul final fiind o mulțime de puncte care pot fi vizualizate pe o suprafață bidimensională. Procesul este analog proiecției unor obiecte din lumea reală pe hârtie fotografică.

Figura 3.2 prezintă sistemele de coordonate prin care trece fiecare vârf al triunghiurilor date ca intrare până ajung să fie transformate în coordonate ale ferestrei de vizualizare. După cum se poate observa, primele două din cele patru operații de transpunere presupun înmulțiri de matrici cu vectori.

Matricea de modelare este matricea care se aplică vârfurilor în coordonate obiect și ea rezultată în urma compunerii translației, rotației și scalării. După cum se va putea remarca din secțiunea 3.4, această compunere constă din înmulțirea matricilor aferente transformărilor individuale.

Următoarea matrice care apare în transformare de coordonate este matricea de proiecție. Elementele ei se obțin din parametrii camerei prin care se realizează vizualizarea obiectelor.

Ultimele două etape de transformări nu presupun operații matrice-vector ci operații aritmetice pe elementele vectorilor de poziție obținuți până în această etapă.

3.4 Transformări Geometrice Afine

3.4.1 Cazul general

Transformările geometrice afine reprezintă funcții care transpun puncte dintr-un spațiu afin în puncte dintr-un alt spațiu afin, păstrând coliniaritatea și proporționalitatea distanțelor [73]. Câteva astfel de transformări sunt translația, scalarea, rotația și reflecția. Principalul avantaj al transformărilor afine este că prin compunerea mai multor astfel de funcții se obține tot o transformare afină.

Unul dintre cele mai folosite moduri de reprezentare al transformărilor afine este folosirea matricilor și vectorilor în coordonate omogene [73]:

$$\begin{bmatrix} \vec{y} \\ 1 \end{bmatrix} = \begin{bmatrix} A & \vec{b} \\ 0, \ldots, 0 & 1 \end{bmatrix} \times \begin{bmatrix} \vec{y} \\ 1 \end{bmatrix} \tag{3.1}$$

Compunerea mai multor transformări afine presupune înmulțirea matricilor

BUPT

asociate fiecărei transformări individuale [72]:

$$C = T_n \times \ldots \times T_3 \times T_2 \times T_1 \tag{3.2}$$

Ecuația 3.2 presupune că T_1 va fi prima transformare aplicată, apoi T_2 și așa mai departe până când ultima transformare T_n a fost aplicată. Acest lucru este neintuitiv deoarece înmulțirea se execută de obicei de la stânga la dreapta, dar din moment ce înmulțirea este asociativă, transformările descrise de ecuația 3.2 aplicate unui vector $\vec{x}$ se pot reprezenta și astfel:

$$C \times \vec{x} = (T_n \times \ldots (\times T_3 \times (T_2 \times (T_1 \times \vec{x}))\ldots) \tag{3.3}$$

fiind astfel clar de ce T_1 este prima transformare aplicată vectorului $\vec{x}$, T_2 este a doua și așa mai departe.

3.4.2 Translația

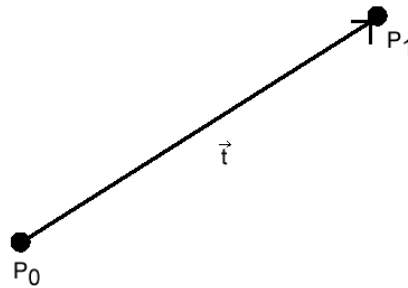


Figura 3.5: Translația într-un spațiu bidimensional

Un exemplu de translație, într-un spațiu bidimensional, a punctului P_0 cu vectorul $\vec{t}$ poate fi observată în Fig. 3.5. Astfel, avem:

$$P_1 = P_0 + \vec{t} \tag{3.4}$$

Transpunând ecuația 3.4 în formă matricială, obținem:

$$\begin{bmatrix} P_1x \\ P_1y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & o & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} P_0x \\ P_0y \\ 1 \end{bmatrix} \tag{3.5}$$

Generalizând ecuația 3.5, matricea de translație într-un spațiu n-dimensional are următoarea formă [73]:

$$T_n = \begin{bmatrix} I_n & \vec{t_n} \\ 0,\ldots,0 & 1 \end{bmatrix} \tag{3.6}$$

BUPT

unde I_n este matricea unitate de dimensiune n și $\vec{t_n}$ este vectorul de translație de dimensiune n.

Astfel, pentru a aplica matricea de translație într-un spațiu n-dimensional vectorului $\vec{x}$, avem [73]:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} & & t_1 \\ & & t_2 \\ & I_n & \vdots \\ & & t_n \\ 0, \ldots, 0 & & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix} \tag{3.7}$$

Din ecuația 3.7, putem deduce că $y_i = t_i + x_i$, unde $i = \overline{1, n}$.

Compunerea Translațiilor

Pentru a compune o translație cu o transformare compusă, avem [73]:

$$\begin{bmatrix} & t_1 \\ & t_2 \\ I_n & \vdots \\ & t_n \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n+1} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n+1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n+1,1} & c_{n+1,2} & \cdots & c_{n+1,n+1} \end{bmatrix} \tag{3.8}$$

Din ecuația 3.8 putem extrage următoarele formule iterative:

$$r_{i,j} = c_{i,j} + t_i \cdot c_{n+1,j}, \{i, j\} \in \overline{1, n} \times \overline{1, n+1} \tag{3.9}$$

$$r_{n+1,j} = c_{n+1,j}, j = \overline{1, n+1} \tag{3.10}$$

Ecuația 3.9 evidențiază faptul că pentru a calcula valoarea unui element din matricea rezultat este nevoie de o operație de înmulțire urmată de una de adunare, în timp ce ecuația 3.10 subliniază faptul că ultima linie a matricii rezultat e identică cu cea a matricii de intrare.

Translația într-un spațiu bidimensional

Aplicând ecuația 3.8 pentru un spațiu bidimensional, reiese că o translație 2D aplicată unei matrici de transformări compuse are următoarea formă [72]:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix} =$$

$$\begin{bmatrix} c_1 + t_x \cdot c_7 & c_2 + t_x \cdot c_8 & c_3 + t_x \cdot c_9 \\ c_4 + t_y \cdot c_7 & c_5 + t_y \cdot c_8 & c_6 + t_y \cdot c_9 \\ c_7 & c_8 & c_9 \end{bmatrix} \tag{3.11}$$

BUPT

3.4.3 Scalarea

Scalarea unui punct P_0 cu vectorul $\vec{s}$ presupune înmulțirea coordonatelor punctului cu cele ale vectorului. Geometric, scalarea semnifică mărirea sau micșorarea distanței unui punct față de origine, după cum se poate observa și în figura 3.6.

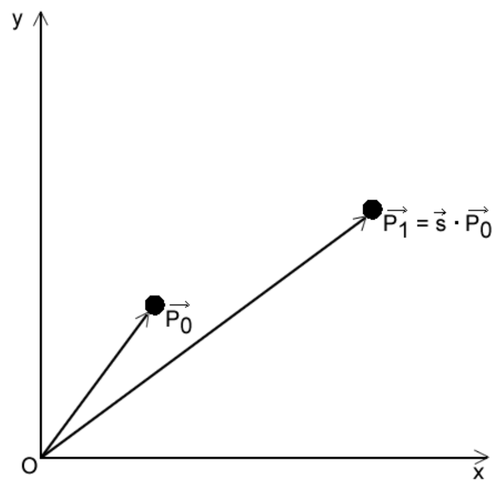


Figura 3.6: Scalarea într-un spațiu bidimensional

Astfel, avem:

$$\vec{P_1} = \vec{s} \cdot \vec{P_0} \tag{3.12}$$

Transpunând ecuația 3.12 în formă matricială, obținem:

$$\begin{bmatrix} P_1x \\ P_1y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & o & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} P_0x \\ P_0y \\ 1 \end{bmatrix} \tag{3.13}$$

Generalizând ecuația 3.13, matricea de scalare într-un spațiu n-dimensional are următoarea formă:

$$S_n = \begin{bmatrix} s_1 & 0 & 0 & \dots & 0 \\ 0 & s_2 & 0 & \dots & 0 \\ 0 & 0 & s_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \tag{3.14}$$

BUPT

unde s_1, s_2, s_3, $\ldots s_n$ sunt coordonatele vectorului de scalare $\vec{s}$ într-un spațiu n dimensional.

Astfel, pentru a aplica matrice de scalare vectorului $\vec{x}$, avem:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} s_0 & 0 & 0 & \ldots & 0 \\ 0 & s_1 & 0 & \ldots & 0 \\ 0 & 0 & s_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \\ 1 \end{bmatrix} \tag{3.15}$$

Din ecuația 3.15 reiese că $y_i = s_i \cdot x_i$, unde $i = \overline{1, n}$, ceea ce confirmă ecuația 3.12 pentru un spațiu bidimensional.

Compunerea Scalării

Pentru a compune o scalare cu o transformare compusă, avem următoarea înmulțire:

$$\begin{bmatrix} s_0 & 0 & 0 & \ldots & 0 \\ 0 & s_1 & 0 & \ldots & 0 \\ 0 & 0 & s_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{bmatrix} \times \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n+1} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n+1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n+1,1} & c_{n+1,2} & \cdots & c_{n+1,n+1} \end{bmatrix} \tag{3.16}$$

Astfel, putem observa că pentru elementele din matricea rezultat avem următoarele formule iterative:

$$r_{i,j} = s_i \cdot c_{i,j}, \{i, j\} \in \overline{1, n} \times \overline{1, n+1} \tag{3.17}$$

$$r_{n+1,j} = c_{n+1,j}, j = \overline{1, n+1} \tag{3.18}$$

Ecuațiile 3.17 și 3.18 sugerează că pentru a calcula elementele matricii rezultat avem nevoie de o singură înmulțire, iar elementele de pe ultima linia sunt identice cu cele ale matricii de intrare.

Scalarea într-un spațiu bidimensional

Particularizând ecuația 3.16 pentru un spațiu bidimensional, obținem:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix} = \begin{bmatrix} s_x \cdot c_1 & s_x \cdot c_2 & s_x \cdot c_3 \\ s_y \cdot c_4 & s_y \cdot c_5 & s_y \cdot c_6 \\ c_7 & c_8 & c_9 \end{bmatrix} \tag{3.19}$$

BUPT

3.5 Reprezentări de numere reale

În lumea procesoarelor digitale există două mari familii de reprezentări utilizate pentru numerele reale: reprezentarea în virgulă mobilă și reprezentarea în virgulă fixă. Ambele sunt reprezentări aproximative, cu un grad limitat de precizie, depinzând în primul rând de numărul de biți utilizați pentru reprezentare.

3.5.1 Virgulă fixă

Denumirea vine de la faptul că virgula are o poziție fixă, la fel ca și în reprezentarea zecimală utilizată curent. Astfel, având o reprezentare pe 32 de biți, cu 10 biți pentru partea întreagă, biții pe care este reprezentată partea întreagă are ponderi de la 2^0 la 2^9, iar biții pentru parte fracționară au ponderi de la 2^{-1} la 2^{-22}.

Generalizând, având o reprezentare cu i biți pentru parte întreagă și f biți pentru partea fracționară, se pot reprezenta numere cu o precizie de 2^{-f}, iar numărul maxim care poate fi reprezentat este $2^i - 2^{-f}$. Analog, numărul minim care poate fi reprezentat este $-(2^i - 2^{-f})$. În același timp, numerele cele mai apropiate de 0 care pot fi reprezentate sunt $\pm 2^{-f}$.

Reprezentarea în virgulă fixă a numerelor reale este foarte des întâlnită în procesoarele încorporate. Principalul motiv este reprezentat de faptul că operațiile aritmetice pot fi efectuate cu unitatea aritmetico-logică pentru numere întregi, având astfel o viteză de execuție ridicată.

3.5.2 Virgulă mobilă

Denumirea acestei reprezentări provine de la faptul că virgula nu are o poziție fixă, predefinită. Valoarea unui număr reprezentat în virgulă mobilă, definit prin standardul IEEE 754 este dată de ecuația 3.20. Astfel, se poate observa că este nevoie de următoarele câmpuri pentru a reprezenta un număr real: $semn$, $mantisa$, $exponent$ și $bias$. Pentru a limita numărul de câmpuri utilizate, s-a stabilit prin convenție că valoarea bias-ului este $b^{dimensiune(exponent)} - 1$, unde b reprezintă baza de numerație utilizată. În general, în procesoarele digitale, baza de numerație utilizată este 2.

$$x = -1^{semn} \cdot 1.mantisa \cdot b^{exponent-bias} \tag{3.20}$$

Principalul avantaj al acestei reprezentări îl reprezintă dimensiunea mare a intervalului în care pot fi reprezentate valori, spre deosebire de formatul de virgulă fixă. În același timp, pentru numerele apropiate de 0 precizia poate fi de asemenea mai ridicată decât a aceluiași număr reprezentat în virgulă fixă.

BUPT

Capitolul 4

Operatori matriciali folosiți în pipeline-ul grafic

O primă îmbunătățire propusă constă în crearea unor operatori matriciali particularizați după nevoile operațiilor matriciale specifice unui pipeline grafic 3D. Astfel, din capitolul anterior, am putut vedea că operatorii matriciali de translație și scalare au forme particulare, în care majoritatea pozițiilor sunt ocupate de valoarea zero. S-au putut astfel simplifica operațiile necesare fiecărui operator deoarece înmulțirea cu zero are ca rezultat tot zero; în plus, zero este termen neutru pentru adunare.

Principalul avantaj al unei astfel de optimizări o reprezintă păstrarea interfeței specifice unui operator matricial, nefiind necesară o adaptare software pentru utilizarea unui astfel de operator. În plus, prin simplificarea și paralelizarea operatorilor creați, se obține și o rată ridicată de procesare a datelor.

În continuare, în acest capitol am prezentat detaliile de implementare specifice acestor operatori precum și rezultatele sintezei lor.

4.1 Prezentare generală

Folosind modelele matematice prezentate în secțiunile anterioare, am dezvoltat operatori matriciali pentru aceste transformări afine. Din ecuațiile 3.11 și 3.19 am observat că matricea rezultat are ieșirile dependente doar de anumite intrări, restul fiind zero, astfel putând fi ignorate. Din cauza acestei forme particulare, este mult mai eficient din punct de vedere al performanței să fie creați operatori specializați pentru astfel de înmulțiri de matrici decât să fie folosit un înmulțitor general pentru matrici.

Pentru fiecare transformare afină, translație respectiv scalare, am izolat operația necesară calculării unei poziții din matricea rezultat. Astfel, pentru

BUPT

translație am creat un operator de înmulțire-adunare, iar pentru scalare am folosit un înmulțitor. Acești operatori de bază i-am numit unități de procesare și le-am folosit, în funcție de arhitectura concepută, de șase, două și respectiv o dată. Am făcut astfel trecerea de la o arhitectură complet paralelă, în care toate pozițiile matricii rezultat sunt calculate în același timp, la o arhitectură complet serială, în care rezultatele sunt obținute unul după celălalt.

Pentru descrierea operatorilor, am folosit FloPoCo [32]. În acest fel am descris comportamentul general al operatorilor, independent de dimensiunea operanzilor și stagiile interne, FloPoCo gestionând aceste lucruri pe baza parametrilor primiți la generare. Astfel, am putut genera cod VHDL sintetizabil pentru Xilinx Virtex V. Pe lângă codul aferent operatorilor de translație și scalare, am generat și entități de test pentru validarea acestor operatori.

Operatorii au fost concepuți pentru a avea următorii parametrii configurabili:

- Dimensiunea numerelor reale. Am folosit reprezentarea cu virgulă mobilă descrisă în specificația IEEE 754 [74], putând configura dimensiunea mantisei, precum și a exponentului.

- Frecvența țintă pentru generare. Acest parametru influențează numărul de stagii ale pipeline-ului operatorului.

- Gradul de paralelizare. Acest parametru influențează direct resursele utilizate în implementarea operatorului.

4.2 Justificare

Având în vedere faptul că FPGA-urile dau rezultatele cele mai bune atunci când algoritmii implementați cu ajutorul lor sunt particularizați pentru o anumită aplicație, am creat o arhitectură generică care poate fi particularizată pe fiecare din cele trei direcții prezentate anterior. În acest fel se obțin un număr semnificativ de operatori dintre care se poate alege cel care satisface cel mai exact constrângerile aplicației țintă. Domeniul aplicațiilor pentru care am țintit acești operatori au fost pipeline-urile grafice.

În acest mod se pot alege operatori care favorizează precizia numerelor reale folosite, în dauna vitezei de execuție, pentru aplicațiile care au nevoie de imagini finale cât mai precise. La polul opus se pot alege operatori mai puțin preciși, dar care au o viteză ridicată de execuție. În același timp, se poate varia și gradul de paralelism folosit, astfel încât să se poată regla costurile de implementare.

Între aceste extreme, se poate varia fiecare parametru configurabil astfel încât să se obțină un operator matricial cât mai aproape de necesitățile pipeline-ului grafic în care va fi utilizat.

BUPT

4.3 Implementare

În această secțiune sunt prezentate detaliile de implementare specifice fiecărui operator în parte, precum și particularitățile fiecăruia.

4.3.1 Operatorul de translație

Plecând de la ecuația 3.11, se poate observa că, pentru fiecare din cele șase elemente compuse ale matricii rezultat, este nevoie de o operație de înmulțire, urmată de o adunare. Astfel, pentru acest operator, vom denumi unitate de procesare(PU) unitatea care efectuează operația compusă de înmulțire-adunare.

Plecând tot de la ecuația 3.11, am decis să creăm trei arhitecturi de operatori, în funcție de numărul de unități de procesare folosite:

- Cu șase unități, câte una pentru fiecare operație de înmulțire-adunare corespunzătoare unei poziții din matricea rezultat.
- Cu două unități, câte una pentru fiecare linie din matricea rezultat.
- Cu o singură unitate pentru toate cele șase elemente compuse din matricea rezultat.

Operator cu șase unități de procesare

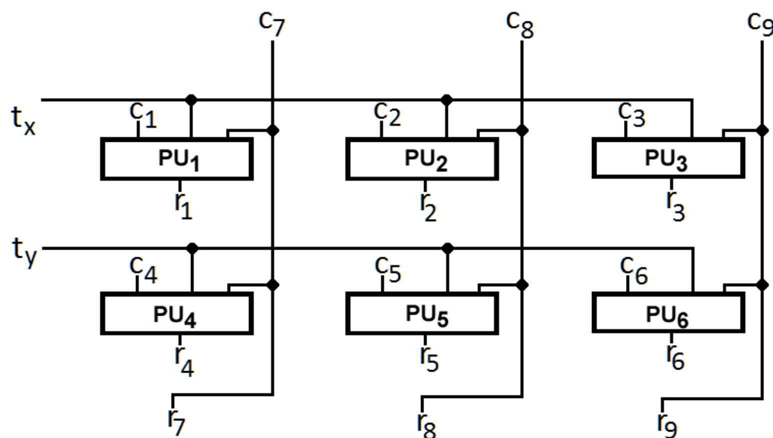


Figura 4.1: Operator de translație cu șase unități de procesare [75]

Pentru această arhitectură am ales câte o unitate de procesare pentru fiecare element din matricea rezultat, iar ieșirile care sunt identice cu intrările sunt cablate direct la acestea din urmă.

BUPT

Utilizând FloPoCo am descris arhitectura lăsând ca parametrii de ajustare a acesteia dimensiunea mantisei precum și a exponentului. În acest mod, se pot genera, pe baza modelului descris în C++, cu ajutorul infrastructurii oferite de FloPoCo, operatori de translație pentru diferite precizii de numere reprezentate în virgulă flotantă. Astfel, am ales să generăm operatori pentru următoarele dimensiuni:

- exponent 5 biți; mantisă: 10 biți(half precision float)
- exponent 8 biți; mantisă: 23 biți(float)
- exponent 11 biți; mantisă: 52 biți(double precision float)

În același timp, am folosit interfața oferită de FloPoCo pentru a construi această arhitectură pentru diferite frecvențe țintă:

- 100 Mhz
- 200 Mhz
- 300 Mhz
- 400 Mhz

Principala caracteristică influențată de frecvența țintă a fost numărul de stagii pipeline utilizate. Cu cât frecvența țintă a fost mai mare, cu atât a crescut și numărul de stagii utilizate și prin urmare și numărul de resurse utilizate pentru sintetizarea acestor operatori.

Principala caracteristică a acestei arhitecturi propuse o reprezintă gradul mare de paralelizare a operațiilor efectuate, fiind prezentă câte o unitate de procesare pentru fiecare operație de înmulțire-adunare necesară.

Operator cu două unități de procesare

Pentru această arhitectură am redus numărul de unități de procesare, utilizând doar două, câte una pentru fiecare linie a matricii rezultat unde sunt operații aritmetice. Astfel, numărul de resurse scade, însă nu direct proporțional cu numărul de unități de procesare, deoarece este necesară o unitate de control suplimentară pentru ambele unități de procesare. Această unitate de control este un numărător modulo trei care dă semnalul s, reprezentat în figura 4.2. Pe lângă această unitate de control mai sunt necesare și trei multiplexoare. Acestea au rolul de a selecta care dintre intrări sunt active pentru unitatea de procesare corespunzătoare.

În acest fel, în ciclul de tact k se vor procesa intrările c_1, c_7 și t_x, iar unitatea de procesare va produce ieșirea r_1, decalată față de intrări cu un număr de

BUPT

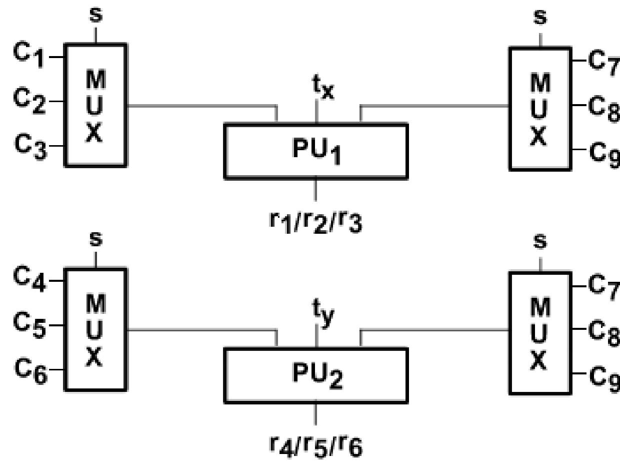


Figura 4.2: Operator de translație cu două unități de procesare [75]

cicluri de tact δ, unde δ reprezintă numărul de cicluri de tact necesare unității de procesare pentru a produce rezultatul. Acest număr este dat de adâncimea pipeline-ului unității. Analog, în ciclul $k+1$ vor fi procesate c_2, c_8 și t_x , rezultând r_2 în ciclul $k+1+\delta$. În cele din urmă, în ciclul $k+2$ vor fi procesate c_3, c_9 și t_x, rezultând r_3 în ciclul $k+2+\delta$.

Se poate astfel observa că este nevoie de două cicluri de tact suplimentare pentru a produce aceleași rezultate ca și operatorul generat după arhitectura cu șase unități de procesare.

La fel ca și operatorul cu șase unități de procesare și acesta a fost generat pentru toate cele trei precizii și pentru toate cele patru frecvențe țintă.

Operator cu o singură unitate de procesare

Acest operator a fost conceput pentru a reduce și mai mult resursele utilizate, astfel el folosește o singură unitate de procesare, trei multiplexoare și o unitate de control. După cum se poate observa și din figura 4.3, multiplexoarele au șase intrări, fiind controlate de semnalul s generat de unitatea de control. De această dată, unitatea de control este un numărător modulo șase.

Analog cu arhitectura cu două unități de procesare, în ciclul de tact k sunt necesare intrările c_1, c_7 și t_x, în ciclul $k+1$ intrările c_2, c_8 și t_x, în ciclul $k+2$ intrările c_3, c_9 și t_x, în ciclul $k+3$ intrările c_4, c_7 și t_y, în ciclul $k+4$ intrările c_5, c_8 și t_y, iar în ciclul $k+5$ intrările c_6, c_9 și t_y. Astfel, având un pipeline cu δ stagii și presupunând că fiecare stagiu durează un ciclu de tact, rezultatele vor fi disponibile în ciclurile de tact $k+\delta$, $k+\delta+1$, $k+\delta+2$, $k+\delta+3$, $k+\delta+4$ și $k+\delta+5$.

BUPT

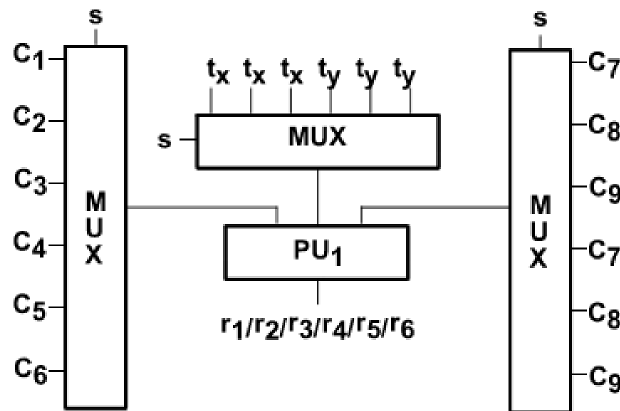


Figura 4.3: Operator de translație cu o unitate de procesare [75]

Din cele prezentate anterior, reiese că va fi nevoie de șase cicluri de tact suplimentare pentru a produce rezultatele din matricea de transformare. În toate aceste șase cicluri de tact auxiliare va fi nevoie ca intrările să fie valide pentru a obține rezultatele corecte.

În mod asemănător cu precedentele două arhitecturi s-au generat doisprezece operatori, câte unul pentru fiecare combinație de precizie și frecvență țintă.

Rezultatele sintezei

În tabelul 4.1 sunt prezentate rezultatele sintezei codului VHDL al celor 36 de operatori de translație generați pe baza arhitecturilor descrise anterior. Platforma țintă este Xilinx Virtex V.

Se poate observa că numărul de resurse folosite scade cu numărul de unități de procesare folosite, dar și cu dimensiunea operanzilor. Astfel, cele mai puține resurse sunt utilizate în cadrul operatorilor care folosesc operanzi de dimensiune 5 + 10 și doar o unitate de procesare. La polul opus, cele mai multe resurse sunt folosite pentru sintetizare operatorilor cu șase unități de procesare și care au dimensiunea operanzilor 11 + 52.

O creștere semnificativă de resurse se remarcă în cadrul operatorilor de dimensiune 11 + 52, unde sunt folosite toate modulele DSP48E disponibile. Pentru că ar fi nevoie de mai multe astfel de module, lipsa lor se suplinește prin utilizarea celorlalte tipuri de resurse pentru sintetizarea de sumatoare și înmulțitoare.

Totodată, frecvența țintă nu influențează în mod decisiv cantitatea de resurse utilizate, ea presupunând doar introducerea unor registre auxiliare pentru un număr mai mare de stagii de pipeline. Crescând numărul de stagii de

BUPT

Grad de paralelizare	Dimensiune operanzi exponent + mantisă	Frecvența țintă [MHz]	Perechi de LUT Flip Flop-uri folosite	Regiștri Slice	Slice LUTs	unități DSP48E	Frecvență maximă [MHz]
Maxim	5 + 10	100	1862	456	1824	6	185.19
Mediu			752	190	750	2	48.157
Minim			448	124	445	1	47.778
Maxim		200	2015	894	1932	6	156.51
Mediu			677	334	673	2	102.328
Minim			411	196	409	1	100.201
Maxim		300	2100	1464	1854	6	204.94
Mediu			791	562	674	2	121.036
Minim			465	292	427	1	117.965
Maxim		400	2283	1830	2100	6	210.16
Mediu			853	684	802	2	210.16
Minim			495	353	471	1	175.103
Maxim	8 + 23	100	3338	882	3275	12	150.348
Mediu			1267	364	1262	4	38.742
Minim			782	235	778	2	38.442
Maxim		200	3657	1770	3228	12	103.758
Mediu			1335	660	1257	4	103.713
Minim			797	383	758	2	103.713
Maxim		300	4413	2580	4020	12	186.409
Mediu			1574	998	1389	4	120.607
Minim			917	518	862	2	117.709
Maxim		400	5121	3768	4584	12	213.34
Mediu			1848	1394	1699	4	120.607
Minim			1048	716	981	2	117.709
Maxim	11 + 52	100	10787	3186	9726	48	90.929
Mediu			3243	1196	3203	24	76.572
Minim			1878	699	1850	12	75.493
Maxim		200	12416	6444	10770	48	159.119
Mediu			3829	2414	3525	24	77.878
Minim			2155	1242	2019	12	76.762
Maxim		300	13023	9192	11148	48	237.651
Mediu			4102	3170	3649	24	135.828
Minim			2267	1631	2081	12	132.469
Maxim		400	16383	12006	14250	48	272.365
Mediu			5190	4040	4699	24	135.828
Minim			2807	2055	2604	12	132.469

Tabela 4.1: Rezultate sinteză operator translație [75]

BUPT

pipeline, un stagiu poate avea durata mai mică și astfel frecvența poate crește.

Din punct de vedere al frecvenței maxime obținute, aceasta variază în principal din cauza frecvenței țintă care ar trebuie atinse. Din tabel se poate observa că operatorii fără unitate de control și multiplexoare au frecvențe de lucru mai ridicate. Dimensiunea operanzilor nu influențează frecvența de lucru, ea doar crește numărul de cicli de tact necesari pentru a obține rezultatele finale.

4.3.2 Operatorul de scalare

Având ca model arhitecturile operatorului de translație, am conceput, în mod asemănător, arhitecturi pentru operatorul de scalare, având ca suport matematic ecuația 3.19. Se poate ușor observa că pentru cele șase elemente compuse ale matricii rezultat este suficientă o operație de înmulțire.

La fel ca și la operatorul de translație, am creat trei arhitecturi. Existând o legătură directă între numărul de înmulțitoare și gradul de paralelism al operatorilor rezultați, putem avea următoarea împărțire:

- Operator complet paralel - cu șase înmulțitoare
- Operator serial - cu un singur înmulțitor
- Operator mixt - cu două înmulțitoare

Operator complet paralel

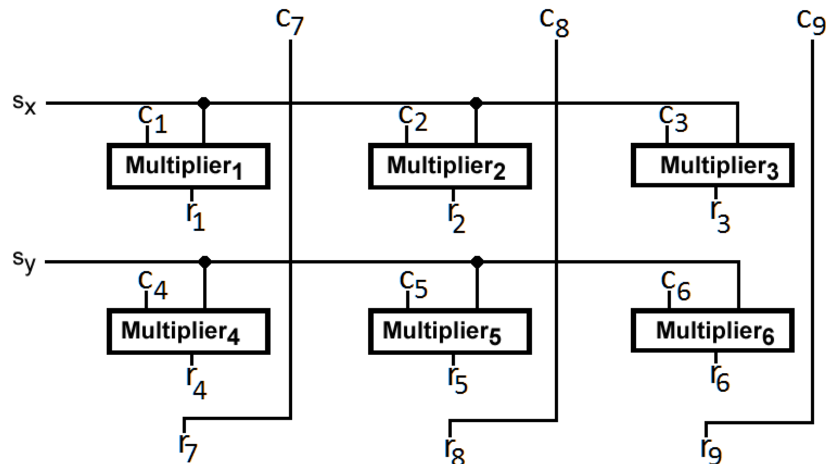


Figura 4.4: Operator de scalare cu șase înmulțitoare [76]

Pentru a efectua în paralel toate operațiile de înmulțire necesare calculării matricii rezultat am folosit șase înmulțitoare legate la intrări ca și în figura

4.4. În acest mod, toate rezultate sunt disponibile în același ciclu de tact, la δ cicluri distanță față de momentul în care intrările sunt disponibile; δ reprezintă numărul de cicluri de tact necesare pentru a efectua o înmulțire. δ variază în funcție de combinația dintre precizia utilizată pentru numerele reprezentate în virgulă flotantă și frecvența țintă. Astfel, cu cât frecvența țintă este mai mare, δ crește și, la fel, cu cât dimensiunea numerelor de virgulă flotantă crește și δ este mai mare.

Ca și în cazul operatorului de translație, au fost generați operatori pentru toate cele trei precizii de virgulă flotantă:

- exponent + mantisă: 5 + 10
- exponent + mantisă: 8 + 23
- exponent + mantisă: 11 + 52

Totodată au fost generați operatori și pentru toate cele patru frecvențe țintă:

- 100 Mhz
- 200 Mhz
- 300 Mhz
- 400 Mhz

Din combinația celor doi parametrii descriși mai sus a rezultat un număr de doisprezece posibili operatori de scalare.

Pentru toți acești operatori de scalare au fost de asemenea generate unități de testare folosind tot FloPoCo. Pentru acest lucru, a fost nevoie de descrierea modelului matematic al operatorului de scalare. În continuare, acest model a fost folosit pentru a transforma intrările date în rezultate așteptate la ieșirea operatorului. În acest fel, fiecare dintre operatorii generați a primit datele de intrare, iar datele de ieșire ale operatorului au fost validate cu datele de ieșire produse de modelul matematic. Pentru datele de intrare, FloPoCo permite specificare unor intrări cu valori fixe, care pot fi folosite pentru anumite valori de prag; în același timp permite specificarea unui număr care reprezintă cantitatea de intrări aleatoare care vor fi generate, acestea fiind distribuite uniform în intervalul valid al datelor de intrare.

Principalul avantaj al acestei arhitecturi o reprezintă gradul mare de paralelism cu care se produc rezultatele, însă acest câștig de performanță vine cu un consum puțin mai ridicat de resurse.

BUPT

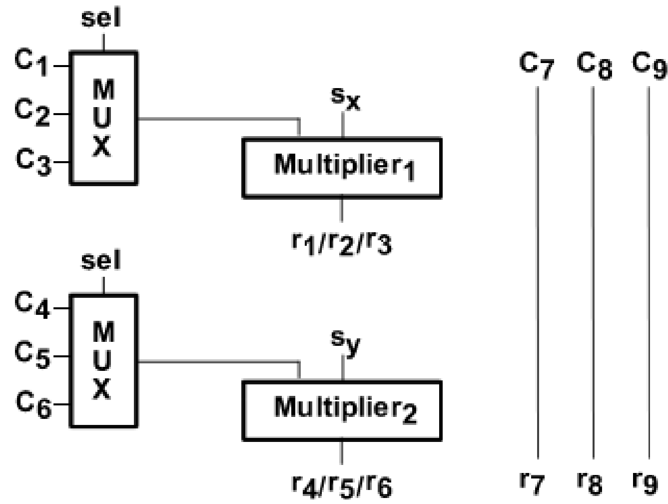


Figura 4.5: Operator de scalare cu două înmulțitoare [76]

Operator mixt

Analog operatorului de translație prezentat în subsecțiunea precedentă, operatorul de scalare mixt folosește două înmulțitoare. Acestea produc rezultatele pentru câte o linie a matricii rezultat, ultima dintre linii fiind trecută printr-un număr de regiștrii cascadați, necesari pentru întârzierea semnalului până ce vor fi disponibile și ieșirile înmulțitorului aferente intrărilor de la acel moment de timp.

Față de operatorul complet paralel, este nevoie de două cicluri de tact în plus în care intrările trebuie menținute active pentru a obține rezultatele corecte.

Spre deosebire de varianta complet paralelă s-a putut observa o scădere a vitezei de procesare, dar și o scădere a resurselor utilizate. Chiar dacă numărul de înmulțitoare s-a redus de trei ori, cantitatea de resurse folosite nu a scăzut tot de atâtea ori din cauza unității de control și a multiplexoarelor suplimentare necesare pentru coordonarea intrărilor.

Ca și în cazul arhitecturii precedente s-au generat operatori pentru toate cele doisprezece combinații posibile ale preciziei numerelor în virgulă flotantă cu frecvența țintă. Totodată au fost generate și unitățile de testare aferente.

Operator serial

În cele din urmă, operatorul serial folosește un singur înmulțitor cu care se calculează toate rezultatele pentru cele șase poziții ale matricii rezultat; pentru celelalte trei ieșiri, ca și în cazul operatorului precedent, se folosesc regiștrii.

BUPT

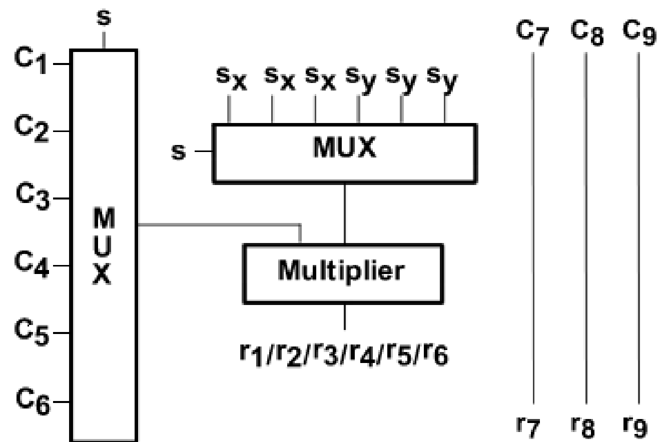


Figura 4.6: Operator de scalare cu un înmulțitor [76]

Comparativ cu operatorul complet paralel, este nevoie de cinci cicluri de tact în plus în care intrările trebuie menținute la aceeași valoare pentru a genera rezultate valide.

Față de variantele precedente, această arhitectură este cea mai lentă dar în același timp cea care utilizează și cele mai puține resurse comparativ cu configurațiile similare ale celorlalte arhitecturi.

În mod similar cu celelalte două arhitecturi prezentate pentru operatorul de scalare s-au generat doisprezece operatori aferenți combinațiilor parametrilor de configurare. Odată cu acești operatori, au fost generate și unitățile de testare corespunzătoare.

Rezultatele sintezei

În tabelul 4.2 sunt prezentate rezultatele sintezei codului VHDL al tuturor operatorilor de scalare generați pe baza arhitecturilor descrise anterior, platforma țintă fiind tot Xilinx Virtex V.

Ca și în cazul operatorilor de translație generați, se poate observa că resursele folosite sunt proporționale cu numărul de înmulțitoare folosite precum și cu dimensiunea operanzilor. Se poate astfel remarca faptul că operatorii care folosesc operanzi pe 5 + 10 biți și un singur înmulțitor consumă cele mai puține resurse; de cealaltă parte, operatorul cu operanzi pe 11 + 52 de biți și cu șase înmulțitoare folosesc cele mai multe resurse ale FPGA-ului.

Și aici lipsa numărului necesar de module DSP48e este suplinită de consumul mai mare al celorlalte resurse. Astfel, pentru surplusul de înmulțitoare necesare operatorului pe 11 + 52 de biți cu șase unități se poate ușor observa

BUPT

Grad de paralelizare	Dimensiune operanzi exponent + mantisă	Frecvența țintă [MHz]	Perechi de LUT Flip Flop-uri folosite	Regiștri Slice	Slice LUTs	unități DSP48E	Frecvență maximă [MHz]
Maxim	5 + 10	100	216	N/A	216	6	N/A
Mediu			190	128	166	2	75.302
Minim			211	147	145	1	67.116
Maxim		200	378	180	378	6	N/A
Mediu			248	190	220	2	110.417
Minim			244	178	176	1	87.879
Maxim		300	512	372	360	6	561.987
Mediu			278	234	200	2	121.211
Minim			263	200	170	1	101.334
Maxim		400	582	468	462	6	278.342
Mediu			300	266	246	2	278.342
Minim			268	216	180	1	159.505
Maxim	8 + 23	100	552	N/A	552	12	N/A
Mediu			398	240	360	4	56.920
Minim			417	275	296	2	52.019
Maxim		200	792	258	618	12	N/A
Mediu			480	328	380	4	120.964
Minim			500	319	344	2	104.845
Maxim		300	1142	588	936	12	318.977
Mediu			551	402	444	4	120.964
Minim			536	356	376	2	104.845
Maxim		400	1659	1140	1200	12	332.928
Mediu			715	586	540	4	120.964
Minim			579	448	386	2	100.792
Maxim	11 + 52	100	4746	1074	4638	48	N/A
Mediu			1630	824	1512	24	76.855
Minim			1248	714	991	12	74.369
Maxim		200	6428	2604	5058	48	159.119
Mediu			1959	1266	1564	24	78.171
Minim			1418	933	1017	12	75.600
Maxim		300	6486	3720	5526	48	286.821
Mediu			1943	1476	1694	24	136.722
Minim			1526	1048	1197	12	118.219
Maxim		400	7830	4854	6918	48	280.147
Mediu			2475	1808	2182	24	136.722
Minim			1675	1202	1326	12	118.848

Tabela 4.2: Rezultate sinteză operator scalare [76]

BUPT

ocuparea tuturor unităților DSP48e, precum și creșterea neproporțională a celorlalte resurse utilizate.

Se poate de asemenea observa că frecvența țintă influențează într-o mică măsură cantitatea de resurse utilizate. Ea influențează în mod special frecvența maximă obținută pentru un operator. Această frecvență mai ridicată nu înseamnă neapărat și o viteză de procesare mai mare a datelor, putând fi necesar un timp asemănător dacă numărul de cicli de tact de care este nevoie pentru obținerea rezultatelor este mai mare.

4.4 Concluzii preliminare

Prin implementarea arhitecturilor prezentate anterior am obținut un număr de 36 de operatori pentru fiecare transformare geometrică afină, particularizați în funcție de:

- precizia folosită
- gradul de paralelizare
- frecvența țintă

Fiecare operator obținut astfel poate fi utilizat într-o aplicație specifică, aducându-și contribuția la viteza de procesare a datelor. De exemplu, un operator cu precizie ridicată, frecvență ridicată de lucru și grad ridicat de paralelizare poate fi folosit în aplicațiile grafice care au nevoie de precizii ridicate în reprezentarea bidimensională a conținutului tridimensional, însă care au și resurse hardware consistente; o aplicație cu astfel de cerințe trebuie să ofere resurse hardware ridicate, care să acopere consumul de putere mai ridicat al unui astfel de operator, precum și suprafață mai ridicată de silicon necesară pentru implementare.

La polul opus, un operator cu precizie scăzută, frecvență scăzută de lucru și grad mai mic de paralelizare poate fi folosit într-un sistem încorporat tolerant la imprecizia reprezentării obiectelor tridimensionale. Este binecunoscut faptul că într-o astfel de aplicație consumul de putere și resursele fizice disponibile reprezintă constrângeri importante. Cu toate aceste constrângeri, folosind operatorii propuși se pot obține rezultate satisfăcătoare în obținerea unui conținut grafic acceptabil de către un utilizator uman.

Între aceste două extreme se pot utiliza oricare dintre operatorii prezentați, ținând cont de specificul fiecărei aplicații în parte, de precizia necesară în redarea conținutului precum și de costurile în care trebuie să se încadreze. Se pot astfel satisface o plajă largă de cerințe, cu o granularitate destul de mică, având în vedere numărul mare de implementări oferite pentru operatorii prezentați.

BUPT

În capitolul următor vom prezenta alte îmbunătățiri care se pot utiliza în pipeline-urile grafice imprecise, folosite în aplicații tolerante la erori, aplicații în care rezultatul grafic final suportă un anumit grad de imprecizie.

BUPT

Capitolul 5

Implementare calcule aproximative

5.1 Motivație

În prima fază a unui pipeline grafic comun [72] se procesează vârfurile poligoanelor prin care este descrisă suprafața obiectelor tridimensionale care urmează să fie proiectate pe un spațiu bidimensional. Această procesare se face prin aplicarea unor transformări geometrice afine punctelor corespunzătoare vârfurilor. Cel mai avantajos mod de reprezentare matematică a acestor transformări geometrice o reprezintă înmulțirile matriciale. Astfel, după cum am putut vedea în capitolul anterior, aceste matrice de transformare au o anumită formă particulară care poate fi exploatată pentru a simplifica și optimiza operațiile aritmetice implicate. Ne propunem să prezentăm în continuare o modificare a unei implementări software proprii a specificației unui pipeline grafic și anume Open GL ES 1.1 [72]. Am realizat această modificare pentru a putea experimenta diferite formate de reprezentări de numere reale de-a lungul pipeline-ului grafic, astfel încât să putem analiza influența pierderii de precizie a reprezentării asupra imaginii finale.

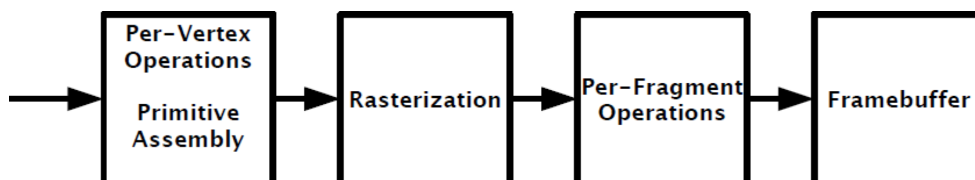


Figura 5.1: Stagiile pipeline-ului descris de OpenGL ES 1.1 [72]

BUPT

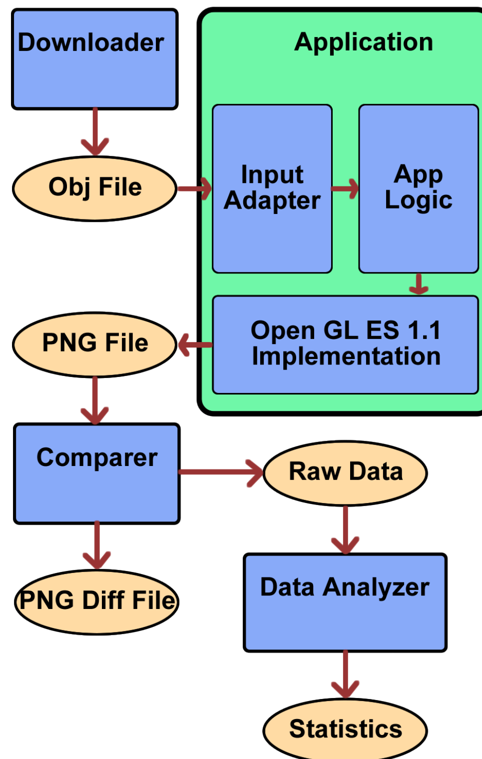


Figura 5.2: Arhitectura mediului de rulare

Principalul stagiu al unui pipeline grafic în care se execută cele mai multe operații matriciale este stagiul ”Per-Vertex Operations”(Figura 5.1). Astfel, am ales să analizăm rezultatele obținute în urma folosirii unui anumit format de reprezentare a numerelor reale în acest stagiu și un alt format în celelalte stagii.

În final, am folosit câteva metrici pentru a putea compara rezultatele obținute folosind formate imprecise cu datele obținute folosind un format precis, de referință pentru reprezentarea numerelor reale.

5.2 Mediul de rulare

Pentru a putea folosi implementarea specificației OpenGL ES 1.1 și a datelor produse de aceasta, am avut nevoie de programe software auxiliare care să colecteze date de intrare, să transforme parțial aceste date de intrare într-un format recunoscut precum și să compare și să centralizeze rezultatele obținute. Am creat astfel următoarele module software:

BUPT

- modul de colectare a fișierelor obj de pe internet
- modul pentru transformarea datelor disponibile în fișierele obj într-un format acceptat de specificația OpenGL ES 1.1
- aplicație care să folosească implementarea proprie a pipeline-ului grafic
- modul de comparare pentru imaginile rezultate care produce o imagine diferență și date statistice
- modul de centralizare, filtrare și interpretare a datelor statistice

Figura 5.2 prezintă componentele menționate anterior, precum și fluxul de date între acestea. În continuare vom prezenta detaliile specifice implementării fiecărui modul software.

5.2.1 Implementare OpenGL ES 1.1 utilizată

Pentru a putea evalua rezultatele obținute în urma folosirii de numere reale cu reprezentări mixte într-un pipeline grafic, am folosit o implementare proprie a specificației de open GL ES 1.1. Această implementare rulează pe procesorul de uz general și nu are ca scop urmărirea performanței sau a consumului de putere al unui astfel de pipeline grafic hibrid. Această implementare a fost adaptată pentru a putea fi ușor instrumentată astfel încât să se poată observa influența unor astfel de combinații de precizie asupra acurateții rezultatului final.

Scopul acestei implementări a fost acela de a evalua numărul de operații matriciale efectuate într-un pipeline grafic [77]. Ea a fost concepută astfel încât să poată fi ușor extinsă pentru a putea avea operatorii aritmetici implementați hardware pe o platformă reconfigurabilă, gen FPGA. Având în vedere aceste proprietăți ale acestei implementării, a fost facil de extins astfel încât să poată utiliza diferite formate de numere reale în diferitele puncte ale pipeline-ului grafic.

Pentru a-și îndeplini scopul pentru care a fost creată [77], această implementare nu conține părțile de texturare, iluminat precum și efectele speciale predefinite de specificația OpenGL ES 1.1, ca de exemplu încețoșarea. În schimb, partea de transformări de coordonate este complet implementată.

Plecând de la implementarea inițială, am extins și modificat astfel încât să poată fi ușor folosită pentru noua direcție de cercetare și anume măsurarea influenței operatorilor aritmetici de numere reale cu precizii diferite asupra rezultatului final al pipeline-ului grafic.

BUPT

5.2.2 Colectarea obiectelor tridimensionale folosite pentru cercetare

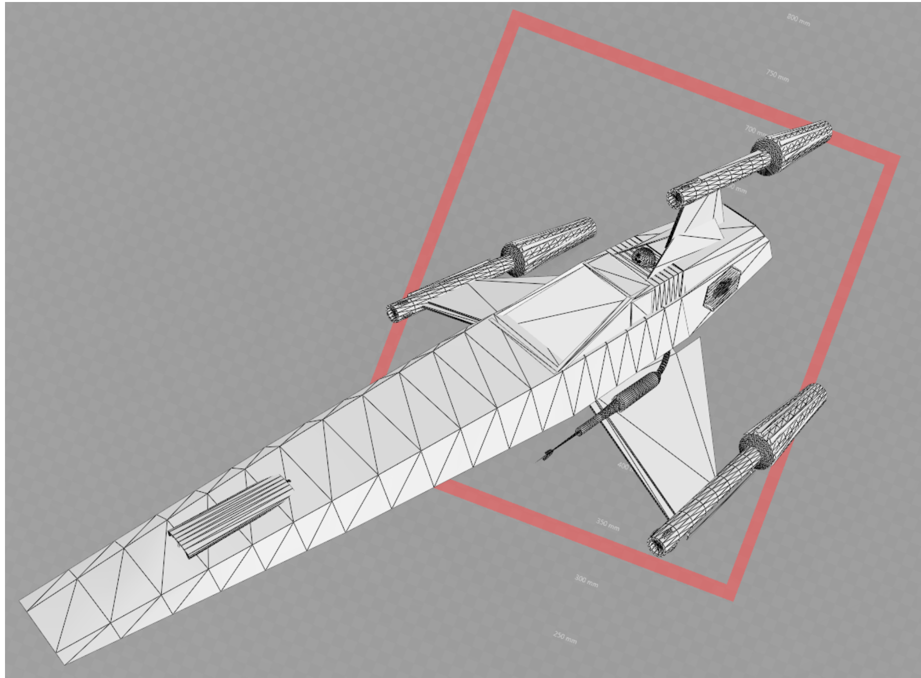


Figura 5.3: Exemplu de obiect 3D reprezentat printr-un fișier .obj

Specificația de OpenGL ES 1.1 prevede ca date de intrare acceptate șiruri de numere care descriu puncte, linii sau vârfurile unor triunghiuri (vezi Figura 3.4). Această versiune de specificație suportă doar triunghiuri ca și poligoane de intrare.

Unul dintre formatele de fișiere care conține astfel de date este formatul obj. Acest format de fișier conține pe fiecare linie coordonatele punctelor unui poligon din mulțimea poligoanelor care alcătuiesc suprafața dorită. Folosind acest format de fișier, am putut implementa într-o manieră directă transformarea datelor din acest format în formatul acceptat de specificația OpenGL ES 1.1. Figura 5.3 prezintă un astfel de obiect tridimensional specificat printr-un fișier obj.

De asemenea, un astfel de fișier obj poate să conțină mai multe astfel de suprafețe care să alcătuiască obiectul final. De exemplu, pentru reprezentarea tridimensională a unei mașini, carcasa, fiecare roată, motorul volanul sau scaunele pot fi reprezentate ca suprafețe individuale și distincte. Acest lucru permite specificare, pentru fiecare în parte, culori sau texturi diferite.

BUPT

O sursă bună de astfel de reprezentări o constituie creațiile artiștilor grafici care folosesc programe specializate pentru crearea unor astfel de obiecte tridimensionale. Rezultatul muncii acestora poate fi exportat într-un astfel de fișier cu format obiectual: fișierul conține lista de vârfuri menționată, precum și alte informații auxiliare, cum ar fi vectorii normali în fiecare punct, coordonate de textură, precum și numele unor fișiere de textură. În continuare, pentru a putea furniza aceste date de intrare, am căutat pe internet site-uri care să ofere astfel de fișiere obj cu licență open source. Ne-am oprit asupra site-ului *http://tf3dm.com*. Principalul motiv a fost numărul mare de fișiere disponibile și structura organizată a paginilor.

Astfel, am creat un modul software care accesa paginile care prezentau listele obiectelor și de aici extrăgea link-ul către fișierul obj. Apoi, în mod paralel am descărcat aceste fișiere. După această fază am strâns un număr de aproximativ 3000 de astfel de obiecte tridimensionale.

Nu toate aceste fișiere au putut fi folosite, în principal deoarece nu foloseau triunghiuri ca și poligoane pentru descrierea suprafețelor tridimensionale. Totuși, pentru a nu elimina un număr foarte mare de obiecte, am adaptat algoritmul de transformare a datelor astfel încât să accepte și suprafețe descrise prin patrulatere, pe care să le interpreteze ca două triunghiuri.

În acest fel, în final am rămas cu un număr de aproximativ 2400 de fișiere obj care să descrie suprafețe tridimensionale considerate valide de implementarea noastră și care să poată fi folosite ca date de intrare pentru implementarea pipeline-ului grafic. De asemenea, aceste fișiere oferă o acoperire foarte largă a dimensiunii obiectelor tridimensionale folosite, având reprezentări care folosesc de la 2 triunghiuri până la aproximativ 450000.

5.2.3 Transformarea datelor de intrare

Principala cerință din spatele acestui modul este nevoia de a furniza datele de intrare într-un anumit format specificat de OpenGL ES 1.1. Astfel, pentru a extrage lista de puncte și a o furniza pipeline-ului grafic prin interfața publică specificată, am construit un modul software care citește punctele dintr-un fișier obj cu o astfel de structură și o transformă într-o înșiruire de numere cu o semantică definită de specificație.

Specificația OpenGL ES 1.1 definește ca singurele poligoane acceptate triunghiurile. Acestea pot fi furnizate conform Figurii 3.4 ca și fâșie de triunghiuri, evantai de triunghiuri și, respectiv, ca și triunghiuri independente. Din cauză că fâșia de triunghiuri reprezintă, dintre cele trei posibilități, modalitatea cea mai eficientă din punct de vedere al memoriei ocupate pentru descrierea aceluiași număr de triunghiuri. Singura constrângere e ca triunghiurile să fie adiacente, ceea ce este adevărat pentru o suprafață tridimensională continuă. Tot din con-

BUPT

siderente de optimizare a consumului de memorie, OpenGL ES 1.1 specifică o modalitate indexată de furnizare a acestor vârfuri de triunghiuri. Astfel, abstractizând, datele de intrare pentru pipeline-ul grafic pot fi reprezentate de două șiruri:

- Mulțimea punctelor care definesc triunghiurile. Această mulțime conține elemente unice și fiecare intrare reprezintă coordonate într-un spațiu tridimensional. Toate aceste puncte se află pe suprafața care urmează să fie descrisă.

- Șirul indicilor. Acest șir conține indici în prima mulțime și ei sunt înșiruiți astfel încât, semantic să definească o înșiruire de triunghiuri conform modului de intrare specificat. Acest șir conține, de obicei, foarte multe intrări duplicate; dacă nu sunt intrări duplicate, acest mod indexat de furnizare a datelor de intrare nu este recomandat.

Modulul astfel conceput transformă datele din fișierele obj în cele două șiruri care sunt apoi furnizate, de către aplicație, pipeline-ului grafic.

De asemenea, unele fișiere obj folosesc patrulatere ca și poligoane pentru descrierea suprafeței tridimensionale. A fost astfel nevoie de conversia fiecărui patrulater în două triunghiuri, conform Figurii 5.4.

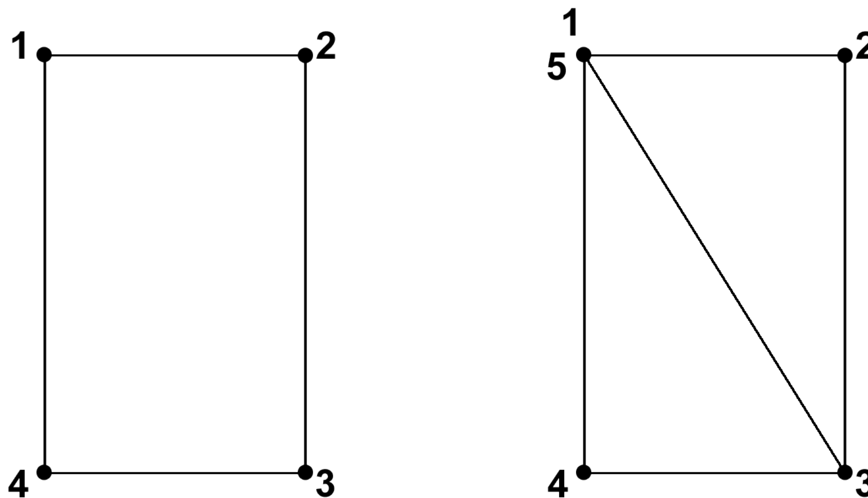


Figura 5.4: Conversie patrulater-triunghiuri

Astfel, din patrulaterul 1-2-3-4 , am obținut două triunghiuri: 1-2-3 și 3-4-5(3-4-1). În acest mod, am păstrat sensul de parcurgere a vârfurilor în ambele cazuri. Din această cauză, nu se poate folosi modul fâșie de triunghiuri si am

BUPT

fost nevoiți să folosim modul triunghiuri independente pentru semantica datelor de intrare.

Având cele descrise anterior, acest modul poate să citească fișiere obj care utilizează triunghiuri sau patrulatere pentru descriere suprafețelor și să furnizeze date într-un format descris de specificația OpenGL ES 1.1.

Suplimentar, acest modul are implementat un algoritm simplu pentru reținerea punctelor extreme ale obiectului tridimensional reprezentat. Aceste puncte extreme sunt folosite apoi de aplicația principală pentru a configura corect pipeline-ul grafic astfel încât proiecția obiectului tridimensional să încapă complet în fereastra de vizualizare.

5.2.4 Aplicația propriu zisă

După cum reiese și din Figura 5.2, am creat aplicația principală astfel încât să încorporeze, pe lângă implementarea specificației OpenGL ES 1.1 și modulul software de citire și transformare a datelor de intrare. Astfel, ea utilizează modulul de citire pentru a colecta datele care descriu suprafața obiectului tridimensional, pe care apoi le va transmite pipeline-ului grafic. În plus, modulul de citire furnizează și coordonatele extreme ale obiectului ce urmează să fie desenat, iar aplicația folosește această informație suplimentară pentru a încadra obiectul tridimensional în interiorul volumului de vedere al camerei și, în final, în interiorul ferestrei de vizualizare.

După citirea datelor, aplicația inițializează pipeline-ul grafic pentru a crea contextul corect în care se va desena ulterior obiectul. Apoi, în fiecare cadru, aplicația comandă pipeline-ului grafic să rotească cu două grade în plus față de cadrul anterior obiectele desenate, după care ii furnizează acestuia obiectul prin intermediul datelor citite inițial. Astfel, pipeline-ul acumulează unghiul de rotație al obiectului. În plus, aplicația furnizează pipeline-ului și vectorul $-1, -1, -1$ în jurul căruia se efectuează rotația. În total, se desenează 180 de cadre, rezultând o rotație completă a obiectului în jurul vectorului dat.

Figura 5.5 prezintă sintetizat, sub formă de schemă logică fluxul principal al aplicației. Astfel, schimbând fișierele de intrare citite, se pot desena toate obiectele dorite folosind aceeași aplicație. Formatul ales pentru fișierele de ieșire este PNG, cu patru canale per pixel: roșu verde, albastru și transparență.

De asemenea, întru-un fișier obj putând fi descrise mai multe suprafețe care să compună un obiect, am ales să desenăm fiecare astfel de suprafață cu o culoare individuală, astfel încât la compararea diferențelor să se facă distincția între fiecare suprafață. Figura 5.6 prezintă câtva astfel de exemple.

BUPT

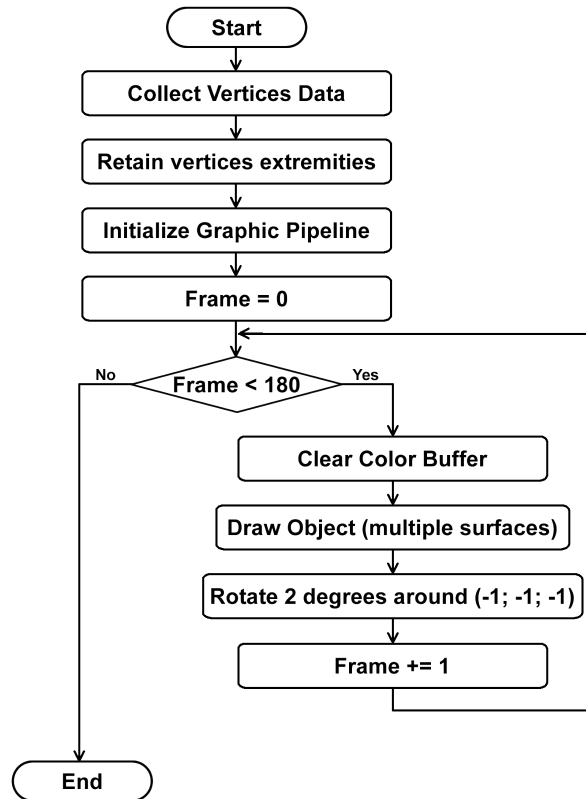


Figura 5.5: Schema logică a aplicației principale

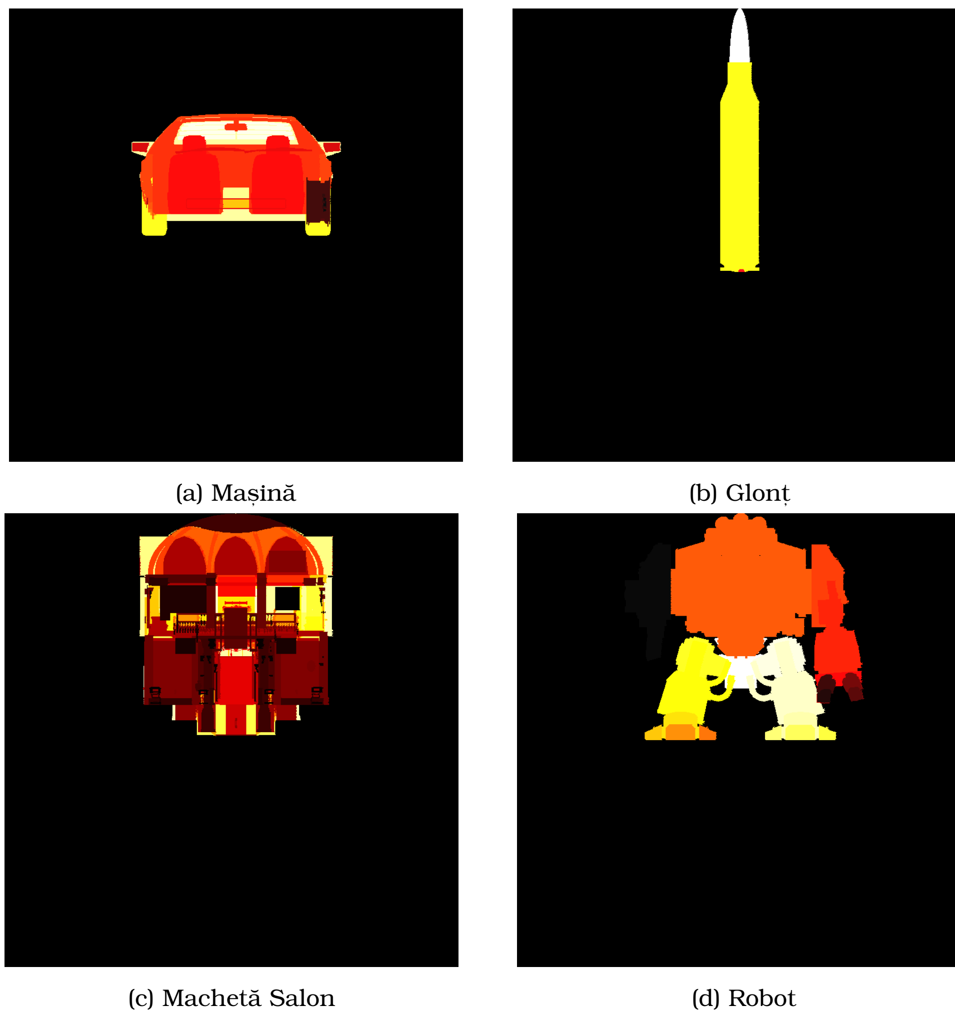
5.2.5 Compararea imaginilor obținute

Pentru fiecare combinație de obiect și format, aplicația principală produce câte 180 de imagini. Pentru a putea analiza diferențele imaginilor produse folosind formatele mai puțin precise, relativ la formatul exact, am creat un modul software care să poată compara, pixel cu pixel două imagini. Acesta primește ca intrare cele două imagini și produce o a treia imagine care reprezintă diferența pe canale de culoare la nivel de fiecare pixel. În plus, mai produce un al doilea fișier care conține valorile diferitelor metrici luate în considerare.

Pentru a putea compara pixel cu pixel cele două imagini, am folosit biblioteca libpng [78]. Având în vedere că formatul imaginilor este PNG, care este un format comprimat, această bibliotecă facilitează accesul direct la fiecare pixel al imaginii. De asemenea, în timpul comparării se calculează și valorile pentru metricile considerate.

Figura 5.7 prezintă un triplet de astfel de imagini în care prima este cea de

BUPT



(a) Mașină (b) Glonț

(c) Machetă Salon (d) Robot

Figura 5.6: Suprafețe multiple

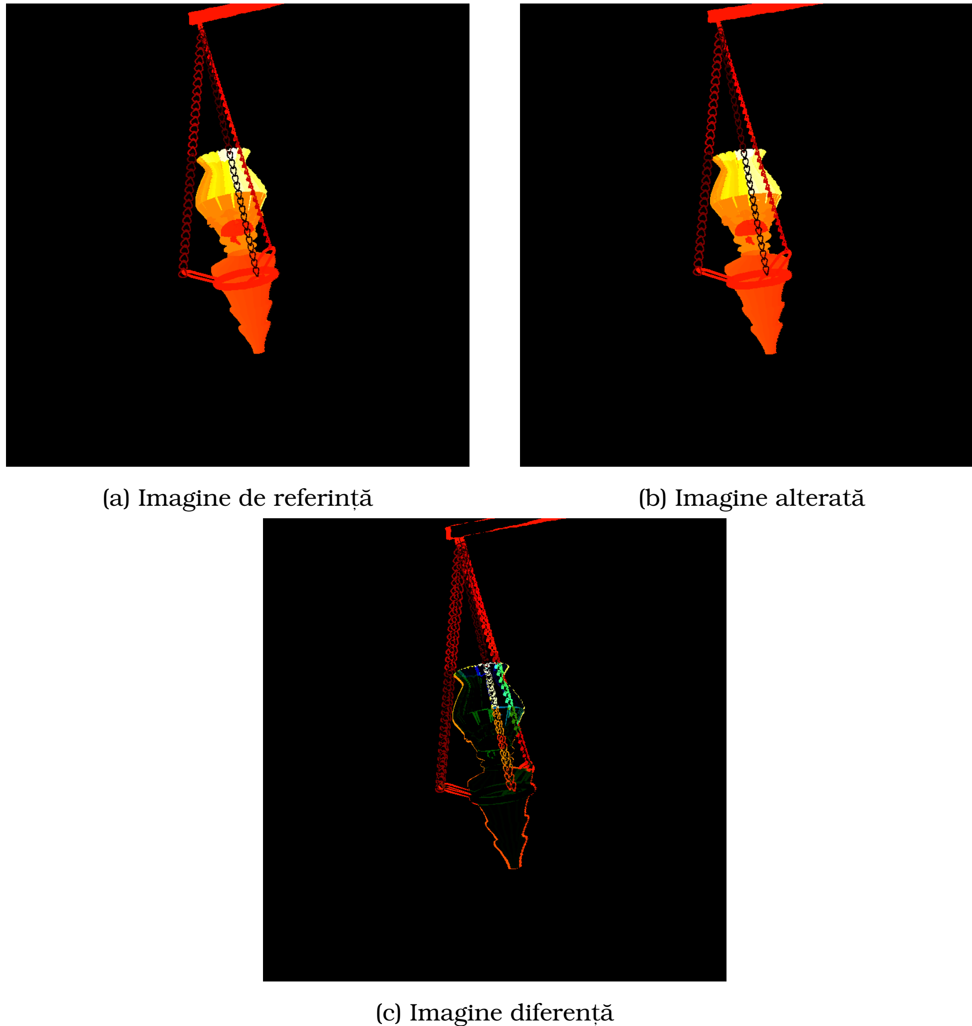
referință, a doua este cea comparată, iar a treia este imaginea diferență între cele două.

În continuare, prezentăm și un exemplu de conținut al unui fișier diferență pentru două imagini aleatoare:

```
diff = 798
 mse = 27.025736
psnr = 33.813026
```

Astfel, $diff$ reprezintă numărul de canale de culoare diferite pentru cele două imagini, mse reprezintă valoarea pentru metrica de medie a erorilor pătratice, iar $psnr$ reprezintă valoarea pentru metrica Peak Signal-to-Noise Ratio.

BUPT



(a) Imagine de referință (b) Imagine alterată

(c) Imagine diferență

Figura 5.7: Set de imagini comparate

În plus, am folosit acest program de comparare pentru a compara imaginile de referință cu o imagine complet neagră(roșu = 0, verde = 0, albastru = 0, transparență = 0). În urma acestei comparații, am obținut la câmpul $diff$ o valoare care ne va ajuta să calculăm numărul de pixeli utili din imaginea de referință. Această număr ne va fi util pentru a calcula valoarea noii metrici propuse.

BUPT

5.2.6 Centralizarea, filtrarea și interpretarea datelor

Contorizând numărul de valori create de modulul de comparare, rezultă un număr de ordinul milioanelor de valori care trebuie interpretate pentru a putea trage o concluzie asupra experimentelor făcute. Având în vedere numărul mare de numere obținute ca valori ale metricilor calculate și ca indicatori secundari, am fost nevoiți să creăm module software care să grupeze, să centralizeze și să dea o semnificație acestor valori numerice.

Din cauza multitudinii de fișiere create de modulul de comparare și a volumului mare de date conținute, citirea tuturor datelor dura aproximativ 30 de minute. Din această cauză, modificări în algoritmul de procesare a datelor necesitau timpi mari de așteptare pentru a putea fi validate.

Am încercat astfel să găsim o modalitate prin care citirea datelor să nu se facă de fiecare dată când se dorea modificarea algoritmului de analiză a acestora. Am găsit astfel o soluție în care programul principal are două părți:

- partea de citire a datelor care se execută o singură dată, iar datele sunt apoi încărcate complet în memoria RAM și vor fi accesate doar de acolo.

- de cealaltă parte, modulul care conține algoritmul ce procesează datele a fost creat ca un binar executabil separat care este încărcat de fiecare dată când se cere procesarea datelor. El poate fi astfel adaptat și modificat între două analize succesive.

Ambele părți sunt executate în cadrul aceluiași proces pentru a putea accesa memoria RAM în care primul modul încarcă datele și din care cel de-al doilea le accesează mult mai rapid decât de pe disc.

Figura 5.8 prezintă arhitectura generală a modulului de analiză a datelor, cu cele două părți descrise anterior.

Am putut astfel dezvolta și mai ales valida ușor algoritmii care filtrează datele și calculează valori agregate pe mai multe seturi de imagini ale metricilor luate în considerare. Detaliile de calcul ale acestor metrici și ale agregatelor lor vor fi discutate într-o secțiune ulterioară, după introducerea criteriilor de grupare ale imaginilor, în funcție de formatele de numere reale folosite.

După obținerea valorilor agregate ale metricilor, am creat fișiere formatate astfel încât să poată fi încărcate de aplicații terțe, aplicații cu ajutorul cărora să putem crea grafice prin care să putem interpreta mai ușor rezultatele obținute.

BUPT

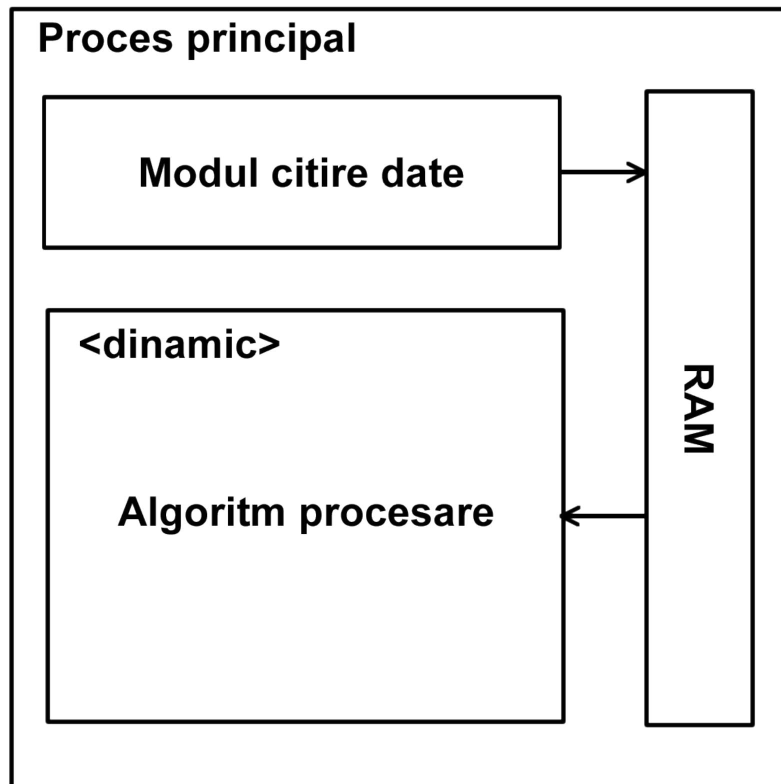


Figura 5.8: Arhitectura modulului de analiză a datelor

5.3 Schimbarea formatului folosit la nivel de stagiu al pipeline-ului grafic

Folosind implementarea proprie inițială a specificației de OpenGL ES 1.1 [77] am putut contoriza numărul de operații cu matrici sau matrice-vector care se efectuează într-un astfel de pipeline grafic. Principala concluzie desprinsă a fost că majoritatea covârșitoare a operațiilor matriciale se efectuează în primul stagiu al pipeline-ului, în stagiul de transformare a coordonatelor în diferitele spații intermediare necesare proiecției bidimensionale.

Astfel, pentru a mări viteza de procesare ar fi util să se folosească reprezentări ale numerelor fracționare care au o viteză sporită la calculul diferitelor operații aritmetice. Un astfel de format este reprezentarea în virgulă fixă. Principalul dezavantaj al unei astfel de reprezentări este precizia limitată pe care o oferă precum și plaja restrânsă de valori(vezi secțiunea 3.5).

BUPT

Dacă s-ar folosi în totalitate această reprezentare, pe un număr rezonabil de biți, precizia rezultatelor obținute ar fi destul de precară. Astfel, am încercat să creăm un pipeline grafic hibrid, care să folosească parțial numere fracționare reprezentate în virgulă fixă și parțial numere fracționare reprezentate în virgulă mobilă.

Folosind implementarea software proprie a specificației OpenGL ES 1.1, am ales câteva puncte unde să schimbăm, la execuție, reprezentarea numerelor reale. Pentru acest lucru am utilizat metode care se vor chema la începutul și la sfârșitul unor stagii. Astfel, după cum sunt reprezentate stagiile OpenGL ES 1.1 în figura 5.1, am ales următoarele două puncte de schimbare a formatului:

- la începutul pipeline-ului, înainte de primul stagiu (Per-Vertex Operations)
- înainte de al doilea stagiu(Rasterization)

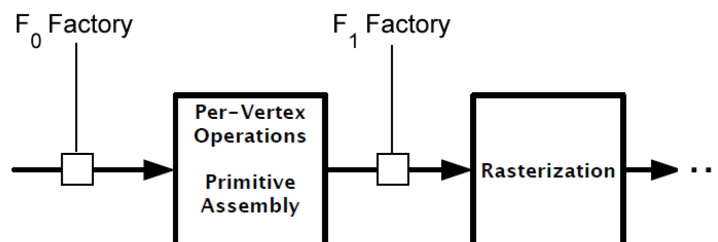


Figura 5.9: Puncte de schimbare a formatului în pipeline-ul grafic

Având stabilite punctele de schimbare a formatului, am avut nevoie de o abstractizare a numerelor fracționare care să poată fi folosită în toată implementarea pipeline-ului grafic. De asemenea era necesar ca formatul de reprezentare a numerelor fracționare să poată fi ușor schimbat și să se poată efectua operații între oricare reprezentare folosită.

Implementarea proprie a specificației de OpenGL ES 1.1 este scrisă în C++ într-o manieră orientată pe obiect, tocmai pentru a putea fi ușor extinsă cu noi funcționalități. Profitând de avantajele programării pe obiect, de facilitățile de acces direct la memorie oferite de limbajul C++ și încercând să implementăm aceste deziderate, am creat o clasă pentru numere fracționare, care oferă o interfață comună pentru operații aritmetice. Având această interfață, am implementat-o în subclase, astfel încât să ținem detaliile legate de precizie în fiecare clasă în parte.

În același timp, pentru a putea injecta cu ușurință, în punctele de modificare, implementările concrete ale reprezentărilor de numerele reale, am folosit o

BUPT

ierarhie de clase paralelă pentru crearea acestor obiecte. Având aceste clase fabrici [79], a fost nevoie doar de schimbarea obiectului care creează numere reale în punctele de schimbare ale formatului. Astfel, începând de la primul punct de instrumentare se folosea aceeași fabrică de formate de numere fracționare până când aceasta era schimbată la următorul punct. Ne-am asigurat astfel că pentru toate numerele fracționare se folosește aceeași reprezentare între punctele în care schimbăm formatul de reprezentare.

Pentru a separa detaliile legate de schimbarea formatului pentru reprezentarea numerelor reale de implementarea propriu-zisă a pipeline-ului grafic, am adăugat puncte de notificare în pipeline. Astfel, pipeline-ul generează diferite evenimente; pentru a putea schimba ușor formatul de reprezentare, am creat evenimente care se generează înainte de începerea stagiului "Per-Vertex Operations"(Evenimentul 1) și înainte de "Rasterization"(Evenimentul 2). Pentru a completa modificarea, am creat o clasă care știe să asculte după astfel de evenimente și să le trateze. Practic, pe metodele care tratează Evenimentul 1 și, respectiv, Evenimentul 2 schimbăm fabrica de obiecte pentru reprezentarea numerelor fracționare în funcție de cazul de test curent.

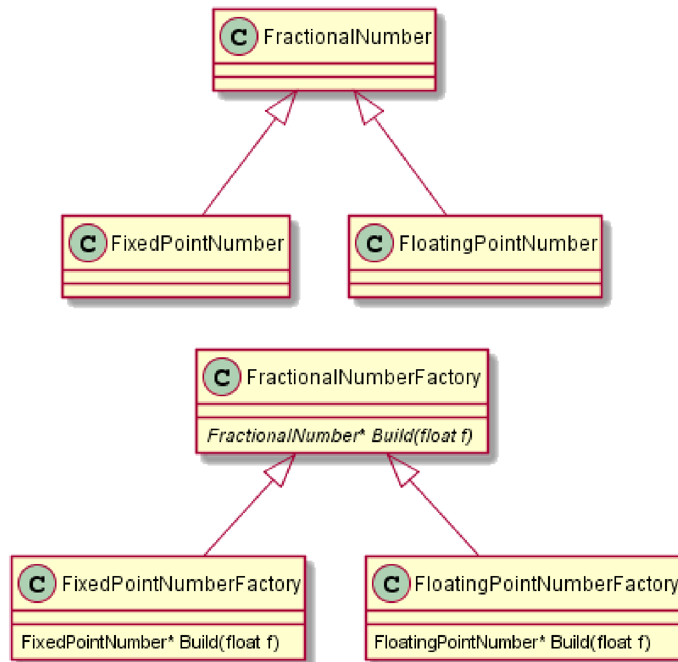


Figura 5.10: Diagramă de clase numere reale

BUPT

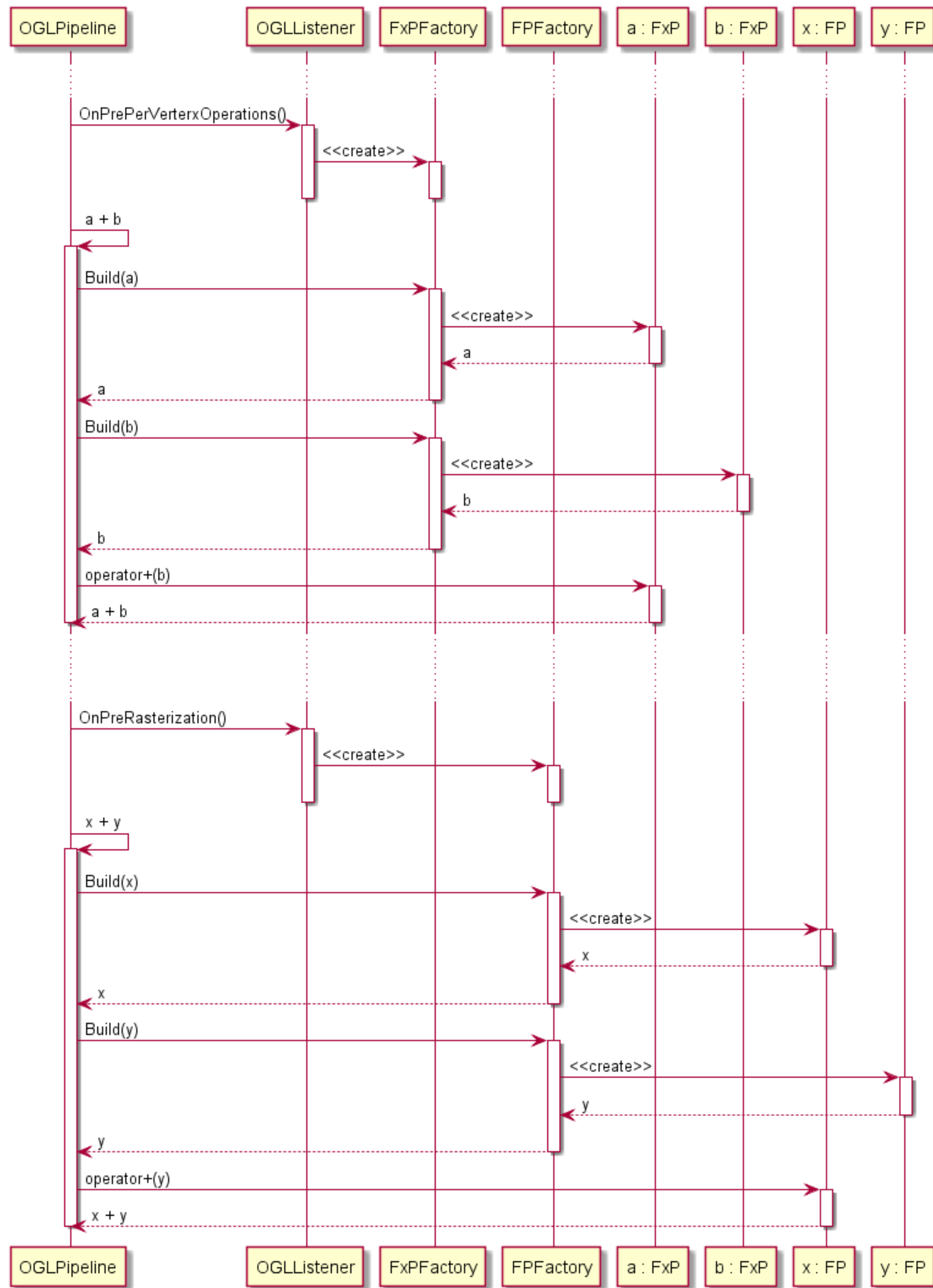


Figura 5.11: Diagramă de secvență a celor două puncte de schimbare a formatului

BUPT

	F_0	F_1
1	FP[1]	FP
2	16:16[2]	16:16
3	16:16	FP
4	FP	16:16
5	32:32	32:32
6	32:32	FP
7	FP	32:32

Tabela 5.1: Formate folosite

În tabelul 5.1 sunt prezentate combinațiile de precizie folosite, rezultând astfel un număr de șapte cazuri de test.

De cealaltă parte, interfața comună asigurată de clasa de bază a făcut posibilă efectuarea operațiilor aritmetice între oricare două numere reale, indiferent de implementarea concretă a reprezentării. De asemenea, crearea obiectelor de numere fracționare a fost astfel separată de partea care implementează operațiile aritmetice.

Figura 5.10 prezintă cele două ierarhii de clase, iar figura 5.11 prezintă o posibilă diagramă de secvență pentru apelurile de după cele două puncte de instrumentare.

5.4 Procesarea datelor și obținerea rezultatelor

Pentru a obține un set de imagini și date pentru o anumită combinație de formate de precizie(vezi Tabelul 5.1), am efectuat următorii pași:

1. Descărcarea fișierelor obj care conțin descrierea obiectelor tridimensionale ce urmează să fie proiectate.

2. Selectarea fișierelor care conțin descrieri prin triunghiuri sau patrulatere.

3. Rularea aplicației principale cu precizia setată pe virgulă flotantă în ambele puncte de schimbare a formatului (prima linie din tabelul 5.1) pentru fiecare fișier obj rămas după pasul anterior. Am obținut astfel o parte din setul de date de referință pentru viitoarele comparații: imaginile de referință.

[1] Floating Point - virgulă flotantă]
[2] format de virgulă fixă specificat ca dimensiune parte întreagă : dimensiune parte fracționară

BUPT

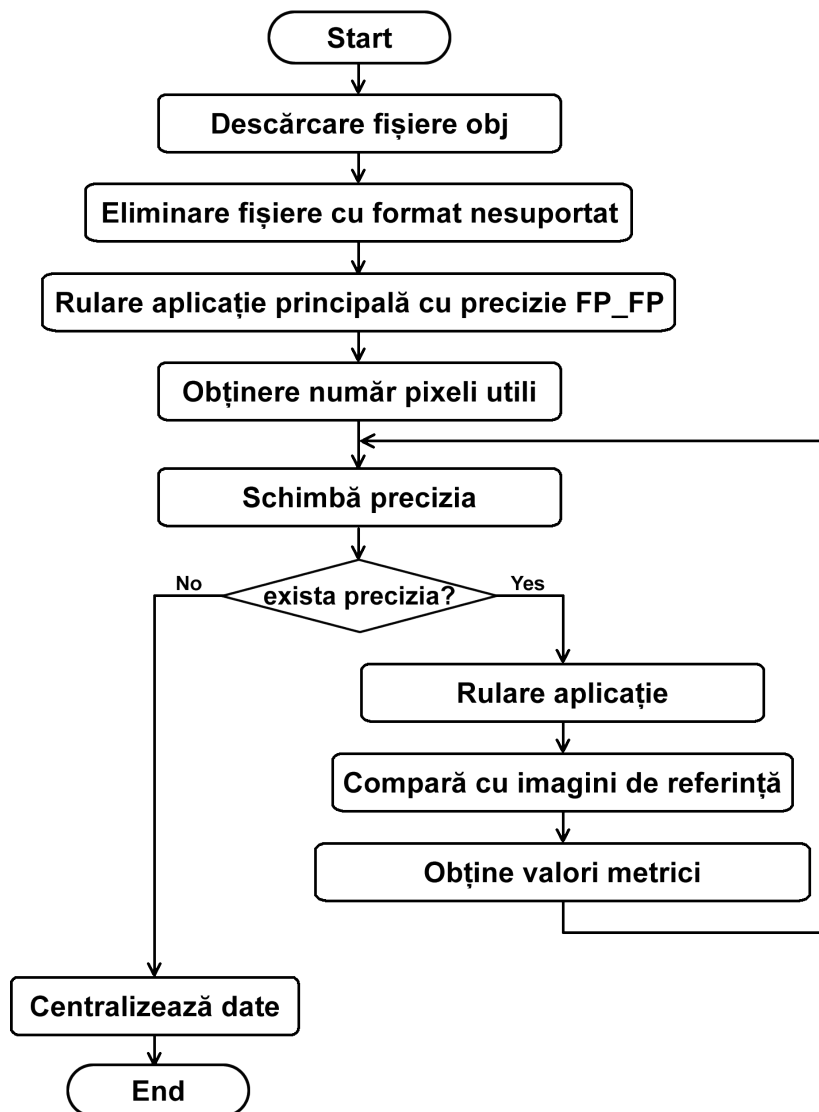


Figura 5.12: Schema logică de rulare a modulelor software create

4. Rularea modulului de comparație pentru fiecare imagine de referință și o imagine complet neagră. În această etapă am obținut numărul de pixeli utili ai fiecărei imagini de referință.

5. Rularea aplicației principale cu precizia setată pe valorile corespunzătoare uneia din liniile următoare din tabelul 5.1 pentru fiecare fișier obj rămas după pasul 2. Am obținut astfel imagini alterate

BUPT

6. Rularea modulului de comparație între fiecare imagine de referință și echivalentul ei alterat. Am obținut astfel datele corespunzătoare unui set de imagini alterate produse de pipeline-ul grafic cu precizie mixtă, conform liniei utilizate din tabelul 5.1.

7. Efectuarea pașilor 5 și 6 până lav epuizarea tuturor liniilor din tabelul 5.1

8. Rularea modulului de centralizare filtrare și interpretare a datelor.

Figura 5.12 prezintă schema logică a rulării modulelor software prezentate anterior conform pașilor descriși mai sus.

Prin modulele software prezentate anterior, am pus bazele unei infrastructuri de lucru și a unui proces care să ne ajute la generarea, colectarea și interpretarea unor valori ale unor metrici care să ne ajute la studierea și înțelegerea influenței combinațiilor de precizie asupra imaginilor produse de un pipeline grafic hibrid. Astfel, vom prezenta în continuare rezultatele obținute prin utilizarea acestei infrastructuri.

BUPT

Capitolul 6

Rezultate calcule aproximative

Pentru a putea analiza în mod obiectiv diferențele dintre imaginile imprecise și cele precise am avut nevoie de o măsură a acestor diferențe. Acest lucru este posibil prin calcularea valorilor unor metrici pentru setul de imagini care se dorește analizat. Am folosit astfel o serie de metrici deja existente, folosite cu succes în analiza imaginilor transmise la distanță prin diferite canale de comunicare imprecise sau a imaginilor compresate cu pierdere de precizie. De asemenea, am propus o nouă metrică care să fie ușor de calculat și care, prin comparație cu metricile deja existente să ofere o valoare cu o semantică ușor de înțeles.

Pentru început, vom furniza câteva date statistice despre setul de obiecte pe care le-am folosit ca date de intrare pentru pipeline-ul grafic modificat.

De asemenea, vom prezenta în continuare o serie de imagini produse de pipeline-ul grafic modificat, folosind diferite combinații de precizie, precum și rezultatele comparațiilor efectuate asupra întregului set de imagini, obținute în urma utilizării diferitelor combinații de precizie. Pentru aceasta, am folosit metricile MSE(eng. “Mean Square Error”) și PSNR(“Peak Signal-to-Noise Ratio”).

6.1 Informații generale despre setul de obiecte tridimensionale folosit

Pentru a realiza studiul propus, am folosit un total de 2400 de fișiere obj, care conțin obiecte tridimensionale cu caracteristicile prezentate în tabelul 6.1. De asemenea, pentru a completa datele prezentate în tabelul 6.1, figurile 6.1 și cele din Anexa A prezintă distribuțiile numărului de obiecte pentru fiecare

BUPT

caracteristică în parte.

	minim	maxim	mediu
vârfuri	4	257085	12862
triunghiuri	2	449472	20921
dimensiune volum	(1, 1, 1)	(4843610, 4825247, 4843610)	(4766, 4756,4765)
dim(x)/dim(y)	0.384	1.998	1.002
dim(y)/dim(z)	0.500	1.581	0.999
dim(z)/dim(x)	0.529	1.945	1.0005

Tabela 6.1: Caracteristici obiecte tridimensionale

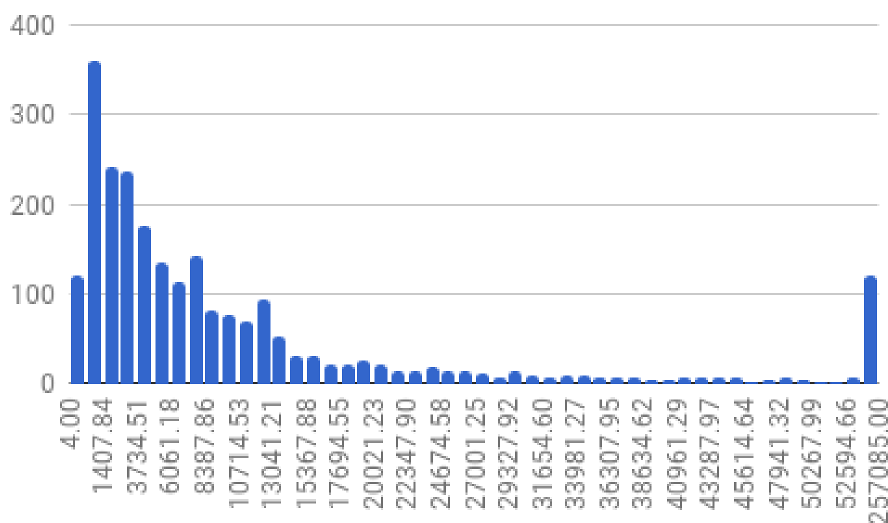


Figura 6.1: Histogramă distribuție vârfuri

Astfel, din figura 6.1 se poate observa că majoritatea obiectelor au până în 15000 de vârfuri, cu o distribuție relativ uniformă pe restul intervalului până la 250000 de vârfuri. De asemenea, se observă că există peste 100 de obiecte descrise prin mai mult de 250000 de vârfuri.

Asemănătoare cu distribuția numărului de vârfuri este și distribuția numărului de triunghiuri(figura A.1). Acest lucru era de așteptat, având în vedere faptul că triunghiurile sunt o grupare de vârfuri.

De asemenea, figura A.2 evidențiază volumul obiectelor tridimensionale din biblioteca construită. Astfel, se poate observa că majoritatea obiectelor au un volum cuprins în intervalul 1 - 39 unități de volum. Cu toate acestea, există

BUPT

un număr mai mic de obiecte cu un volum care atinge aproximativ 5000000 unități de volum. Acest lucru evidențiază dimensiunile variate ale obiectelor folosite pentru proiecție.

De asemenea, figurile A.3, A.4 și A.5 prezintă rapoartele dimensiunilor maxime ale obiectelor pe câte două axe: $max(x)/max(y)$, $max(y)/max(z)$ și respectiv $max(z)/max(x)$. Având în vedere că valorile rapoartelor prezentate în aceste histograme variază aproximativ în intervalul 0.4 - 2.0, cu majoritatea valorilor cetrate în jurul lui 1.0, am tras concluzia că majoritatea obiectelor tridimensionale se încadrează în volume cubice, de dimensiuni variate. Cu toate acestea, există obiecte tridimensionale care se încadrează în volume paralelipipedice aliniate cu axele, în care una dintre dimensiuni are valoarea mai mică decât dublul uneia din celelalte două dimensiuni.

Se poate astfel observa că obiectele acoperă o arie destul de mare de caracteristici, constituind o colecție care oferă un domeniu larg de valori de intrare pentru studiul propus.

6.2 Imagini alterate. Evaluare subiectivă

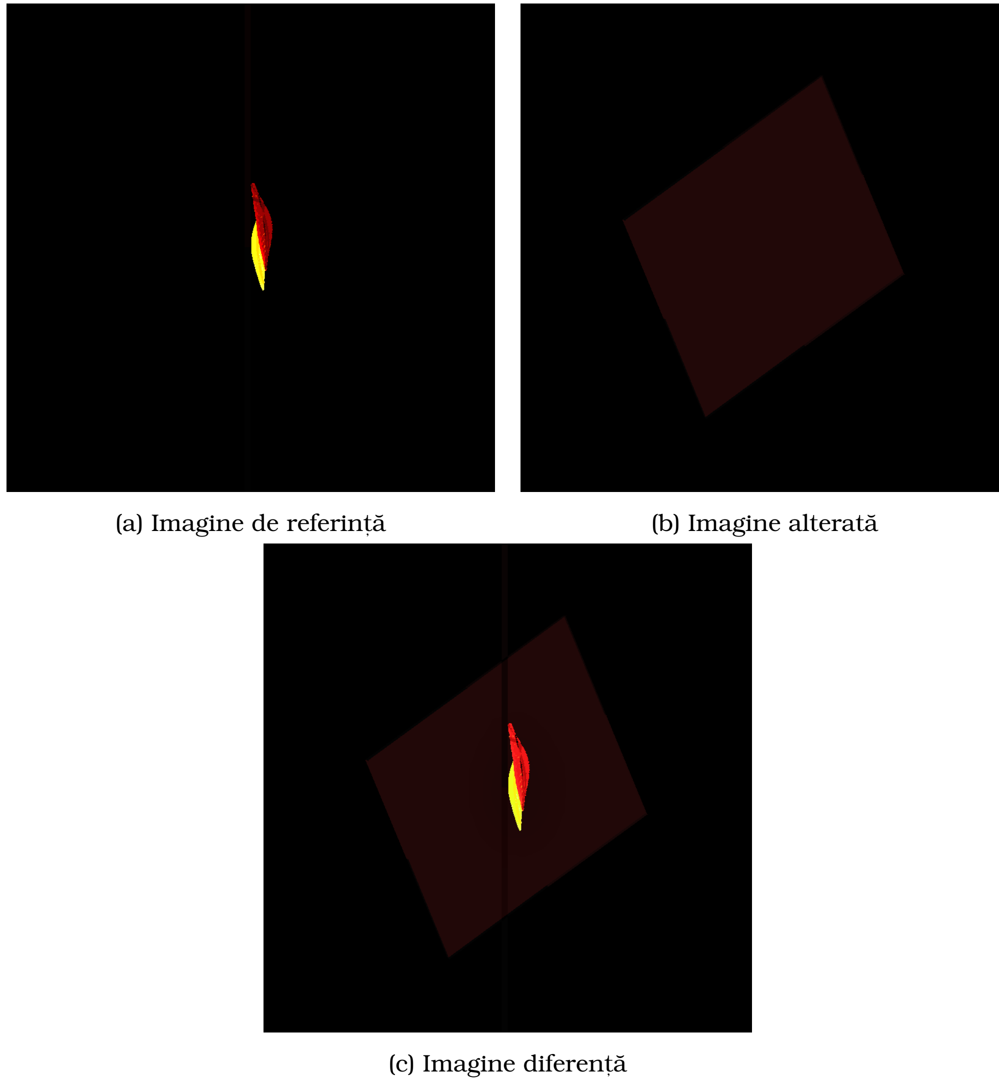
După cum am descris în capitolul anterior (Capitolul 5), la finalul procesului propus am obținut o serie de imagini de referință, obținute prin utilizarea formatului de virgulă flotantă pe tot parcursul pipeline-ului grafic, precum și o serie de imagini alterate prin utilizarea unor combinații de precizie(vezi tabelul 5.1); în același timp am obținut și o serie de valori numerice ale unor metrici, cu ajutorul cărora vom încerca să analizăm diferențele dintre imaginile astfel obținute.

Figura B.1 (Anexa B)prezintă o suită de cadre randate pentru un obiect rotit între 0 și 120 de grade în jurul vectorului $-1, -1, -1$, folosind formatul de virgulă flotantă pe tot parcursul pipeline-ului. Pe de altă parte, figura B.2 prezintă aceeași suită de cadre randate pentru același obiect rotit în jurul aceluiași vector, dar folosind un format alterat 16_16. Se poate observa cu ochiul liber că de la un anumit cadru încolo, diferențele sunt semnificative. Pentru a susține acest punct de vedere subiectiv, figura B.3 prezintă suita de diferențe între precedentele cadre. Se poate foarte ușor observa de aici că diferența crește odată cu rotația obiectului. Acest lucru poate fi cauzat de acumulare pierderilor de precizie în calcularea matricii de rotație.

De asemenea, Figurile C.1, C.2 și, respectiv C.3 (Anexa C) prezintă o aceeași secvență de cadre, folosind un format alterat 32_32. De data aceasta, diferențele între Figurile C.1 și C.2 nu sunt la fel de ușor de observat cu ochiul liber. Pentru a le putea aprecia corect, este nevoie de figura C.3, în care se prezintă diferențele dintre cele două seturi de imagini.

Prezentăm în continuare seturi de imagini selectate astfel încât să evidențiem

BUPT



(a) Imagine de referință

(b) Imagine alterată

(c) Imagine diferență

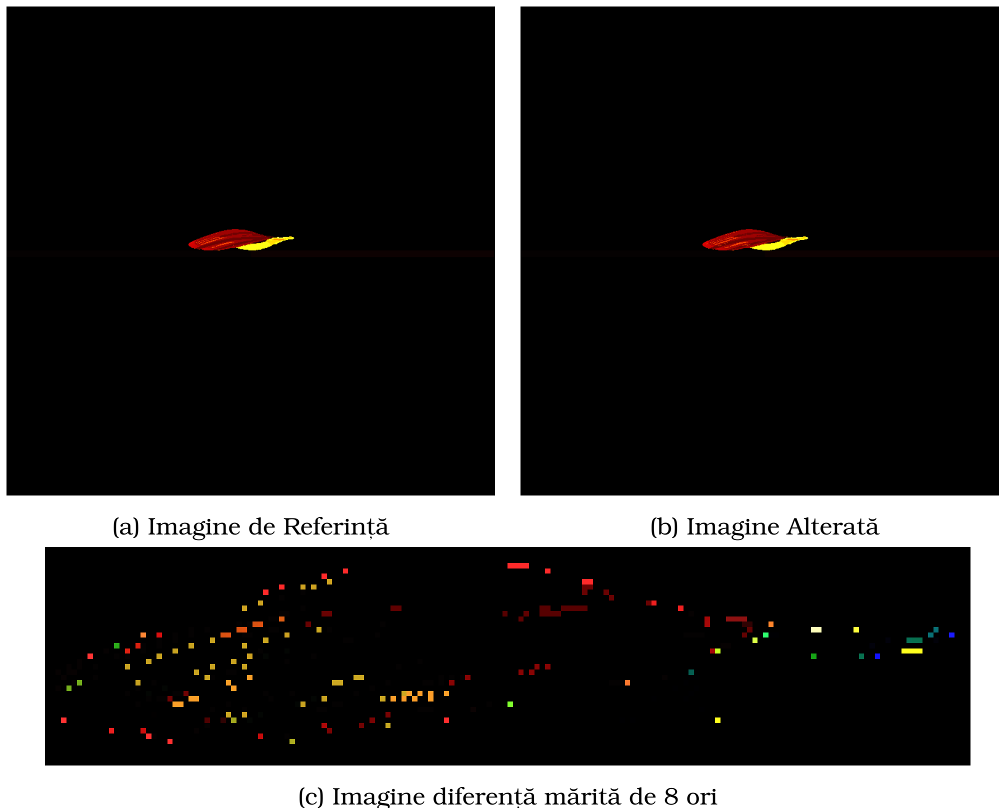
Figura 6.2: Set de imagini cu diferență mare

cele două extreme în care se pot afla două cadre. Astfel, figura 6.2 prezintă un set de trei imagini, din care prima este cea de referință, cea de-a doua este alterată, iar cea de-a treia prezintă diferența dintre cele două imagini. În acest set, diferența este semnificativă între imaginea alterată și cea de referință, după cum se poate observa și din imagini. Practic, imaginea alterată nu are nimic în comun cu cea de referință.

În continuare, figura 6.3 prezintă un alt trio de imagini ale aceluiași obiect, cu același format folosit pentru imaginea alterată, dar un alt cadru, în care

BUPT

diferența este aproape nesemnificativă. Pentru a putea sesiza diferențele, a fost nevoie să mărim imaginea diferență de opt ori.



(a) Imagine de Referință

(b) Imagine Alterată

(c) Imagine diferență mărită de 8 ori

Figura 6.3: Set de imagini cu diferență mică

De asemenea, figura D.1, figura D.2 și, respectiv, figura D.3 (Anexa D) precum și figura E.1, figura E.2 și, respectiv, figura E.3 (Anexa E) prezintă alte două seturi de imagini de referință, alterată și diferență la o dimensiune mai mare, din care se pot distinge mai multe detalii. Și din aceste seturi de imagini se poate vedea că diferențele dintre imagini nu se pot evalua ușor cu ochiul liber, fiind practic insesizabile. Pentru obținerea acestor seturi de imagini s-a folosit combinația de format 32_32.

6.3 Imagini alterate. Evaluare obiectivă: MSE și PSNR

După cum s-a putut vedea în secțiunea precedentă, există diferențe între imaginile alterate și cele de referință; de la diferențe subtile, de doar câțiva

BUPT

pixeli, până la diferențe importante, în care imaginile nu mai seamănă aproape de loc. Pentru a putea evalua uniform și obiectiv aceste diferențe, este nevoie de introducere unor metrici care să ofere o caracterizare unică a acestor diferențe. Am folosit astfel metricile deja existente "MSE", "PSNR" și am definit o nouă metrică "M" care să fie ușor de calculat și care să aibă o semantică ușor de înțeles.

Astfel, vom prezenta în continuare definirea formulelor de calcul ale metricilor MSE și PSNR, câteva valori numerice obținute pentru imaginile prezentate anterior, câteva grafice cu valorile obținute, după care vom introduce noua metrică definită. După aceea vom prezenta agregări prin medie ale valorilor metricilor, obținute pentru mai multe imagini.

6.3.1 Definire

Două dintre cele mai folosite metrici pentru evaluarea alterării unei imagini având o imagine de referință sunt MSE și PSNR. Ele se definesc pe baza diferențelor absolute dintre cele două imagini.

Astfel, în [68] cele două metrici sunt definite prin ecuațiile următoare:

$$MSE(f,g) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(f_{ij} - g_{ij})^2 \tag{6.1}$$

$$PSNR(f,g) = 10\log_{10}(\frac{255^2}{MSE(f,g)}) \tag{6.2}$$

După cum se poate observa, PSNR este o metrică definită pe o scară logaritmică. Astfel, creșterea datorată diferențelor între imagini este mai puțin abruptă decât în cazul MSE. De asemenea, se poate observa că pentru imagini identice, MSE are valoarea 0, în timp ce PSNR tinde la infinit. Altfel spus, două imagini sunt cu atât mai apropiate cu cât MSE are valoare mai mică și, respectiv, PSNR are valoare mai mare.

6.3.2 Valori obținute

În continuare, tabelul 6.2 prezintă valorile obținute pentru metricile MSE și PSNR pentru seturile de 180 de cadre de imagini referință, alterate și diferență din care am extras seturile B.1, B.2 și, respectiv, B.3. De asemenea, ultima coloană reprezintă numărul de canale de culoare diferite între imaginea de referință și cea alterată.

În mod asemănător, tabelul 6.3 prezintă valorile obținute pentru metricile MSE și PSNR pentru seturile de imagini referință, alterate și diferență din care am extras seturile C.1, C.2 și, respectiv, C.3.

BUPT

cadru	MSE	PSNR	diff
0	146.36	26.48	4519
8	941.72	18.39	29035
16	1642.33	15.98	52280
24	2179	14.75	72658
32	2717.66	13.79	91968
40	3179.78	13.11	107081
48	3193.78	13.09	108154
56	3429.89	12.78	115253
60	3590.47	12.58	121176

Tabela 6.2: Valori MSE și PSNR pentru B.3

cadru	MSE	PSNR	diff
0	57.67	30.52	2093
8	58.72	30.44	2153
16	66.11	29.93	2472
24	72.92	29.5	2701
32	71.26	29.6	2611
40	76.75	29.28	2731
48	63.54	30.1	2288
56	52.37	30.94	1848
60	50.45	31.1	1822

Tabela 6.3: Valori MSE și PSNR pentru C.3

După cum se poate remarca, valorile pentru MSE prezentate în tabelul 6.2 sunt în general mai mari decât cele corespunzătoare din tabelul 6.3. Analog, valorile pentru PSNR sunt în general mai mici. Acest lucru indică o degradare mai mare a imaginilor alterate prin utilizarea reprezentării 16_16 față de imaginile alterate prin utilizarea reprezentării 32_32. De asemenea, dacă în primul cadru valoarea metricii MSE este mai mică în tabelul 6.3, față de tabelul 6.2, respectiv este mai mare pentru PSNR, se poate observa că variația valorilor PSNR este mai mică decât variația valorilor MSE.

6.3.3 Grafice ale valorilor obținute

După cum se poate vedea și din tabelele 6.2 și 6.3, este destul de greu de făcut comparații între valorile acestor metrici. Trebuie de asemenea avut în considerare că tabelele respective prezintă doar un mic număr de valori ale metricilor. Din această cauză, o reprezentare vizuală a acestor valori ajută la interpretarea lor. În plus, simplifică compararea valorilor obținute pentru diferite seturi de imagini. Astfel, figurile din Anexa F prezintă grafice pentru toate

BUPT

cele 180 de cadre randate pentru un anumit obiect, folosind diferite combinații de formate pentru reprezentarea numerelor fracționare.

Astfel, Figurile F.1 și F.2 (Anexa F) prezintă graficele pentru valorile MSE și, respectiv, PSNR pentru toate cele 180 de cadre desenate folosind combinația de formate 16_16, ale obiectului cu suita prezentată în Figurile B.1, B.2 și B.3. Figurile F.3 și F.4 prezintă graficele pentru valorile MSE și, respectiv, PSNR pentru toate cele 180 de cadre desenate folosind combinația de formate 32_32, ale aceluiași obiect.

În continuare, Figurile F.5 și F.6 prezintă graficele pentru valorile MSE și, respectiv, PSNR pentru toate cele 180 de cadre desenate folosind combinația de formate 16_16, ale obiectului cu suita prezentată în Figurile C.1, C.2 și C.3. Figurile F.7 și F.8 prezintă graficele pentru valorile MSE și, respectiv, PSNR pentru toate cele 180 de cadre desenate folosind combinația de formate 32_32, ale aceluiași obiect.

Păstrând aceeași logică a graficelor, Anexa F prezintă o serie de alte grafice ale valorilor metricilor MSE și PSNR pentru seturi suplimentare de obiecte și combinații de formate folosite pentru reprezentarea numerelor reale.

Din toate cele patru seturi de grafice prezentate anterior, precum și din restul graficelor din Anexa F se poate ușor observa că valorile metricilor variază, pe lângă combinația de formate de precizie folosite, și în funcție de obiectul care este randat. Același lucru se poate desprinde și din Tabelele 6.2 și 6.3.

În plus, având în vedere ecuația 6.2, PSNR poate lua valori ce tind spre infinit atunci când valoare MSE tinde spre zero. Altfel spus, în cazul în care pozele sunt identice, valoarea PSNR tinde spre infinit.

6.4 Introducerea unei metrici noi

După cum s-a putut observa în secțiunea precedentă MSE are o plajă de valori foarte largă, fără a avea o semantică clară. Pentru a reduce plaja de valori, metrica PSNR a fost definită pe baza MSE, dar folosind o scară logaritmică. Chiar dacă plaja de valori a fost oarecum limitată, nici semantica valorilor metrici PSNR nu este clară.

6.4.1 Definire

Având în vedere aceste aspecte, am definit o metrică a cărei valori să aibă în primul rând un domeniu de valori limitat sau, cel puțin, să aibă o semantică ușor de înțeles pentru un domeniu de valori limitat.

Astfel, ecuația 6.3 definește formula de calcul a metricii propuse.

BUPT

$$M(f,g) = \frac{1}{P(f)} \sum_{i=1}^{M} \sum_{j=1}^{N} D(f_{ij}, g_{ij}) \tag{6.3}$$

unde

$$D(f_{ij}, g_{ij}) = \begin{cases} 1, f_{ij} \neq g_{ij} \\ 0, f_{ij} = g_{ij} \end{cases} \tag{6.4}$$

și

$$P(f) = \textit{numărul de pixeli utili ai matricii de pixeli } f \tag{6.5}$$

În ecuațiile precedente, f este matricea totală de pixeli ai imaginii de referință, iar g este matricea de pixeli ai imaginii alterate. De asemenea, M și N reprezintă dimensiunile matricilor de pixeli.

În plus, din ecuația 6.3 reiese faptul că imaginea de referință și cea alterată trebuie să aibă aceeași dimensiune.

Practic $\sum_{i=1}^{M} \sum_{j=1}^{N} D(f_{ij}, g_{ij})$ din ecuația 6.3 reprezintă diferența de pixeli dintre imaginea de referință și cea alterată. Astfel, semantica metricii propuse este: raportul dintre numărul de pixeli diferiți față de imaginea de referință și pixelii utili ai imaginii de referință. Astfel, dacă $M(f,g)$ are valori mai mari decât 1, diferența este mai mare decât imaginea referință. La polul opus, valoarea 0 a metricii semnalează faptul că cele două imagini sunt identice.

În continuare, putem introduce o interpretare procentuală a metricii M astfel încât valoare să semnifice procentul de pixeli diferiți din numărul de pixeli utili ai imaginii de referință. Astfel, ecuația 6.6 prezintă formula procentuală de calcul a metricii.

$$M\%(f,g) = M(f,g) * 100 \tag{6.6}$$

6.4.2 Valori obținute

cadru	M%
0	2.78368
8	16.68419
16	29.09633
24	40.45388
32	52.07052
40	62.34157
48	66.53747
56	73.10410
60	77.32746

Tabela 6.4: Valori M% pentru B.3

BUPT

Tabelul 6.4 prezintă valorile obținute ale metricii $M\%$, pentru suita de cadre prezentate în B.1, B.2 și B.3, cadre care folosesc combinația 16_16. Se poate astfel ușor observa că la cadrul 60 diferența este de 25% față de imaginea originală.

De asemenea, tabelul 6.5 prezintă valorile obținute ale metricii $M\%$, pentru suita de cadre prezentate în C.1, C.2 și C.3; pentru aceste cadre s-a folosit combinația 32_32. Din acest tabel se poate observa că valorile metricii sunt mai mici, ceea ce era de așteptat, având în vedere că se folosește o combinație de formate mai precise decât în cazul precedent. Această tendință se poate observa în valorile obținute pentru toate obiectele, între combinațiile de formate mai puțin precise și cele mai precise.

De asemenea, în unele cazuri se obțin valori mai mari de 100, ceea ce semnifică diferențe mai mari de pixeli decât numărul de pixeli existenți în imaginea de referință. În aceste cazuri, imaginea alterată este clar diferită față de cea originală. Încercând să limitam și mai mult valoarea metricii pentru care diferența între două imagini este clar vizibilă, putem afirma că și pentru valoare 20 diferențele sunt semnificative. Această valoare înseamnă că o cincime(20 %) din pixelii imaginii de referință sunt diferiți.

cadru	M%
0	0.34848
8	1.28212
16	2.63448
24	4.46838
32	7.02263
40	10.19940
48	15.04912
56	20.45513
60	23.54235

Tabela 6.5: Valori M% pentru C.3

6.4.3 Grafice ale valorilor obținute

Ca și în cazul metricilor MSE și PSNR, este greu de urmărit variația valorilor metricii. Astfel, Anexa G prezintă o serie de grafice care să exemplifice valorile luate de metrica M%. În plus, față de tabel, un grafic conține valori ale metricii pentru toate cele 180 de cadre randate cu un obiect tridimensional, folosind câte o combinație de precizie.

Astfel, figura G.1 prezintă valorile metricii M% pentru toate cadrele randate folosind combinația de formate de precizie 16_16, ale obiectului 0074, prezentat în figurile B.1, B.2 și B.3. În continuare, G.2 prezintă valorile metricii M%

BUPT

pentru același obiect randat utilizând combinația de formate de precizie 32_32. În continuare, celelalte imagini din Anexa G prezintă grafice ale valorilor metricii M% pentru diferite seturi de obiecte și combinații de precizie.

Astfel, principalele concluzii care se pot extrage și din graficele valorilor metricii M% este că acestea sunt în primul rând influențate de precizia combinațiilor de reprezentare folosite dar și de natura obiectului tridimensional desenat. Astfel, pentru combinațiile de reprezentări mai puțin precise valorile sunt în general mai mari decât pentru cele mai precise.

Pentru a susține aceste idei, am creat graficele I.1, I.2, I.3, I.4, I.5 și I.6 care prezintă valorile medii ale metricii M% pentru fiecare obiect în parte, în funcție de combinația de precizie folosită în pipeline-ul grafic. Din aceste grafice se poate ușor observa că se obțin rezultate mai apropiate de referință (valori mai mici ale metricii M%) pentru combinațiile de precizie care folosesc reprezentări complementare astfel încât o precizie mai mică în prima parte a pipeline-ului este compensată de o precizie ridicată în a doua partea decât în cazul invers, când se folosesc reprezentări precise în prima parte și mai puțin precise în cea de-a doua.

6.5 Analiză comparativă între MSE, PSNR și M%

După cum am putut vedea până acum, din punct de vedere al valorilor, fiecare din cele trei metrici au o plajă de valori variată. Tabelul 6.6 evidențiază tocmai acest lucru.

	min	max	medie
MSE	0	65498.723	556.002
PSNR	-0.0315	110.964	34.83
M%	0	373544.438	33.76

Tabela 6.6: Valori extreme pentru metrici

Astfel, am putut observa că pentru valori apropiate ale M%(figura G.2 și G.7), MSE are valori diferite(F.3 și F.13), de la un obiect la altul și de la o combinație de precizie la alta. Pe de altă parte, PSNR are valori aproximativ în același interval în ambele cazuri(F.4 și F.14).

De asemenea, pentru valori mai mari de ≈20 ale M%(G.1, G.3, G.5, G.9), MSE are valori începând de la ≈500, ≈90, ≈2000 și, respectiv ≈100, pentru aceleași combinații de precizie, dar obiecte diferite. Acest lucru întărește concluzia anterioară că valoarea MSE variază în funcție de obiect. De asemenea, pentru aceleași obiecte, cu aceleași combinații de precizie, PSNR are valori mai mici de ≈20, ≈30, ≈15 și, respectiv ≈30. Din aceste exemple reiese că nici PSNR nu are valori proporționale pentru aceleași valori ale M%.

BUPT

Precedentele exemple arată că valorile MSE și PSNR variază în funcție de natura obiectului proiectat și, implicit, de natura și dimensiunea utilă a proiecției. Astfel, cu cât proiecția unui obiect este mai mare, adică ocupă o suprafață mai mare de pixeli pe imaginea bidimensională rezultată, cu atât MSE și PSNR au valori care semnifică diferențe mai mari. Totuși, din imaginile prezentate în anexele B, C, D și E se poate observa că pixelii diferiți sunt dispuși spre periferia proiecțiilor.

Acest lucru confirmă faptul că o metrică relativă la dimensiunea utilă a proiecției, cum este M%, are rezultate superioare, din punct de vedere al aproximării percepției umane, unor metrici absolute, cum sunt MSE și PSNR. Acest lucru este în concordanță cu faptul că ochiul uman nu este foarte sensibil la diferențe dispersate.

6.6 Metrici agregate

Din graficele prezentate în secțiunile anterioare am putut urmări valorile metricilor pentru câteva obiecte și combinații de formate selectate. Din ce am putut observa, există o oarecare variație a valorilor metricilor și în funcție de obiectul tridimensional folosit la intrare, pentru care s-au calculat. Pentru a ne putea crea o imagine de ansamblu asupra distribuției valorilor metricilor pentru toate obiectele, am creat o serie de grafice care agregă valorile pentru toate obiectele, și le prezintă pentru toate cele 180 de cadre, pentru fiecare combinație de precizie în parte.

Se poate astfel observa din graficele agregate, prezente în anexa H că variațiile valorilor fiecărei metrici prezentate pentru fiecare obiect individual în anexele F și G se păstrează pentru fiecare metrică în parte. Astfel, se poate deduce, global, că rezultatul final dat de folosirea unui format imprecis în prima parte și un format precis în ce-a de-a doua este dependent și de gradul de imprecizie al celui mai imprecis format. Astfel, din grafice, reiese că o combinație 16_16 este asemănătoare, ca și rezultate finale, unei combinații FP_16. De cealaltă parte, o combinație 32_32 este asemănătoare unei combinații 32_FP; în acest din urmă caz, diferențele dintre rezultate nu sunt la fel de mari ca în cazul în care se folosește un format de virgulă fixă 16:16.

Pe lângă aceste agregări prezentate anterior și graficele din anexa I reprezintă agregări după obiect ale metricii M%. Se poate astfel observa că se obțin rezultate mai bune pentru combinații de precizie ridicate, ceea ce era de așteptat, dar, în același timp, se observă o scădere a valorii metricii, pentru toate combinațiile de formate de reprezentare folosite, pentru obiectele cu indicele în intervalul 2000-2500. Acest lucru evidențiază încă o dată dependența combinației de formate folosite de natura obiectelor tridimensionale care urmează să fie randate.

BUPT

6.7 Concluzii preliminare

După cum am putut vedea în acest capitol, am reușit colectarea unui număr mare de obiecte tridimensionale, care acoperă o plajă mare de valori pentru datele de intrare ale unui pipeline grafic. Această colecție de obiecte tridimensionale poate fi oricând folosită ca și benchmark pentru alte studii legate de procesare datelor de către un pipeline grafic.

De asemenea, am putut vedea cum, folosind suita de module software descrise în capitolul precedent, am obținut un număr mare de imagini de referință (aproximativ 450000) și un număr și mai mare de imagini alterate (aproximativ 2700000). Totodată, suita de module software dezvoltată a ușurat procesul de obținere a acestor imagini, compararea lor, precum și obținerea valorilor metricilor și a graficelor aferente.

Astfel, am putut analiza și compara cu ușurință valorile calculate ale metricilor de interes. În plus, am definit și o metrică nouă, care să aibă o semantică precisă a valorilor și care să ajute la caracterizarea aproximativă a alterării unei imagini relativ la o imagine de referință.

În cele din urmă, am comparat valorile obținute pentru cele trei metrici atât pentru un set limitat de obiecte și combinații de precizie, cât și pentru toate obiectele din biblioteca creată. În continuare am creat o imagine de ansamblu asupra valorilor metricilor pentru fiecare combinație de precizie în parte, pentru toate obiectele, prin agregarea valorilor metricilor folosind media aritmetică. Toate aceste exemple și comparații au fost efectuate prin utilizarea de grafice ale valorilor metricilor, ușurând astfel înțelegerea lor.

În cele din urmă, toate aceste comparații și exemple susțin avantajele metricii relative M% în fața metricilor absolute MSE și PSNR, atunci când vine vorba de imagini cu o dispersie uniformă a diferențelor, așa cum s-a putut vedea și în cazul imaginilor produse de pipeline-ul grafic.

BUPT

Capitolul 7

Concluzii și perspective

În această teză am prezentat optimizări care pot fi aduse unui pipeline grafic pentru a îmbunătăți viteza de execuție sau consumul de putere, în funcție de nevoile particulare ale aplicației țintă, din ce în ce mai solicitante în contextul dispozitivelor mobile tot mai populare în zilele noastre și în viitor.

În primul rând am prezentat o serie de operatori matriciali destinați optimizării operațiilor aritmetice matriciale din primele etape ale unui pipeline grafic. Acești operatori sunt proiectați și implementați pentru FPGA-uri, tocmai pentru a veni ca soluție nevoii stringente de îmbunătățire într-un ritm alert a dispozitivelor hardware actuale.

În continuare, am studiat impactul folosirii combinațiilor de precizie de reprezentare a numerelor reale în același pipeline grafic. De asemenea, am încercat să construim o bibliotecă cu un număr sporit de obiecte tridimensionale care să fie proiectate utilizând un pipeline grafic hibrid și să măsurăm în mod obiectiv alterarea imaginile bidimensionale produse, relativ la cele produse de un pipeline grafic nealterat. Pentru a realiza acest studiu am folosit metrici deja existente și, în plus, am propus o nouă metrică cu o semantică clară și care să fie ușor de calculat din punct de vedere aritmetic.

7.1 Contribuții originale

În prima parte a tezei, ne-am axat pe studierea și înțelegerea contextului actual al dispozitivelor hardware reconfigurabile, în special cele care folosesc FPGA-uri pentru implementarea diverșilor algoritmi matematici necesari în contextul unui pipeline grafic.

Am trecut apoi la proiectarea, implementarea și validarea unor operatori matriciali extrem de folosiți într-un pipeline grafic, cum sunt operatorii geometrici de translație și scalare.

BUPT

Am proiectat astfel o serie de operatori matriciali, implementați pe FPGA, care pot fi configurați în momentul sintezei pe următoarele planuri:

- Dimensiunea numerelor reale. Se poate configura dimensiunea mantisei și a exponentului reprezentării, putând astfel fi folosite și formate non-standard.
- Frecvența țintă pentru generare. Prin acest parametru putem influența adâncimea pipeline-ului operatorului generat.
- Gradul de paralelizare. Acest parametru stabilește numărul de micro-operații care se execută simultan în cadrul operatorului.

În urma acestei proiectări, am implementat operatorii folosind biblioteca FloPoCo, rezultând un număr total de 36 de operatori pentru fiecare transformare geometrică. De asemenea, folosind infrastructura oferită de biblioteca folosită, am dezvoltat o suită de teste care acoperă atât cazurile speciale cât și cazurile generale pentru domeniul de valori.

În continuare, pentru a putea extinde setul de operatori creați, am studiat influența folosirii mai multor formate de reprezentare a numerelor reale în stagiile unui pipeline grafic. Astfel, am modificat o implementare software proprie a specificației OpenGL ES 1.1 care să permită schimbarea formatelor de numere reale folosite, chiar în timpul rulării. De asemenea, am creat o suită de programe software care să ne ajute să realizăm următoarele:

- Colectarea obiectelor tridimensionale folosite ca date de intrare pentru pipeline-ul grafic. Am creat astfel un modul software care caută și descarcă fișiere cu conținutul căutat.
- Transformarea datelor de intrare într-un format recunoscut de pipeline-ul grafic.
- O aplicație software care încorporează pipeline-ul grafic și îi furnizează datele de intrare pentru a produce proiecțiile obiectelor tridimensionale. În funcție de combinația de precizie folosită, această aplicație produce atât imaginile de referință cât și cele alterate.
- Compararea imaginilor alterate, obținute prin folosirea unor combinații imprecise de precizie, cu cele de referință, obținute prin folosirea unui format considerat precis.
- Analizarea imaginilor obținute și generarea de valori ale unor metrici pentru diferențele imaginilor alterate, relativ la cele de referință
- Centralizarea, filtrarea, prezentarea și interpretarea valorilor metricilor obținute

BUPT

Folosind modulele software anterioare am creat și analizat o serie de valori ale unor metrici deja existente. Pentru a da o semantică clară acestor valori, am simțit nevoia de a introduce o nouă metrică. Astfel, am introdus o metrică procentuală a cărei valoare reprezintă procentul de pixeli diferiți din imaginea alterată relativ la numărul de pixeli utili din imaginea de referință. Această metrică are o variație a valorilor similară ca formă cu variația valorilor MSE, una din metricile deja existente pe care am folosit-o pentru evaluarea obiectivă a diferențelor imaginilor rezultate. Pe lângă aceste două metrici, am folosit și metrica PSNR, des utilizată în evaluarea zgomotului introdus în imaginile digitale.

Astfel, am prezentat o serie de grafice care să evidențieze trăsăturile fiecărei metrici relativ la imaginile produse de pipeline-ul grafic instrumentat. Acest lucru ne-a ajutat să înțelegem și să analizăm influențele combinației de precizie folosite asupra rezultatului final produs de pipeline.

Concluzionând, pe parcursul cercetării aferente elaborării acestei teze, am obținut următoarele rezultate concrete:

- Am realizat o arhitectură de operatori geometrici de translație și scalare care pot opera cu diferite formate de numere reale reprezentate în virgulă flotantă, inclusiv formate nestandard. De asemenea, aceste arhitecturi se pot configura astfel încât să rezulte diferiți operatori, în funcție de formatul ales, adâncimea dorită a pipeline-lor interne sau frecvența țintă.

- Am creat o bibliotecă de obiecte tridimensionale care pot fi folosite ca date de intrare pentru un pipeline grafic.

- Am creat un proces și suita de module software aferente cu ajutorul cărora am studiat și înțeles influența combinațiilor de precizie folosite în pipeline-ul grafic modificat asupra rezultatelor finale.

- Am propus o nouă metrică cu o semantică a valorilor clară, care să ajute la evaluarea obiectivă a diferențelor dintre imaginile alterate și cele de referință.

7.2 Perspective de viitor

Din punct de vedere al acțiunilor care pot fi întreprinse în viitor pentru a continua cercetarea începută și prezentată în această teză, menționăm următoarele:

- Crearea de operatori matriciali configurabili pentru transformarea geometrică de rotație, precum și pentru alte transformări geometrice afine utilizate în pipeline-urile grafice.

BUPT

- Crearea de unități de calcul implementate pe FPGA care să transpună hardware conceptul prezentat prin modificarea pipeline-ului grafic software.
- După implementarea hardware a conceptului, să efectuăm o analiză comparativă cu soluții deja existente, din punct de vedere al performanțelor obținute.
- Utilizarea combinațiilor de formate pe o scară mai restrânsă; astfel, ar putea fi posibilă utilizarea de mai multe formate de reprezentări ale numerelor fracționare în cadrul aceluiași pipeline.
- Definirea unui prag pentru M% astfel încât valori mai mari decât acest prag să indice o imagine inacceptabilă de către un utilizator uman.

BUPT

Anexa A

Histograme caracteristici obiecte tridimensionale

BUPT

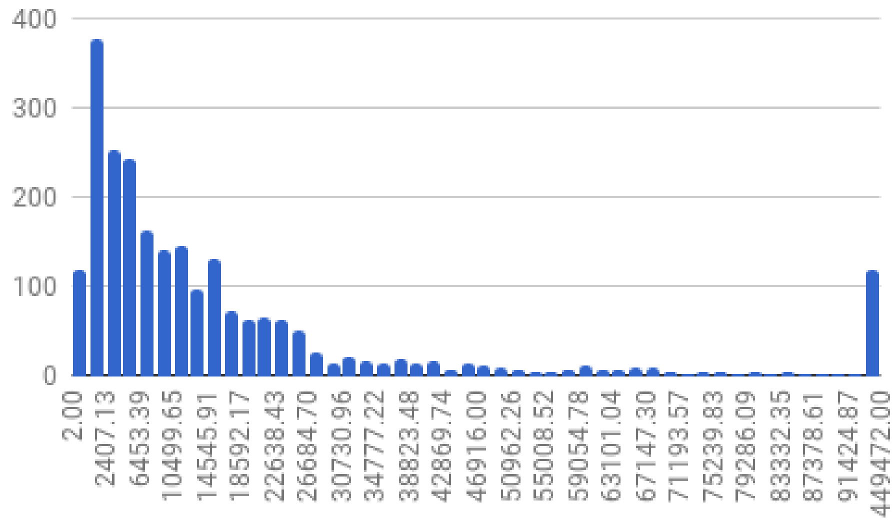


Figura A.1: Histogramă distribuție triunghiuri

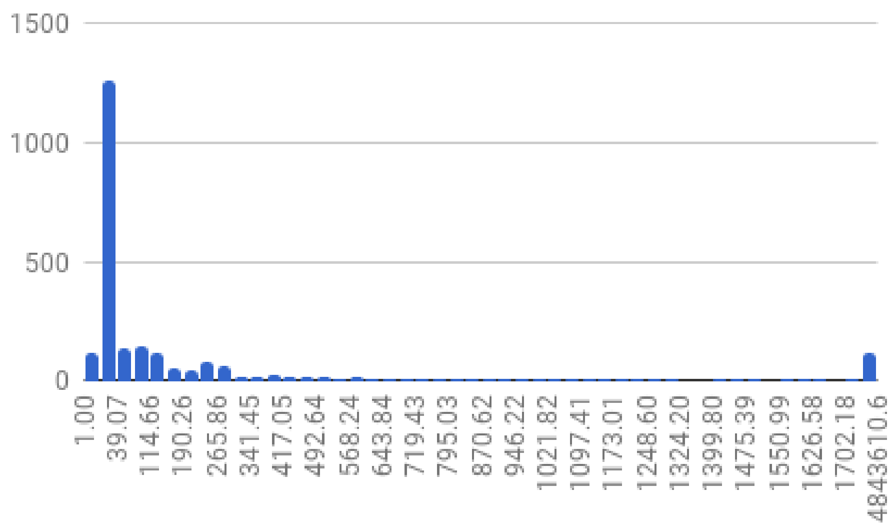


Figura A.2: Histogramă distribuție dimensiune volum

BUPT

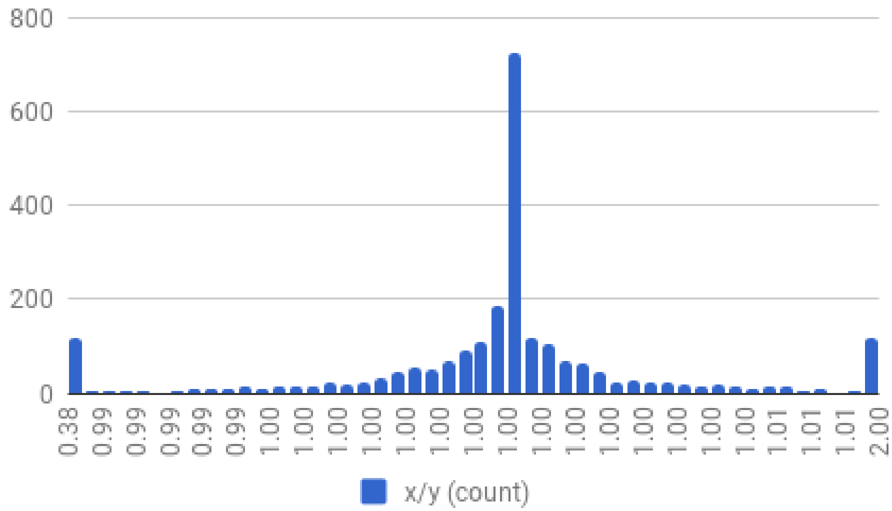


Figura A.3: Histogramă distribuție raport x/y

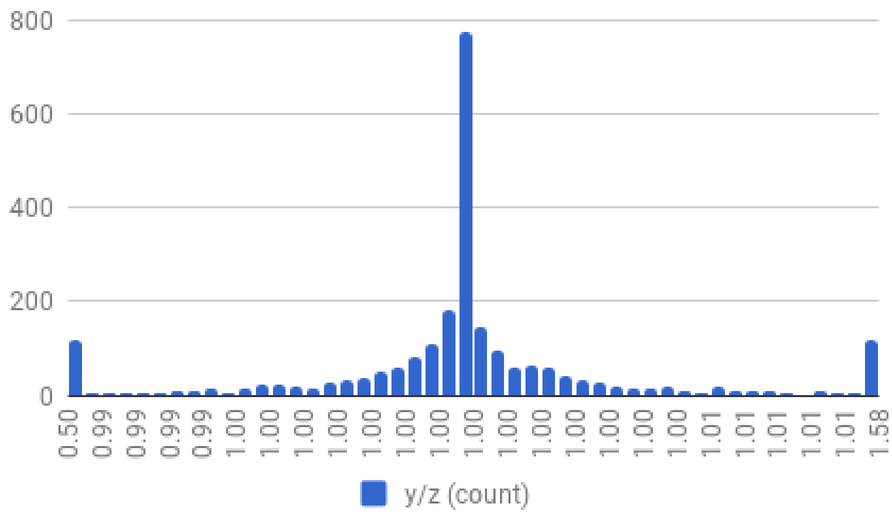


Figura A.4: Histogramă distribuție raport y/z

BUPT

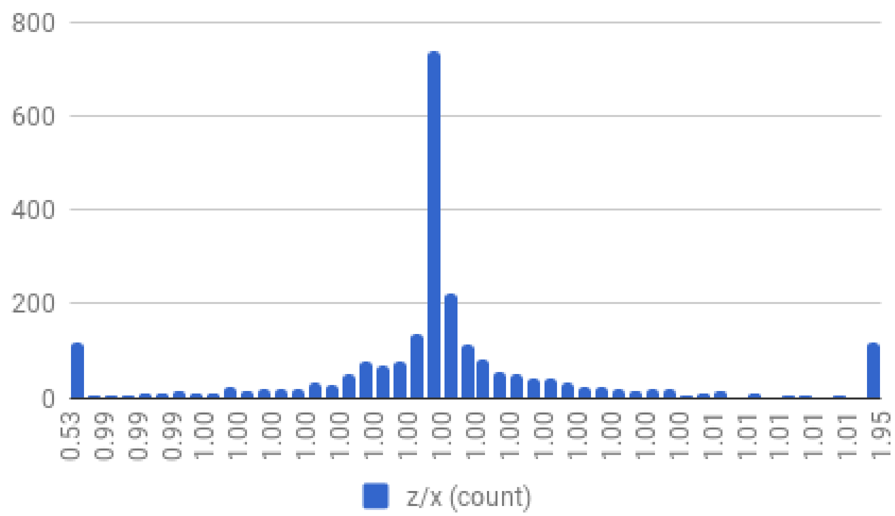


Figura A.5: Histogramă distribuție raport z/x

BUPT

Anexa B

Suită de cadre alterate 16_16

BUPT

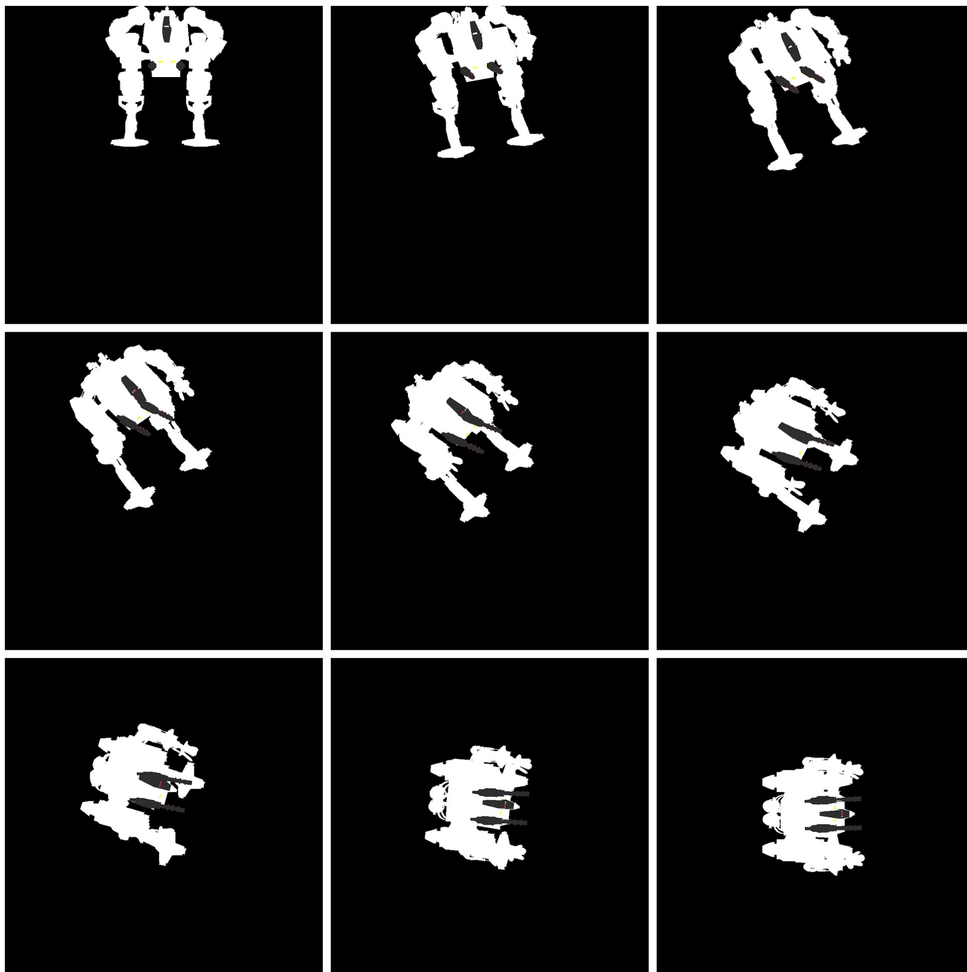


Figura B.1: Suită de cadre de referință(FP_FP)

BUPT

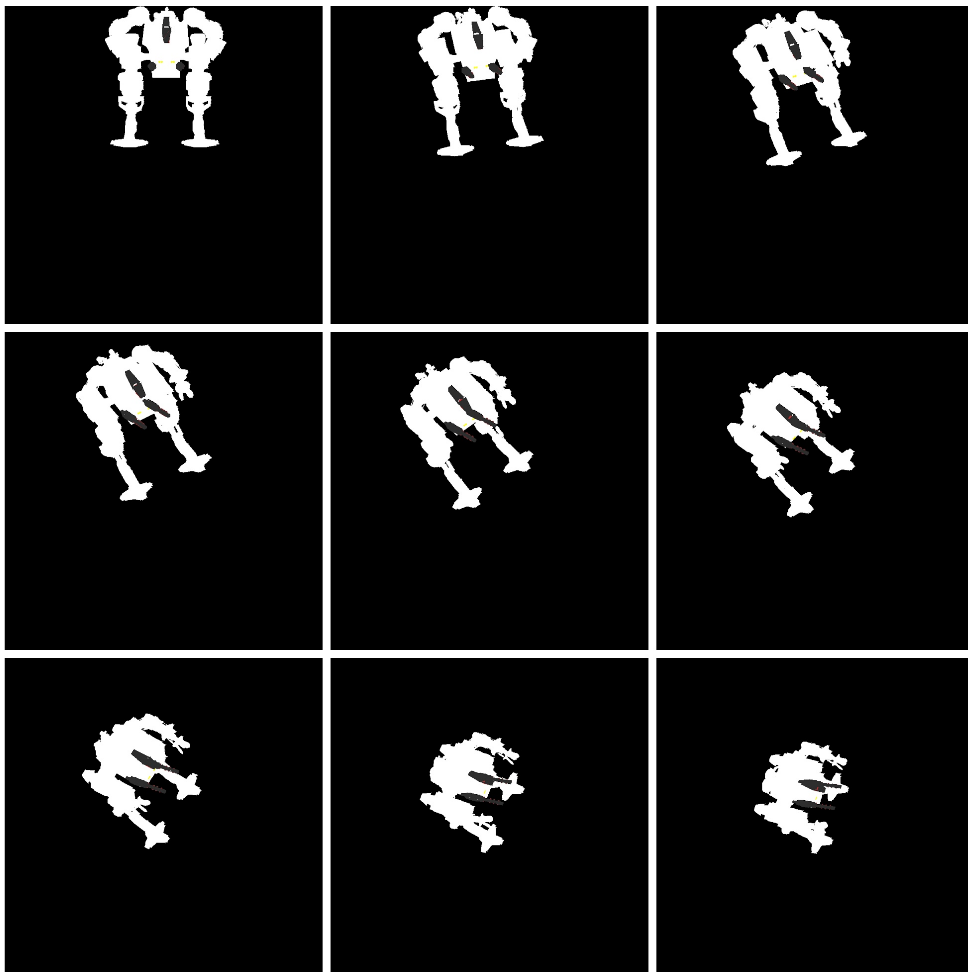


Figura B.2: Suită de cadre alterate(16_16)

BUPT

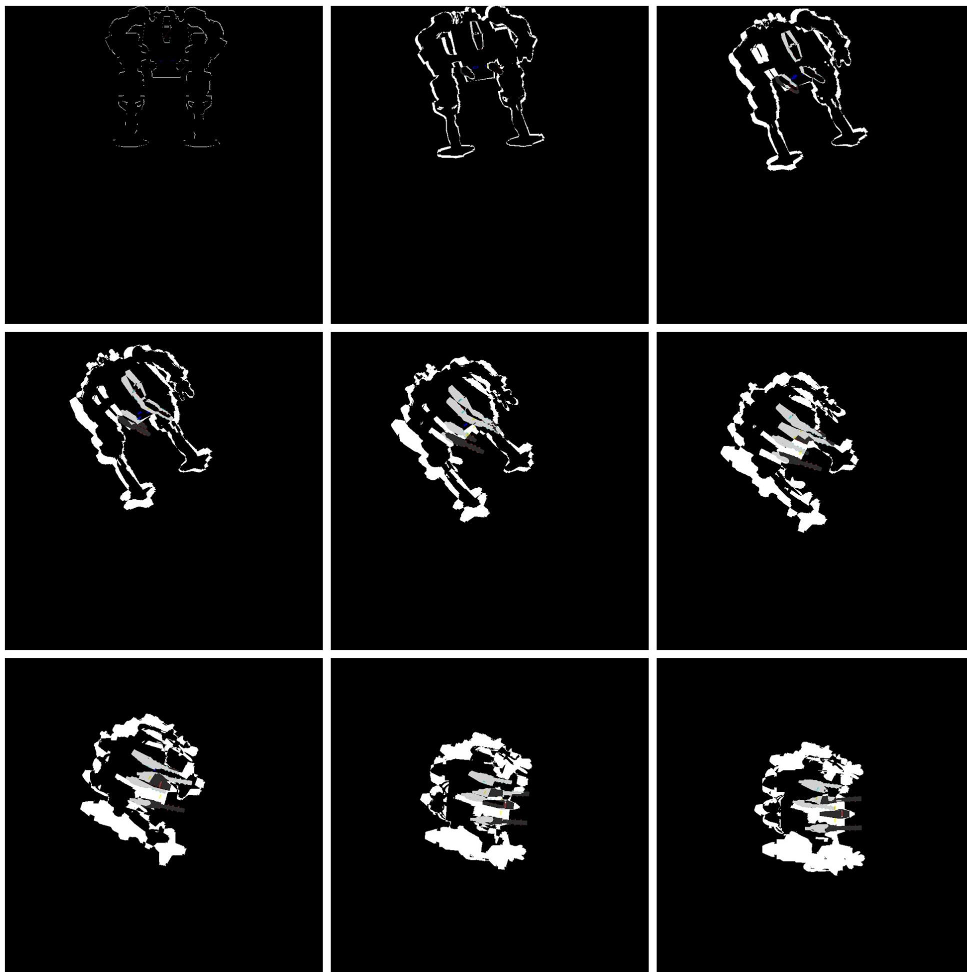


Figura B.3: Suită de cadre diferență(FP_FP vs 16_16)

BUPT

Anexa C

Suită de cadre alterate 32_32

BUPT

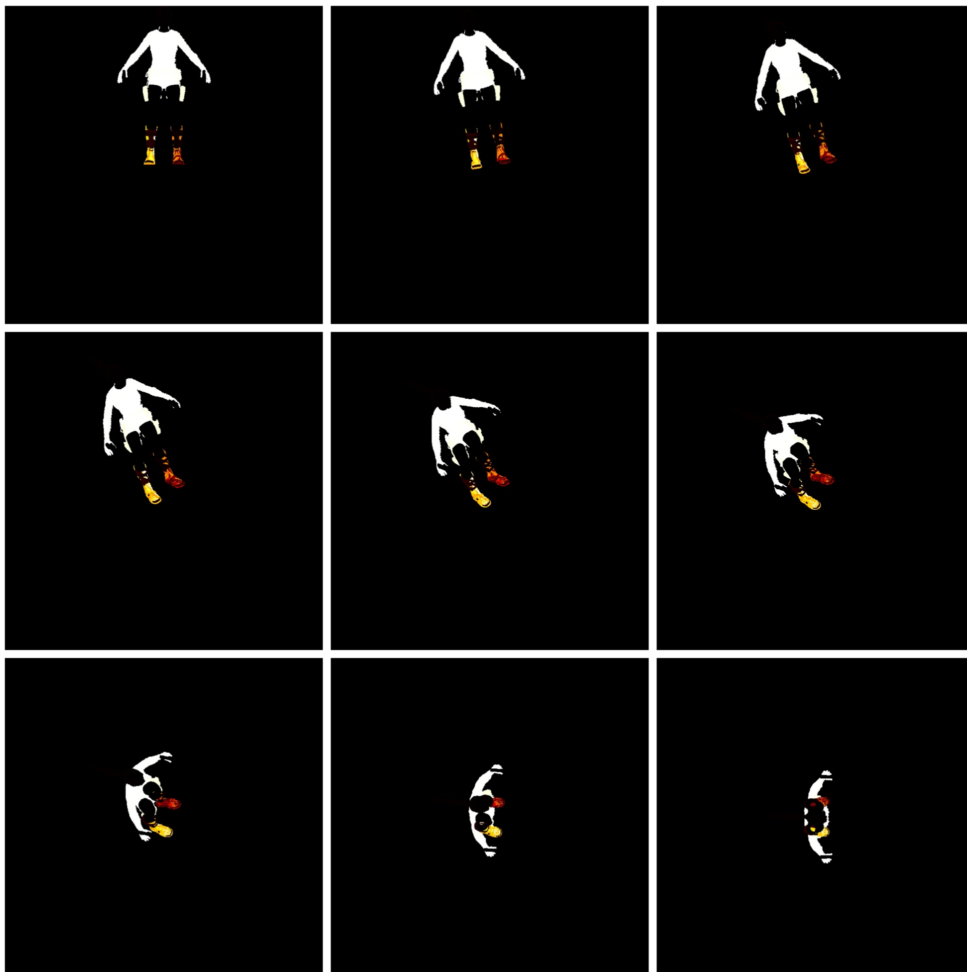


Figura C.1: Suită de cadre de referință(FP_FP)

BUPT

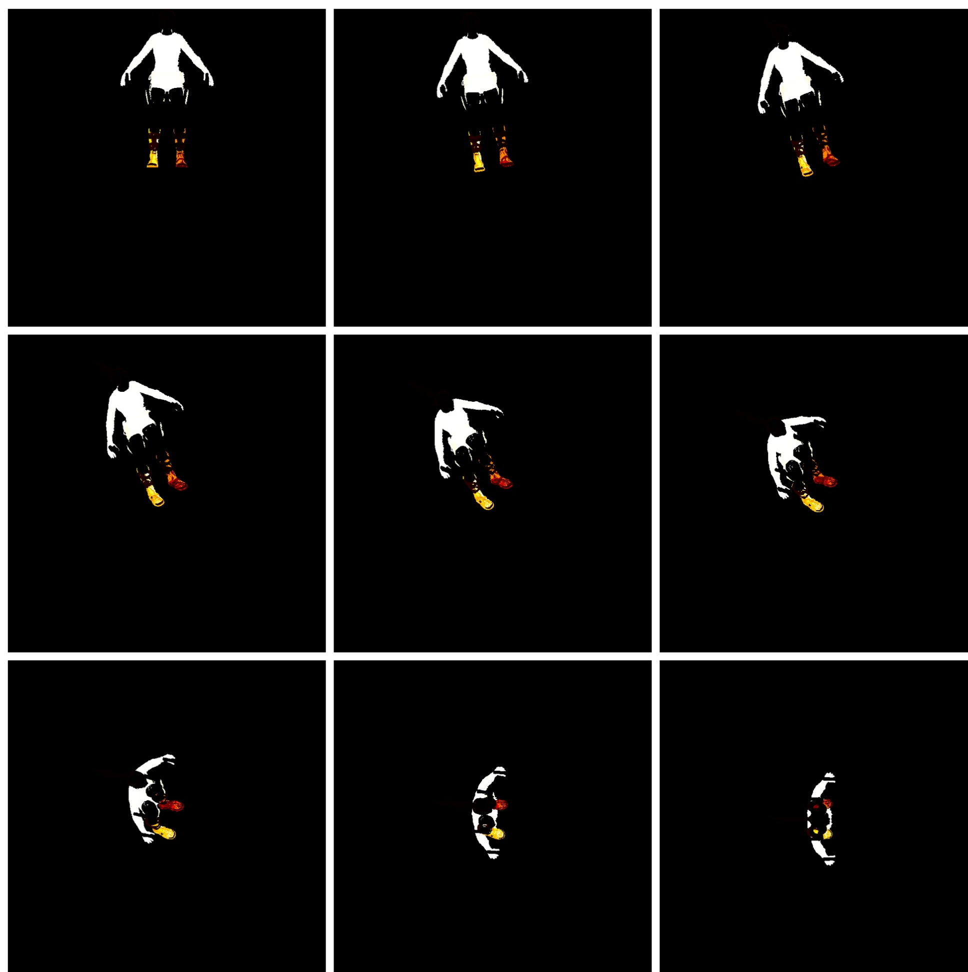


Figura C.2: Suită de cadre alterate(format 32_32)

BUPT

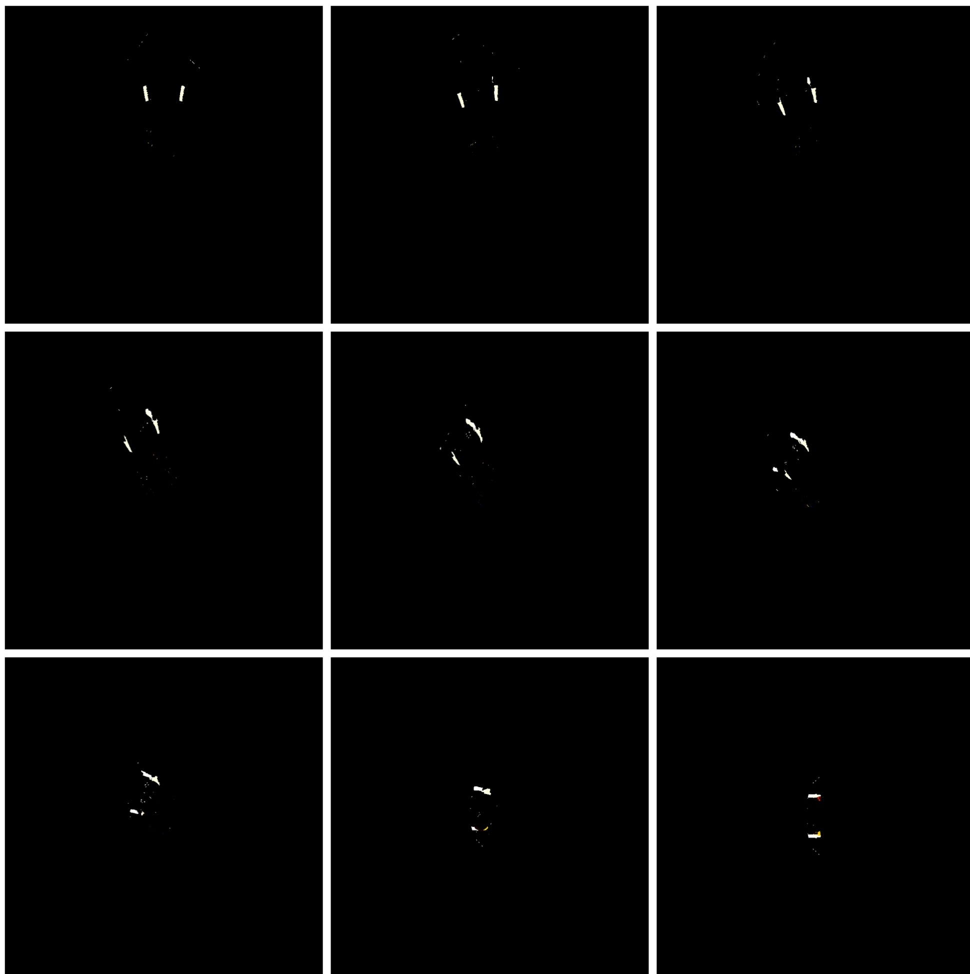


Figura C.3: Suită de cadre diferență(FP_FP vs 32_32)

BUPT

Anexa D

Imagini Rezultate 1

BUPT

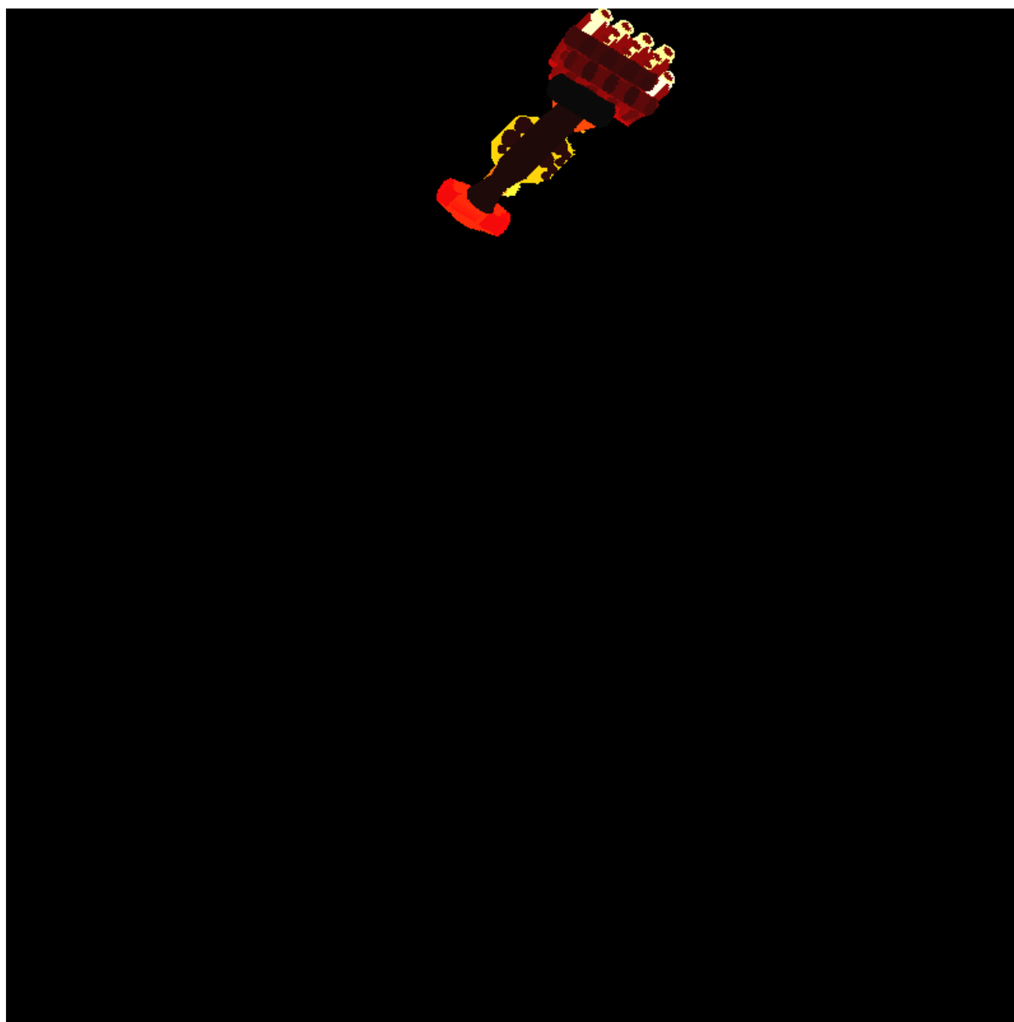


Figura D.1: Imagine de referință

BUPT

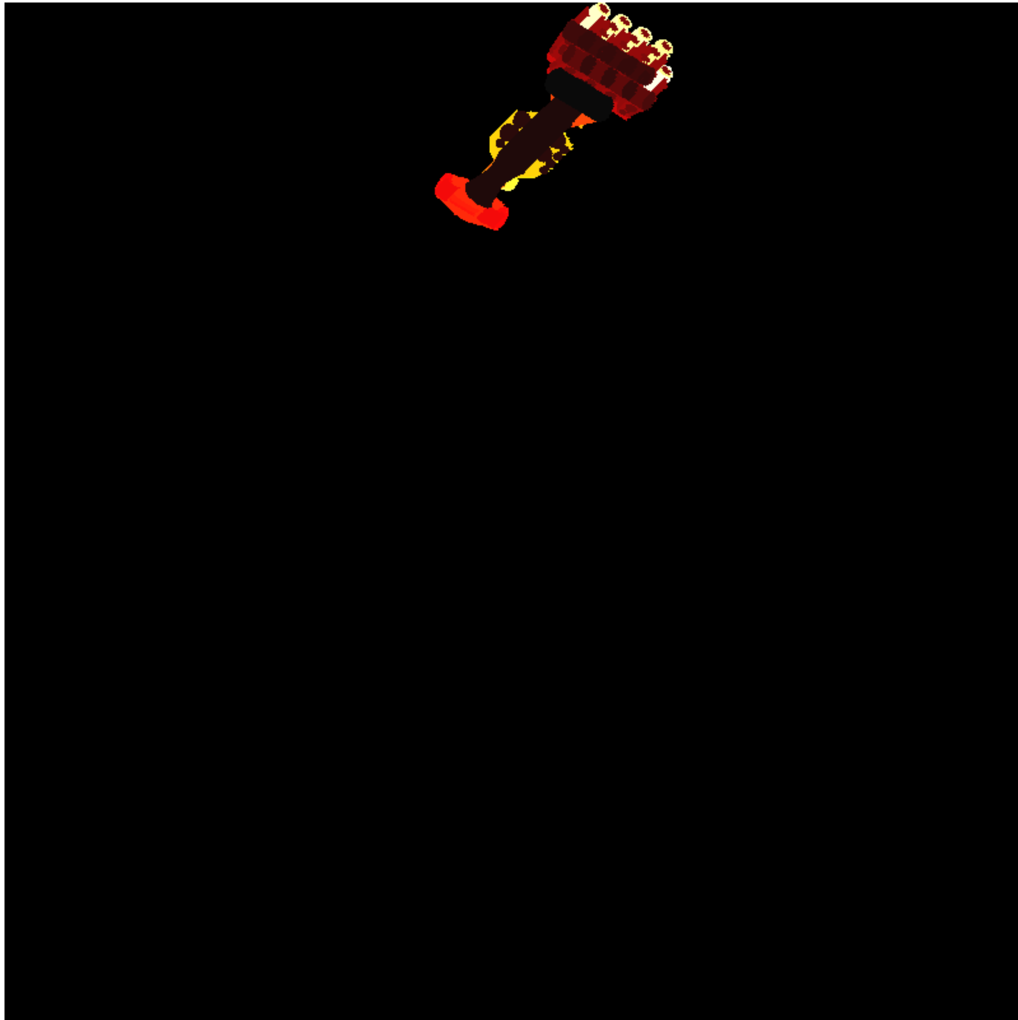


Figura D.2: Imagine alterată

BUPT

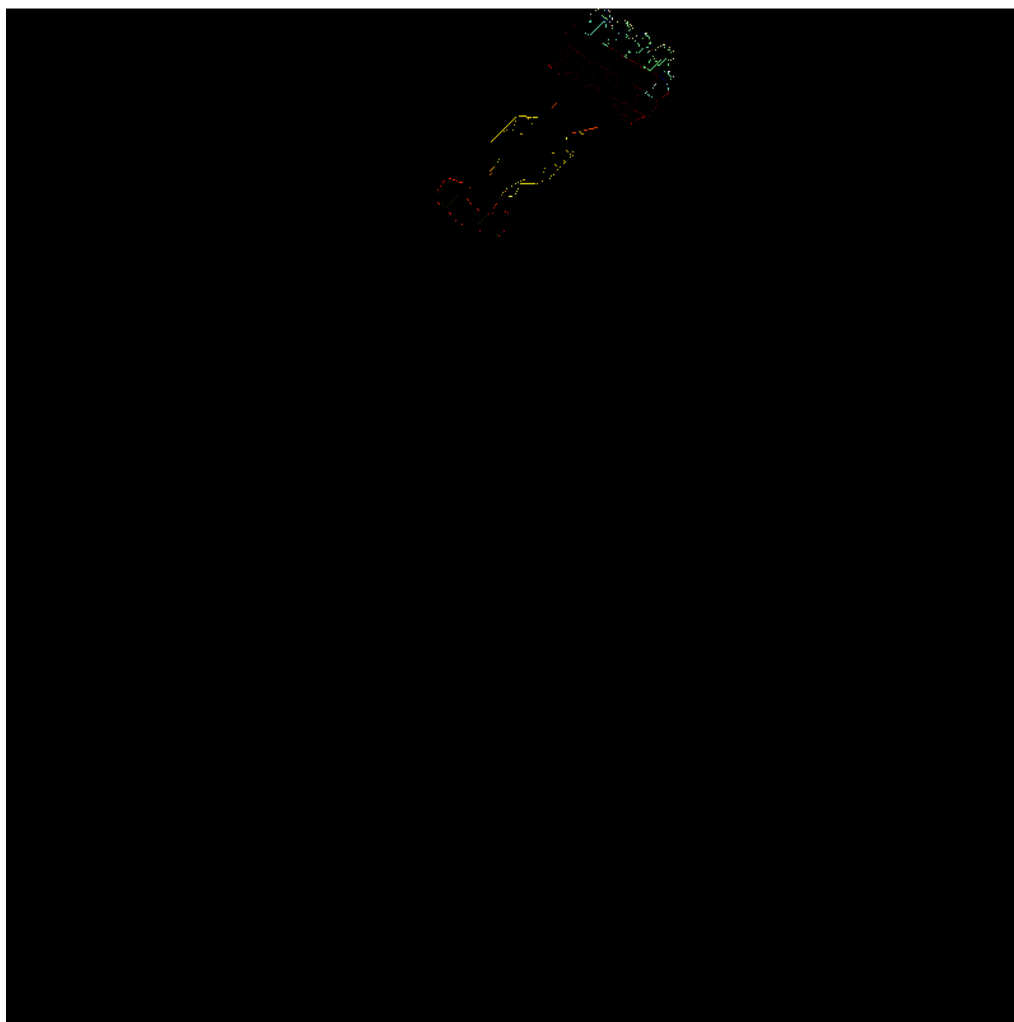


Figura D.3: Imagine diferență

BUPT

Anexa E

Imagini Rezultate 2

BUPT

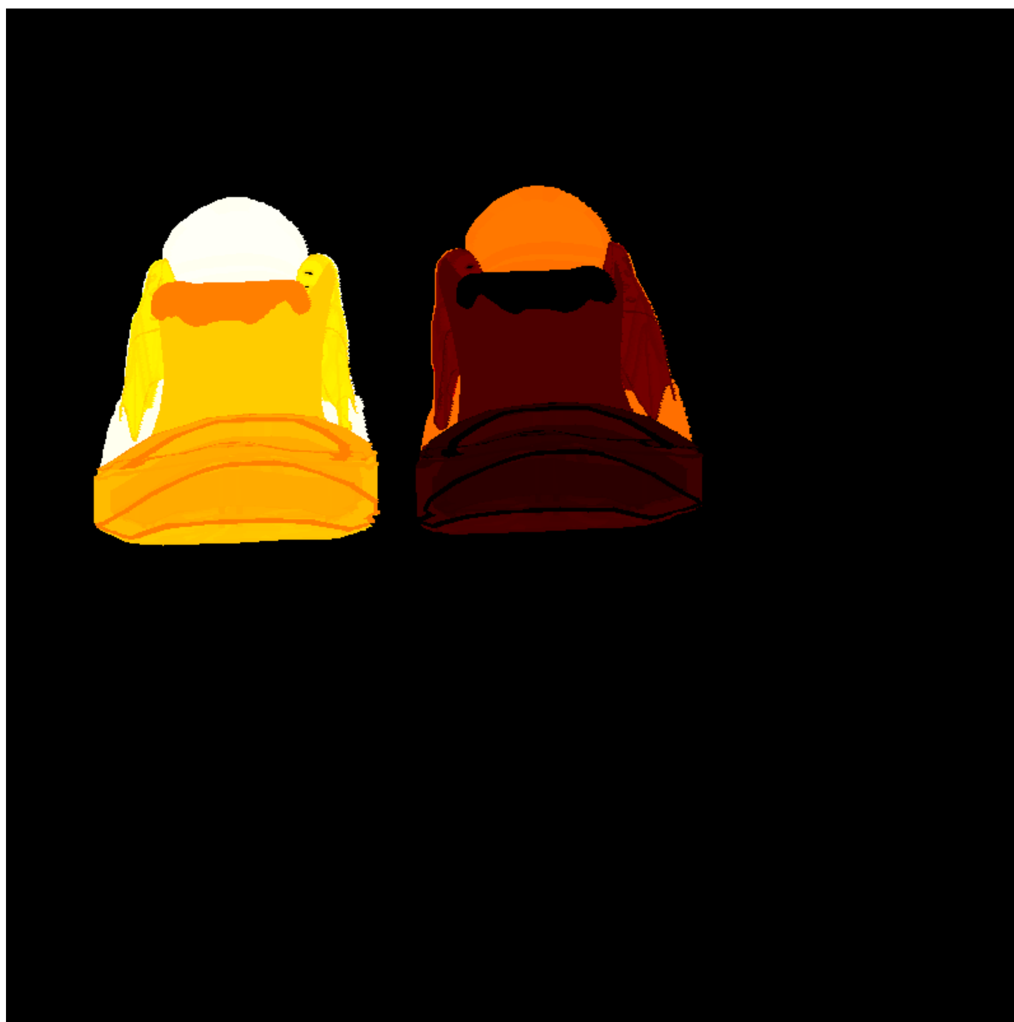


Figura E.1: Imagine de referință

BUPT

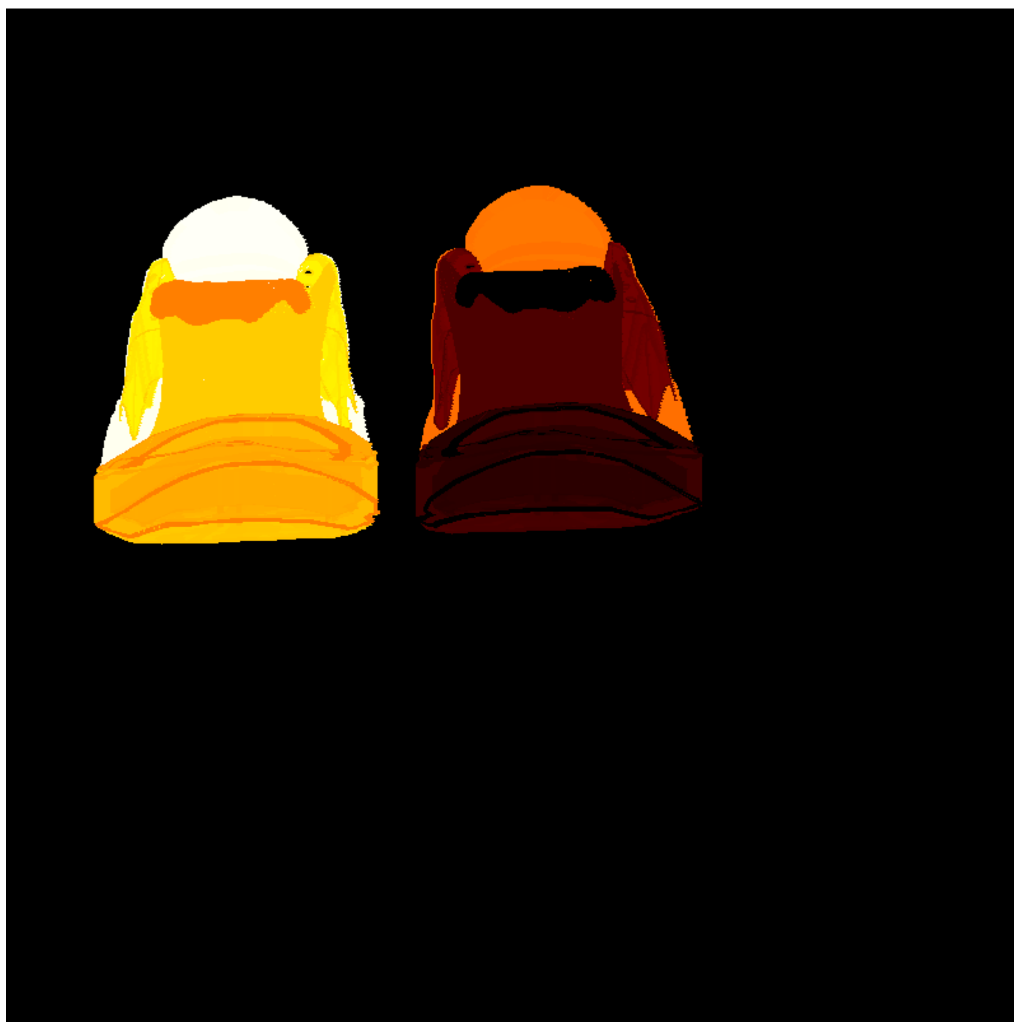


Figura E.2: Imagine alterată

BUPT

Figura E.3: Imagine diferență

BUPT

Anexa F

Grafice valori MSE și PSNR

În această anexă sunt prezentate o serie de grafice cu valorile metricilor MSE și PSNR, pentru o suită de 180 de cadre ale câte unui obiect. Fiecare astfel de grafic reprezintă valorile obținute pentru metrici asupra imaginilor alterate, relativ la cele de referință folosind combinația specificată de format de reprezentare a numerelor reale.

BUPT

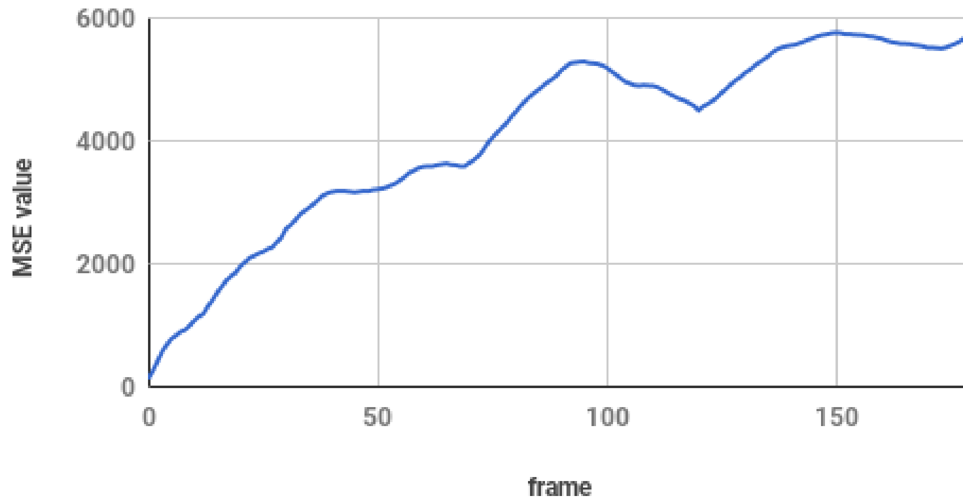


Figura F.1: Grafic valori MSE: obiect 0074, format 16_16

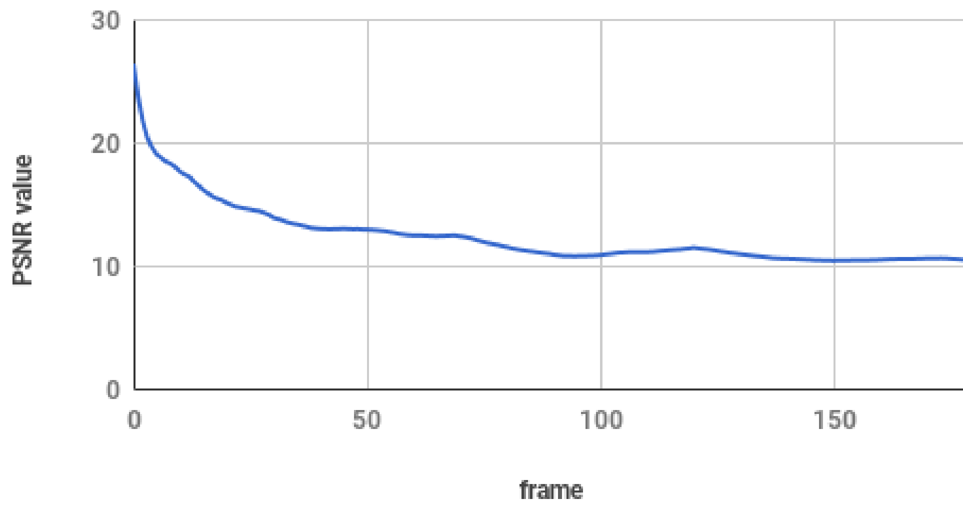


Figura F.2: Grafic valori PSNR: obiect 0074, format 16_16

BUPT

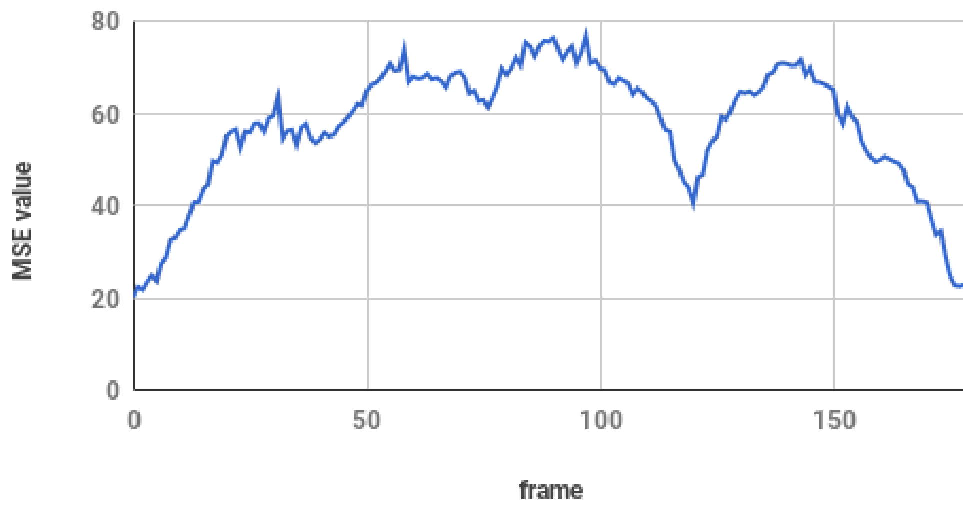


Figura F.3: Grafic valori MSE: obiect 0074, format 32_32

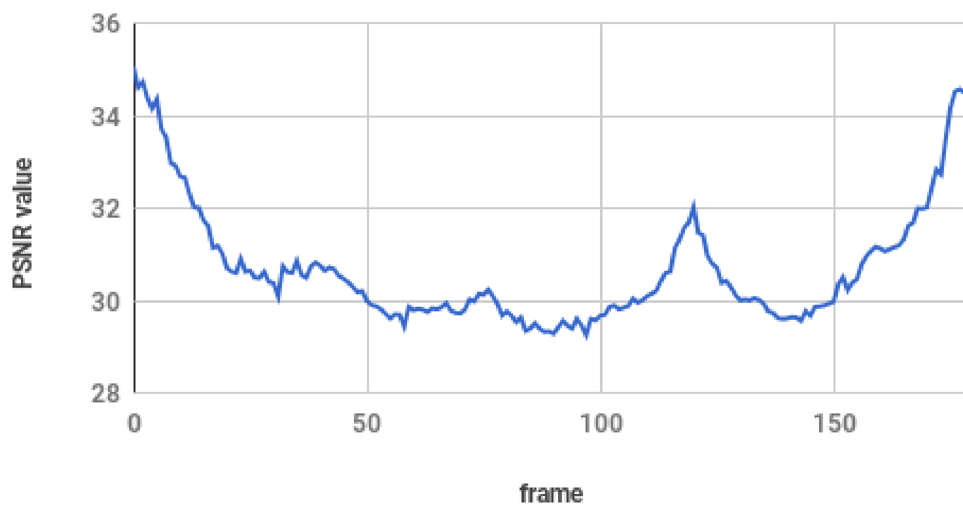


Figura F.4: Grafic valori PSNR: obiect 0074, format 32_32

BUPT

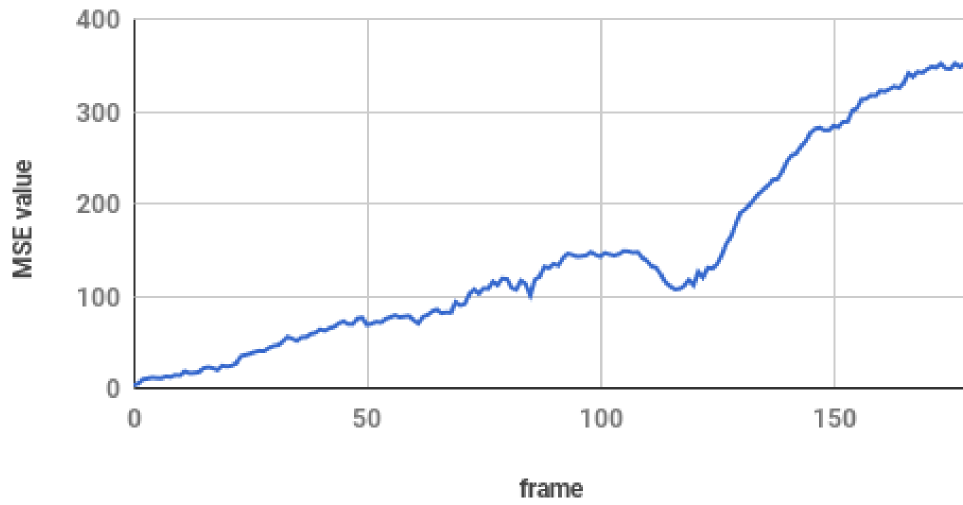


Figura F.5: Grafic valori MSE: obiect 0081, format 16_16

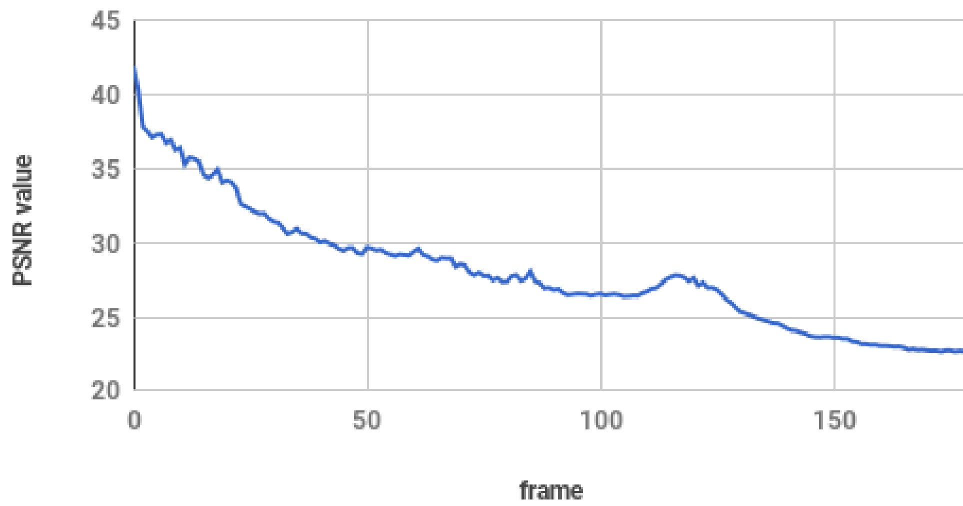


Figura F.6: Grafic valori PSNR: obiect 0081, format 16_16

BUPT

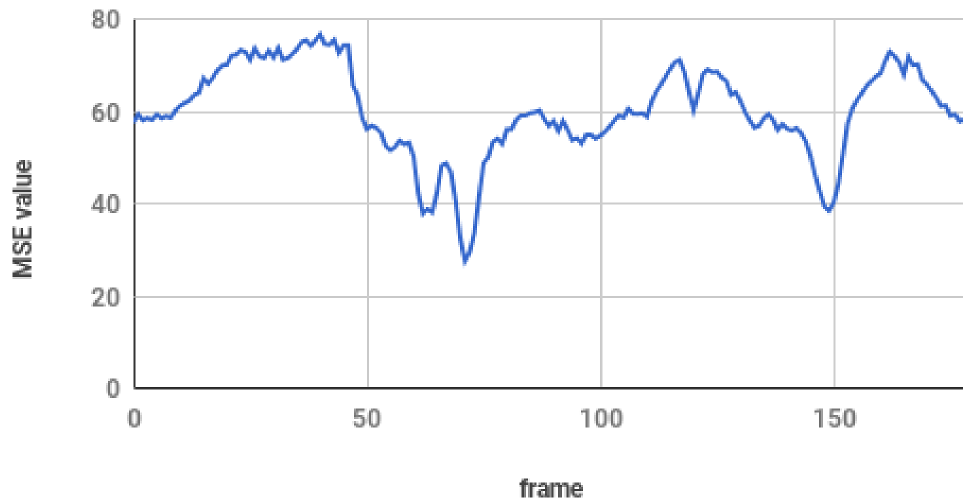


Figura F.7: Grafic valori MSE: obiect 0081, format 32_32

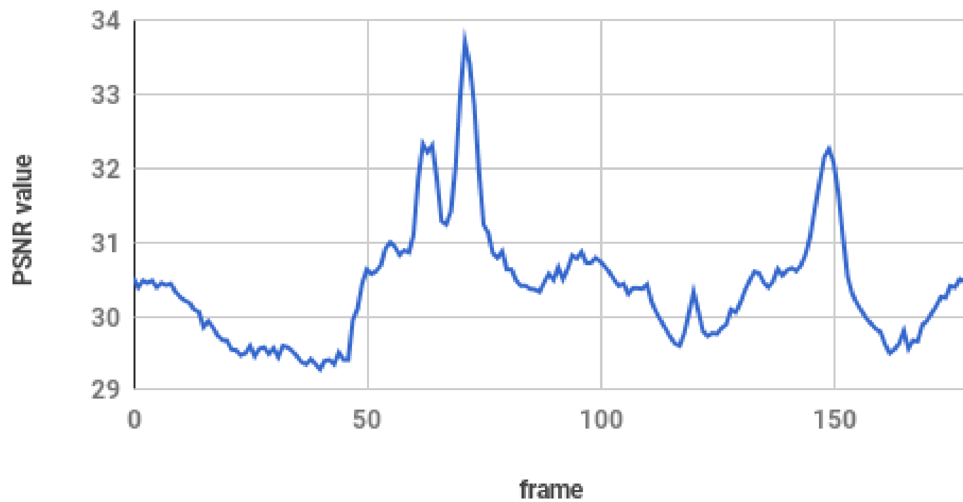


Figura F.8: Grafic valori PSNR: obiect 0081, format 32_32

BUPT

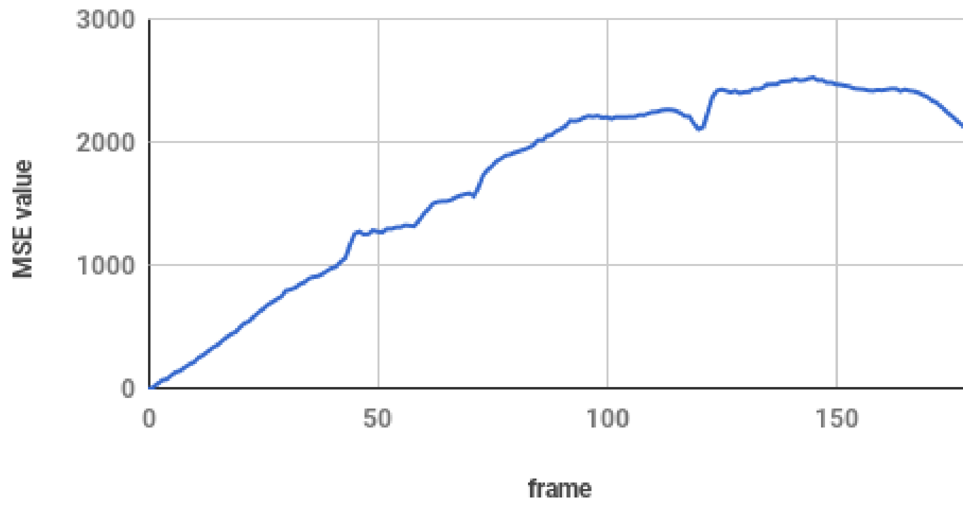


Figura F.9: Grafic valori MSE: obiect 0115, format 16_16

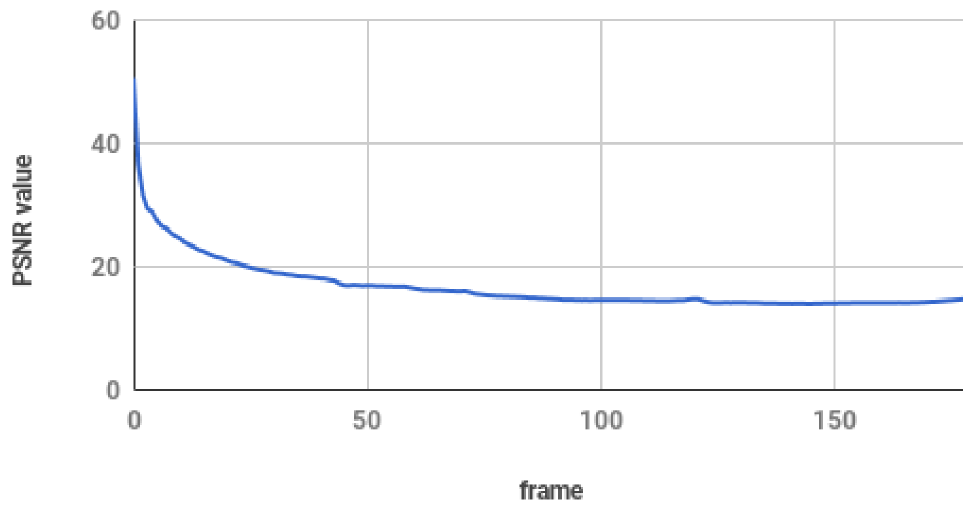


Figura F.10: Grafic valori PSNR: obiect 0115, format 16_16

BUPT

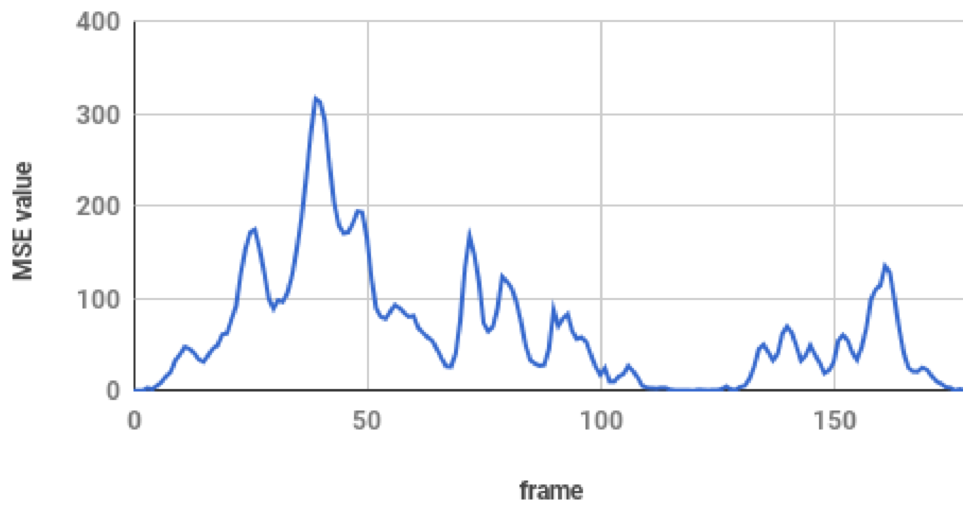


Figura F.11: Grafic valori MSE: obiect 0115, format 32_32

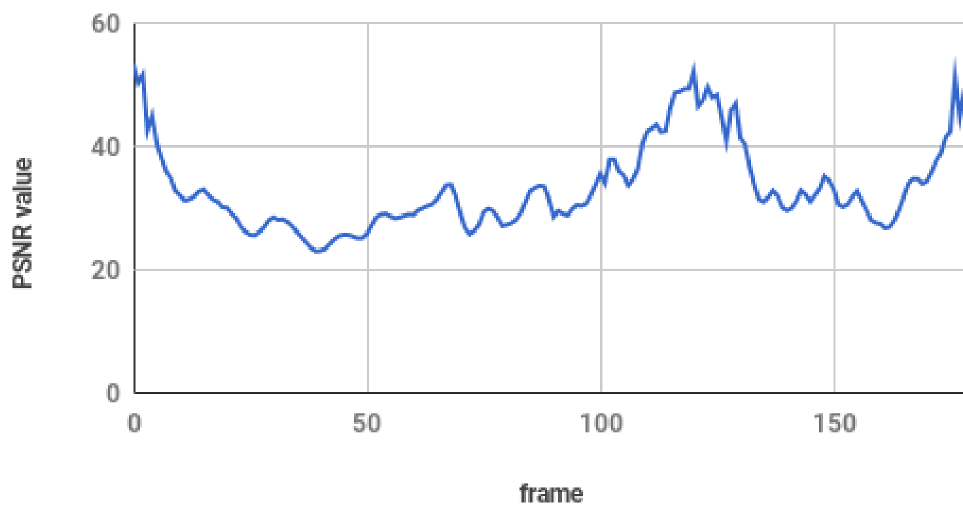


Figura F.12: Grafic valori PSNR: obiect 0115, format 32_32

BUPT

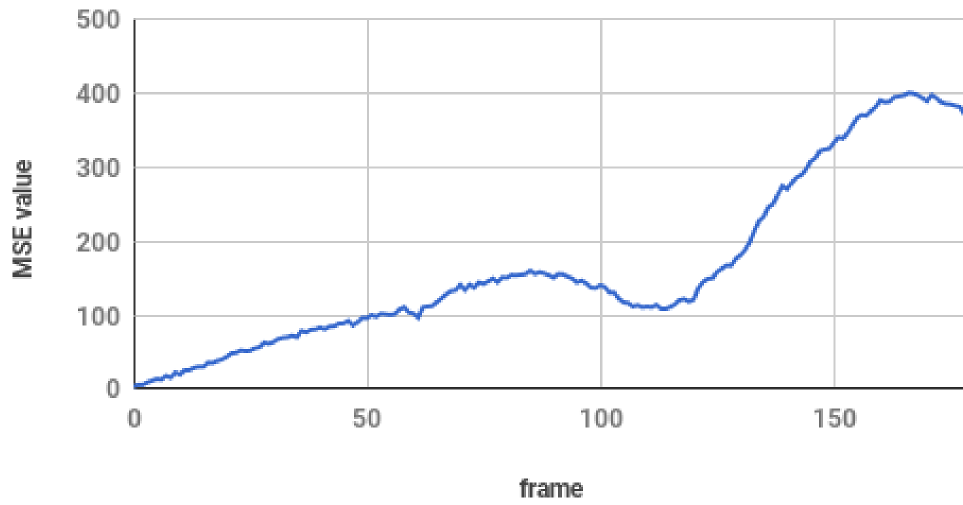


Figura F.13: Grafic valori MSE: obiect 0538, format 16_16

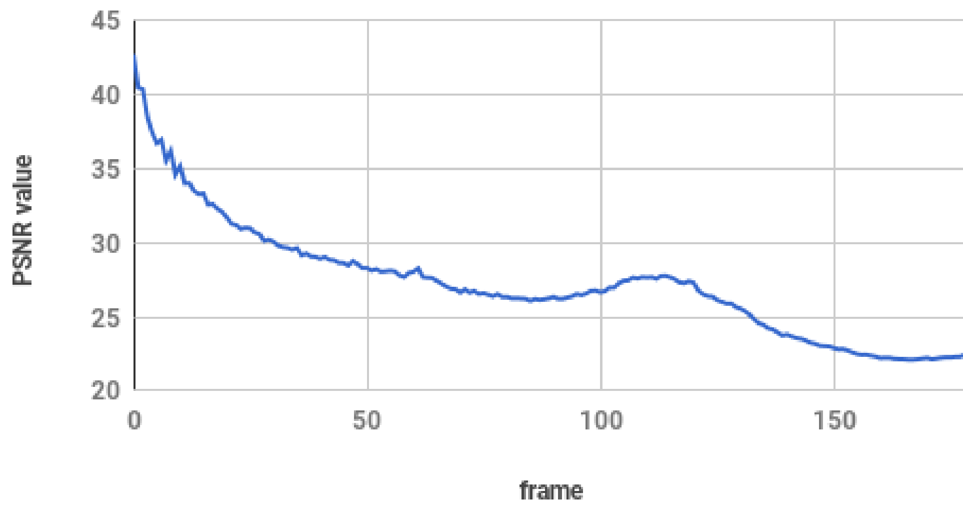


Figura F.14: Grafic valori PSNR: obiect 0538, format 16_16

BUPT

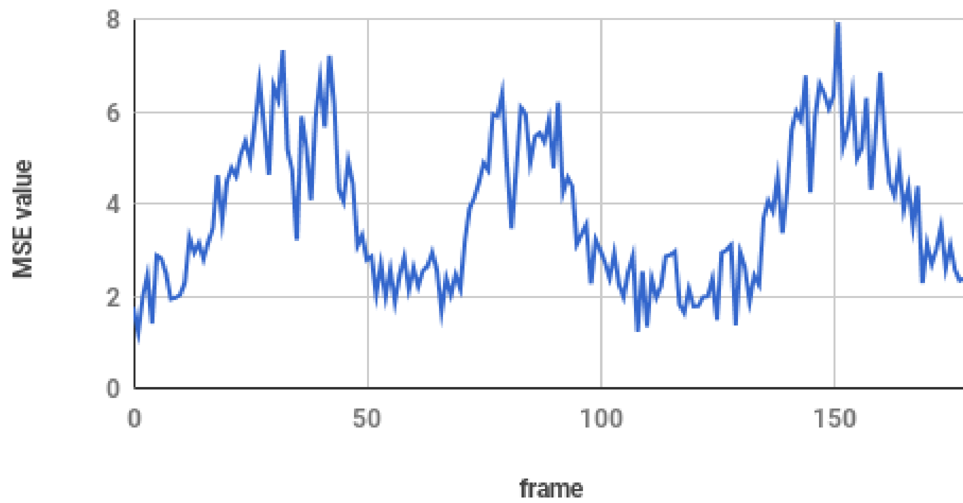


Figura F.15: Grafic valori MSE: obiect 0538, format 32_32

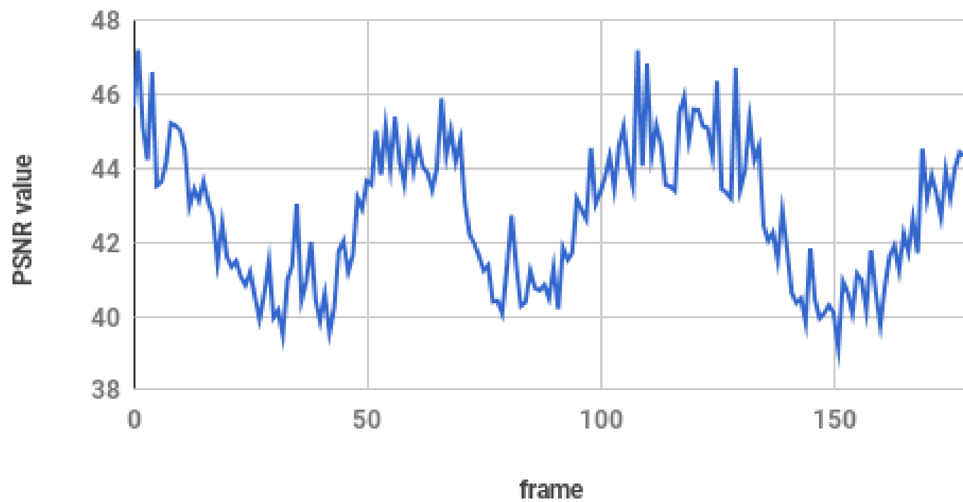


Figura F.16: Grafic valori PSNR: obiect 0538, format 32_32

BUPT

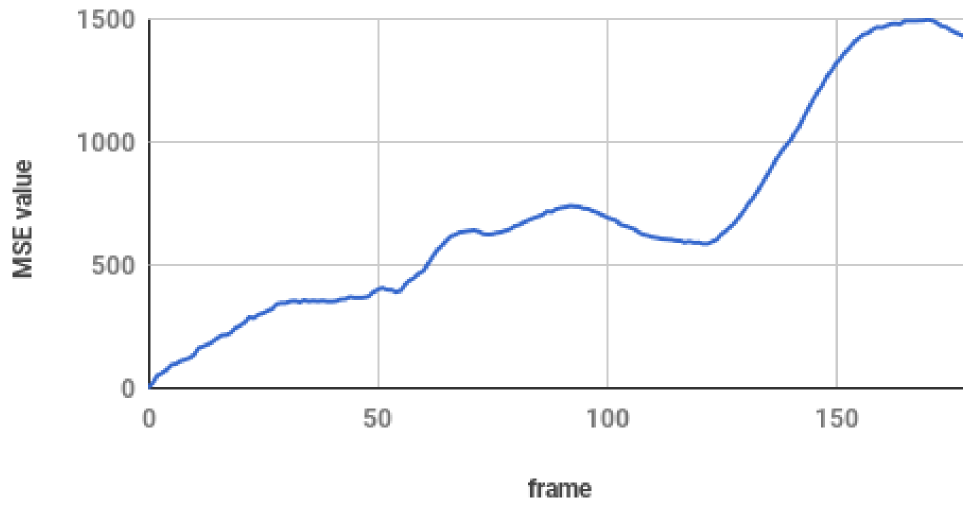


Figura F.17: Grafic valori MSE: obiect 1886, format 16_16

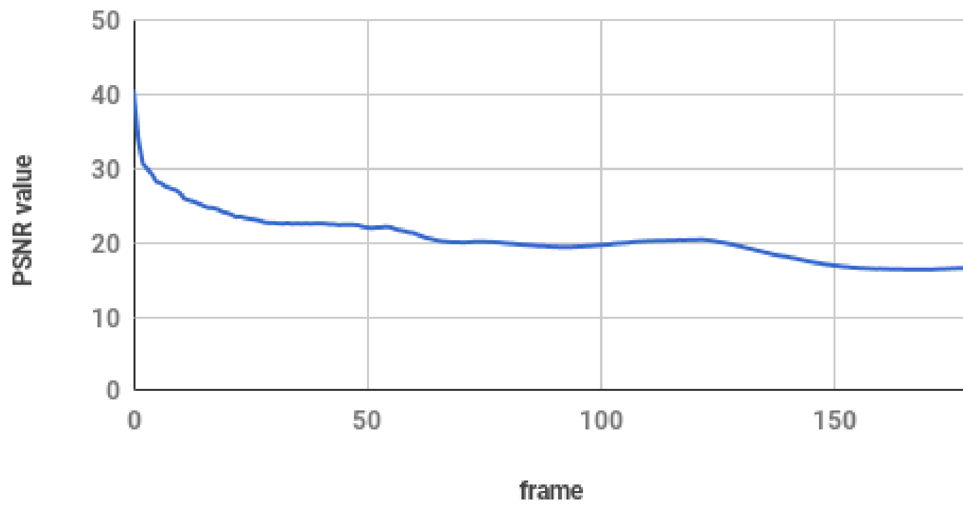


Figura F.18: Grafic valori PSNR: obiect 1886, format 16_16

BUPT

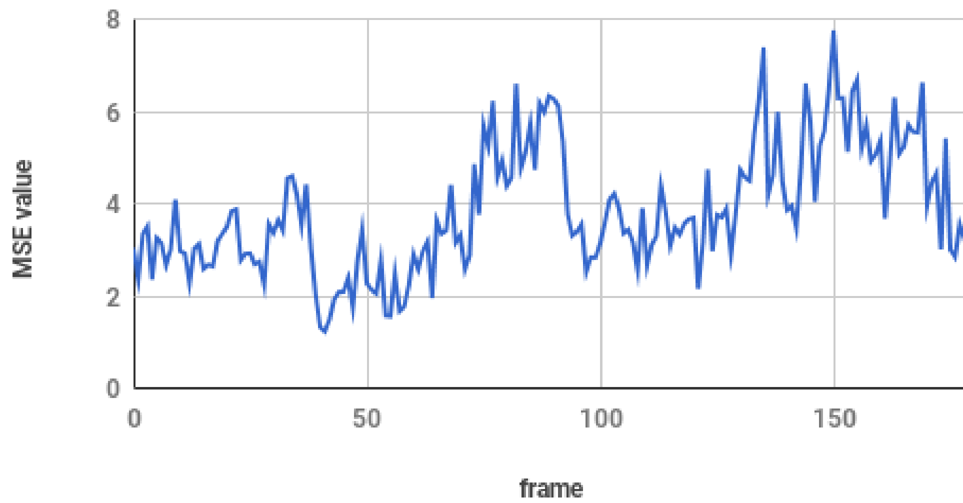


Figura F.19: Grafic valori MSE: obiect 1886, format 32_32

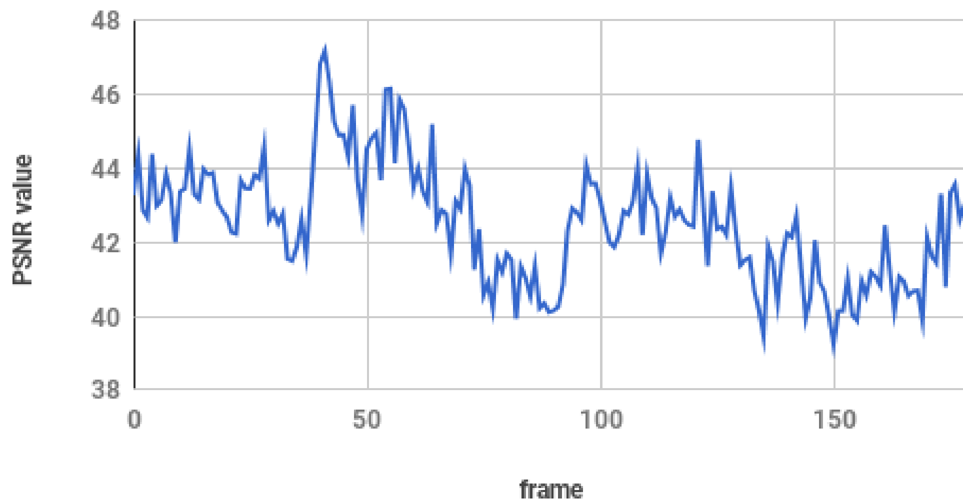


Figura F.20: Grafic valori PSNR: obiect 1886, format 32_32

BUPT

Anexa G

Grafice valori M%

În această anexă sunt prezentate grafice cu valori ale metricii M%, pentru o suită de 180 de cadre ale câte unui obiect. Fiecare astfel de grafic reprezintă valorile obținute pentru metrici asupra imaginilor alterate, relativ la cele de referință folosind combinația specificată de format de reprezentare a numerelor reale.

BUPT

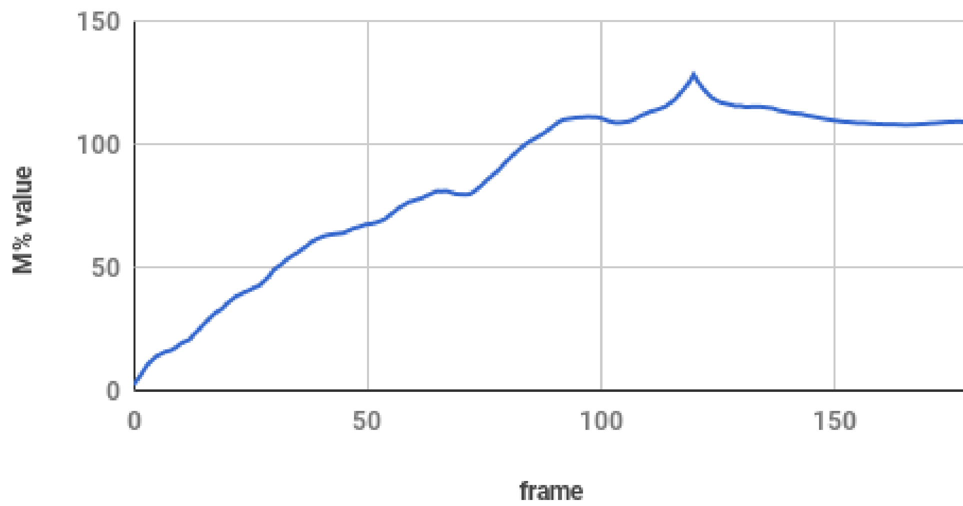


Figura G.1: Grafic valori M%: obiect 0074, format 16_16

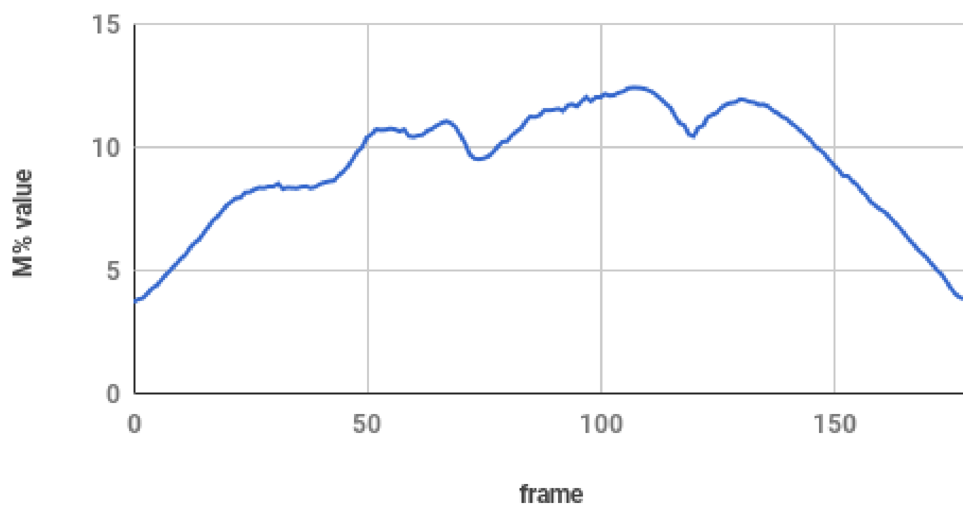


Figura G.2: Grafic valori M%: obiect 0074, format 32_32

BUPT

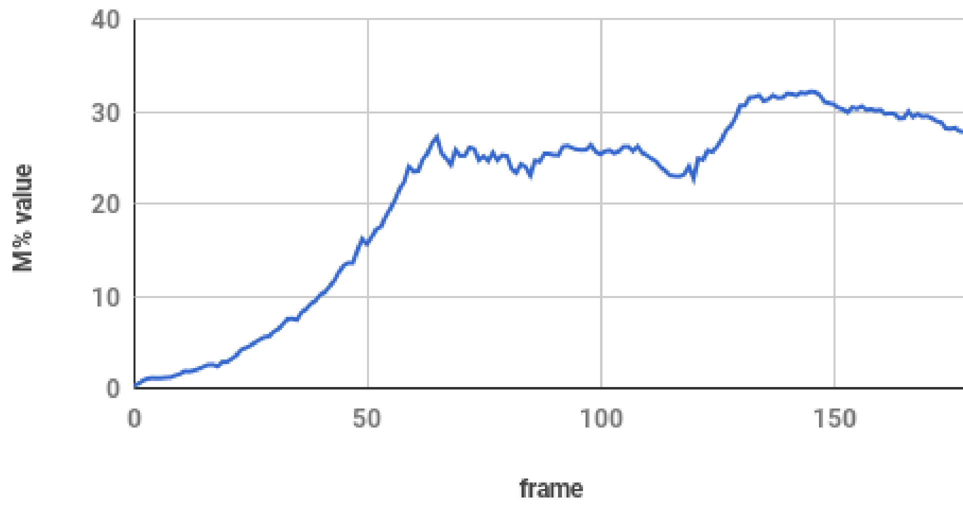


Figura G.3: Grafic valori M%: obiect 0081, format 16_16

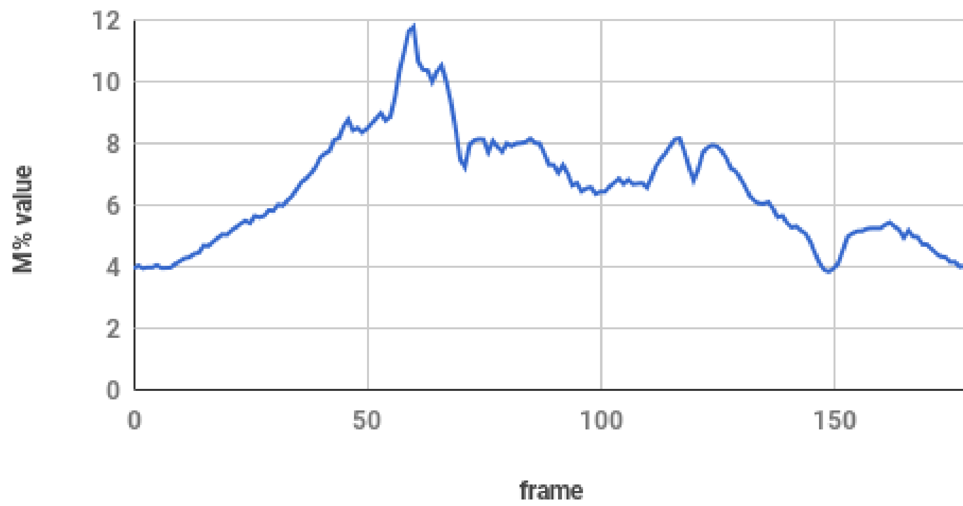


Figura G.4: Grafic valori M%: obiect 0081, format 32_32

BUPT

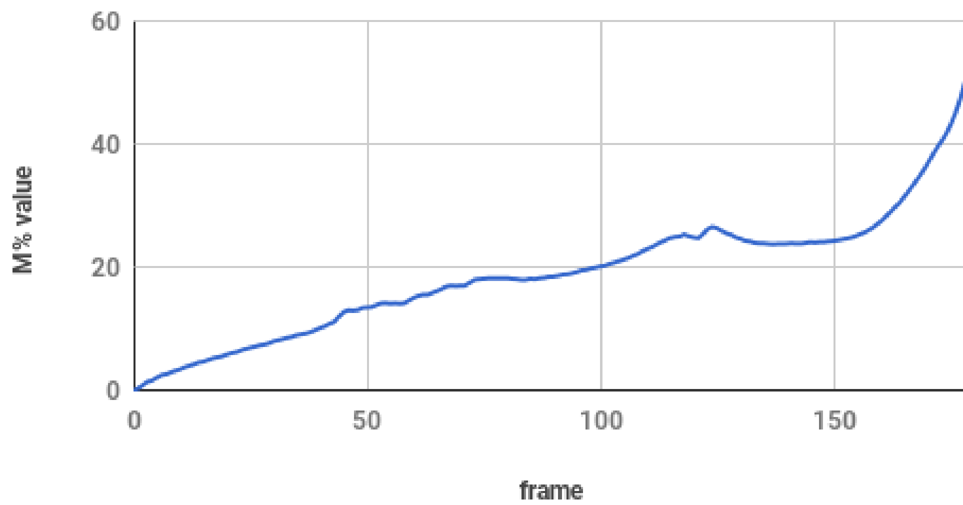


Figura G.5: Grafic valori M%: obiect 0115, format 16_16

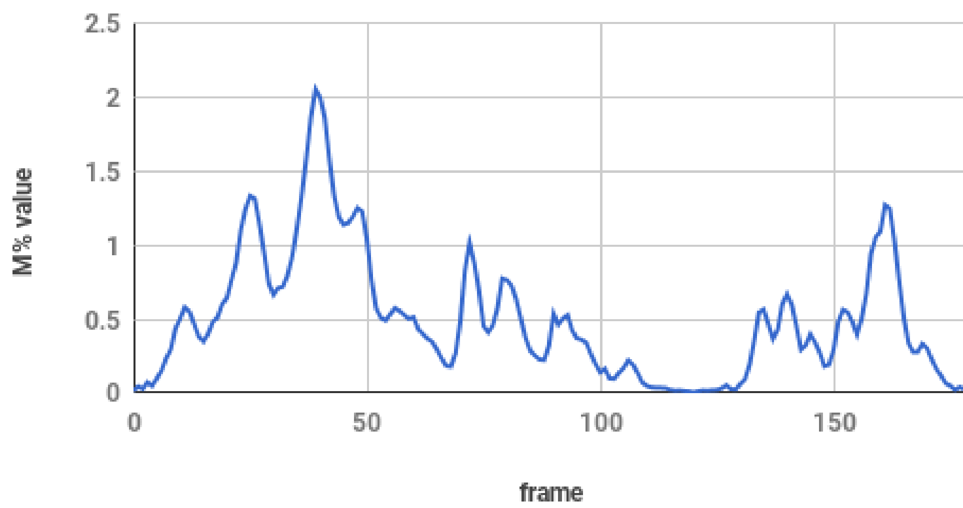


Figura G.6: Grafic valori M%: obiect 0115, format 32_32

BUPT

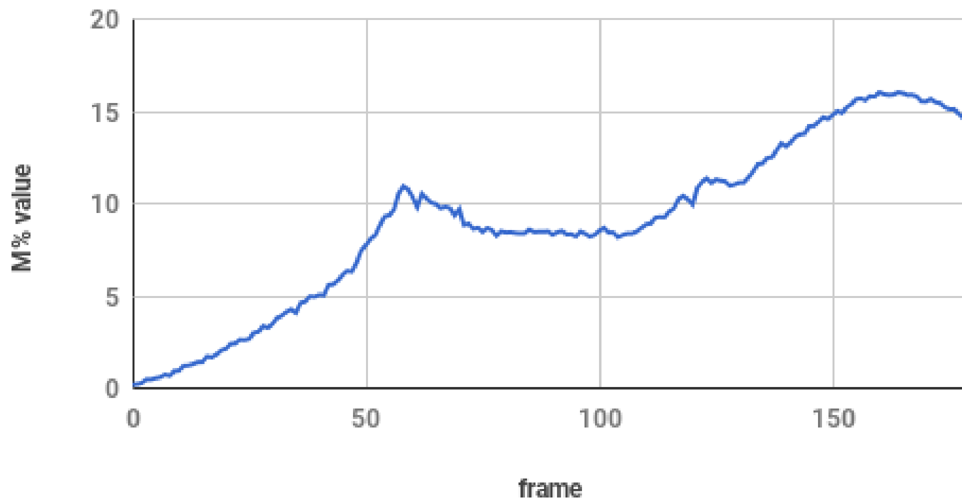


Figura G.7: Grafic valori M%: obiect 0538, format 16_16

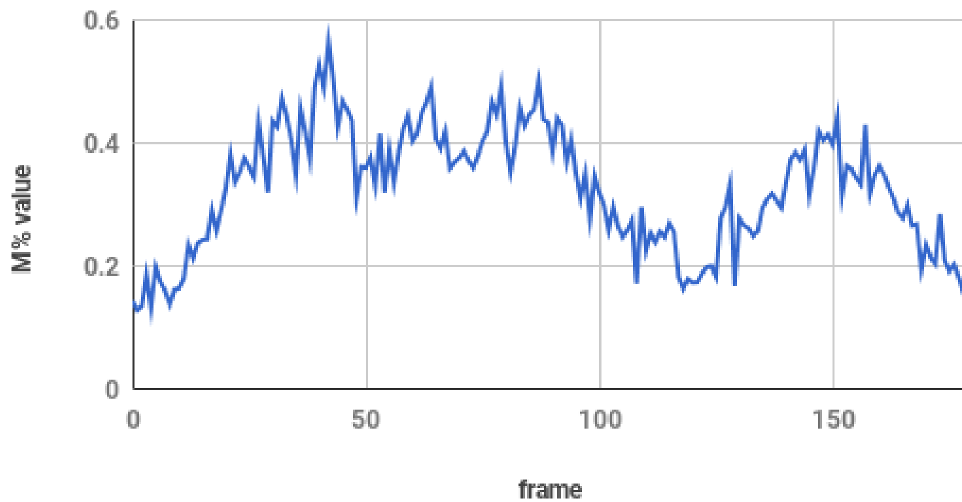


Figura G.8: Grafic valori M%: obiect 0538, format 32_32

BUPT

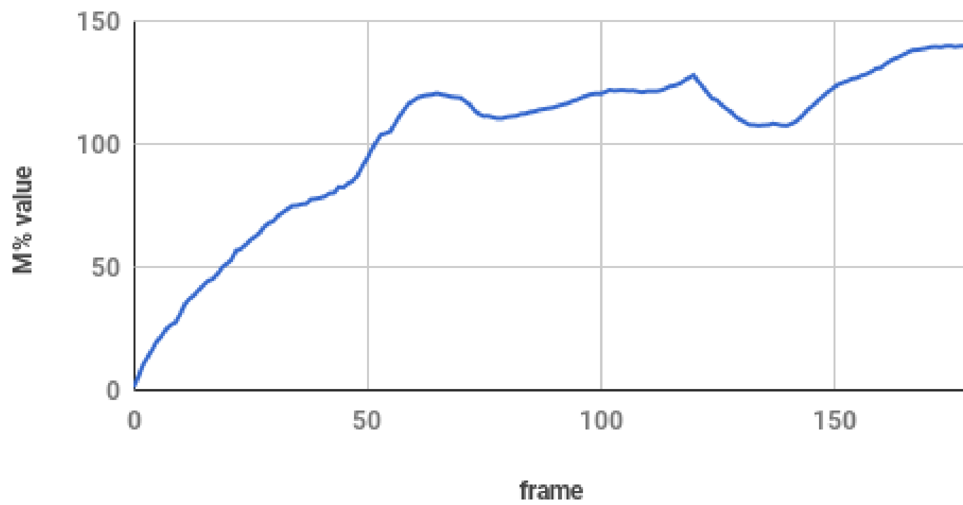


Figura G.9: Grafic valori M%: obiect 1886, format 16_16

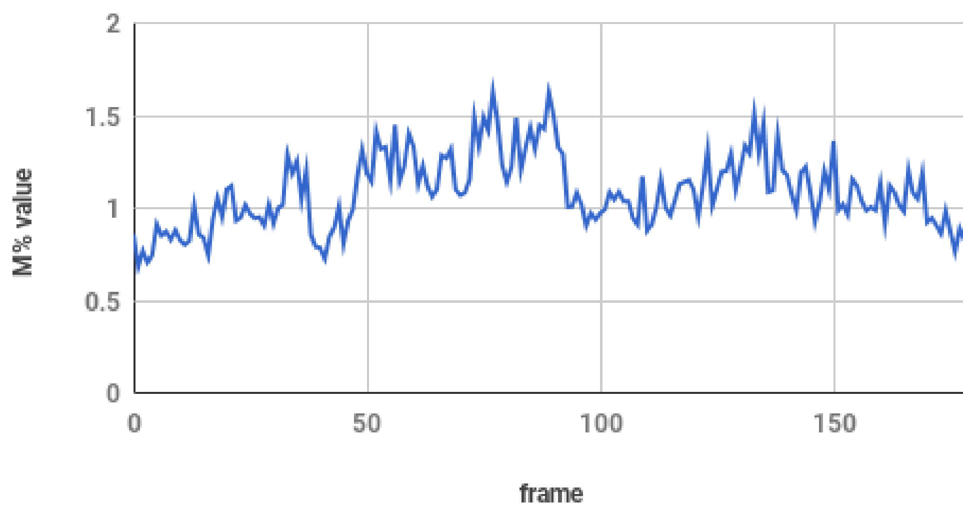


Figura G.10: Grafic valori M%: obiect 1886, format 32_32

BUPT

Anexa H

Grafice valori medii metrici

În această anexă sunt prezentate grafice cu valori medii per cadru. Fiecare astfel de grafic reprezintă valorile obținute pentru una dintre metrici, în funcție de combinația de precizie folosită, aplicată asupra imaginilor alterate, relativ la cele de referință.

Pentru a spori claritatea graficelor mediilor metricii M% prezentate în această anexă, am ales să nu figurăm valorile de la cadrele 60 și 120, unde valorile medii ale acestei metrici sunt mult diferite de cadrele alăturate.

BUPT

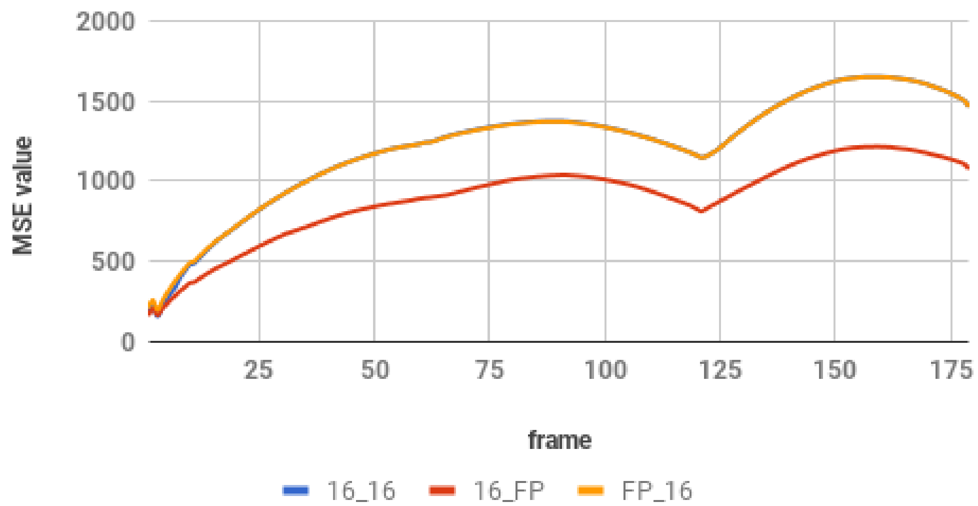


Figura H.1: Grafic valori medii MSE, formate 16

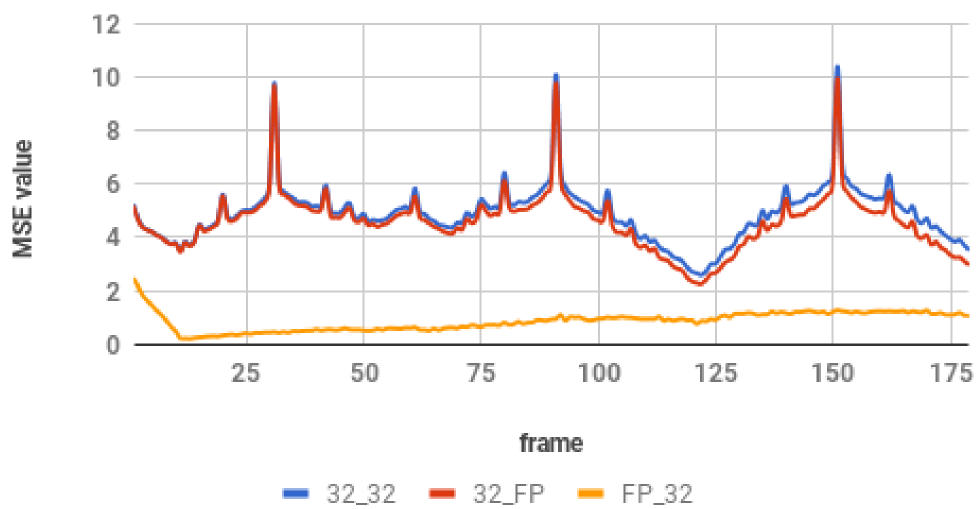


Figura H.2: Grafic valori medii MSE, formate 32

BUPT

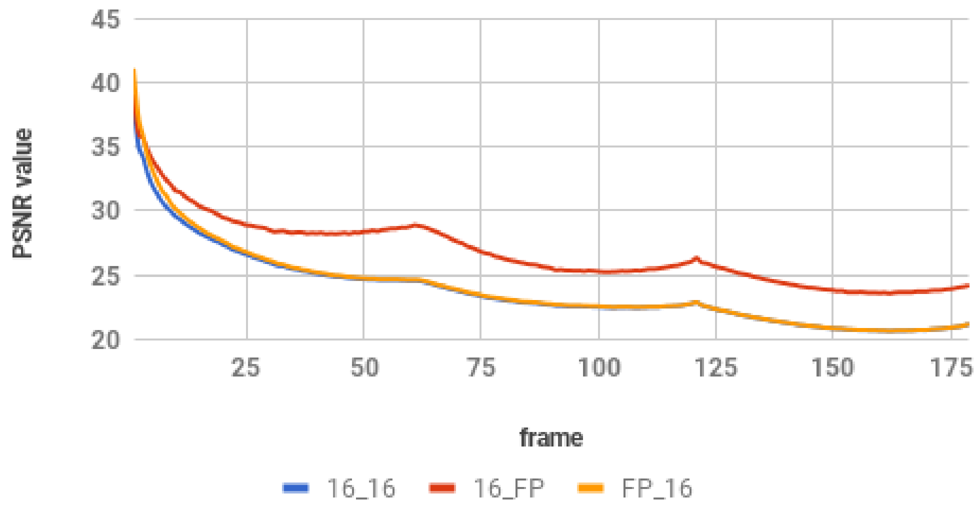


Figura H.3: Grafic valori medii PSNR, formate 16

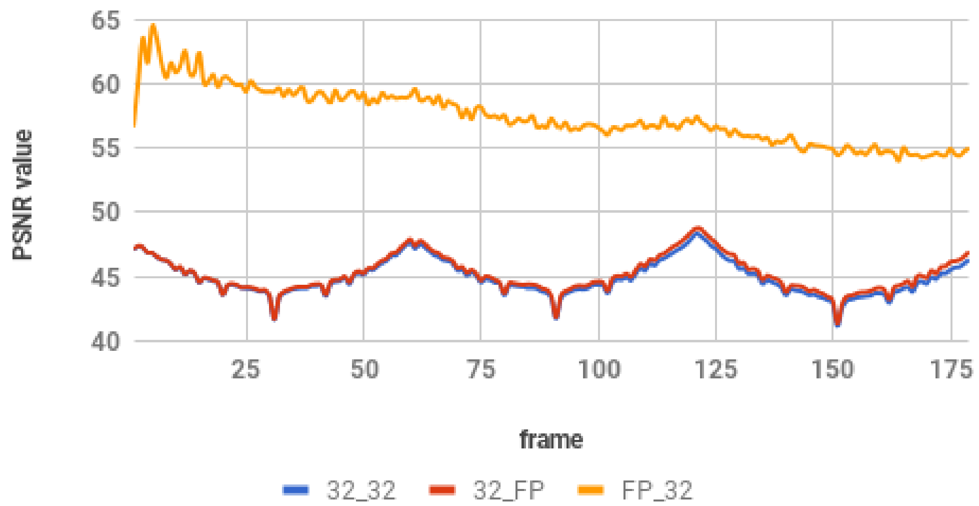


Figura H.4: Grafic valori medii PSNR, formate 32

BUPT

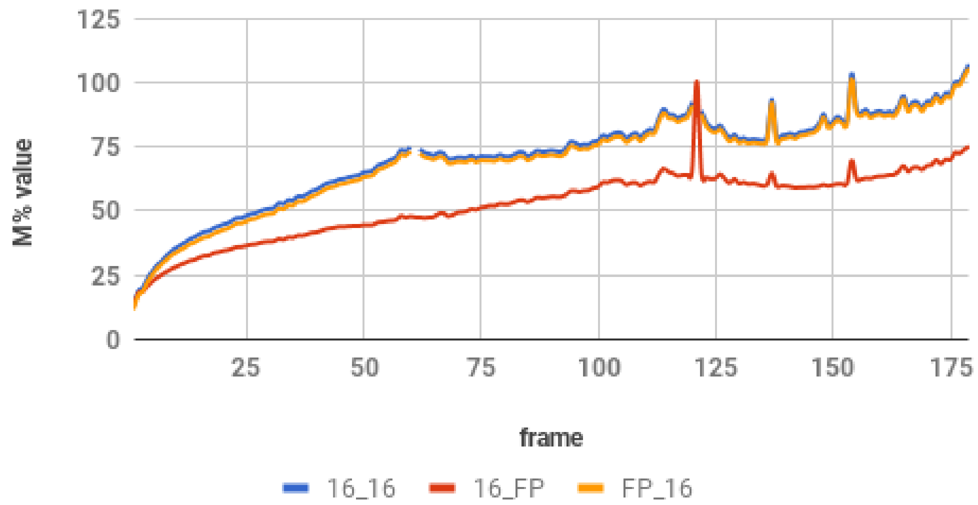


Figura H.5: Grafic valori medii M%, formate 16

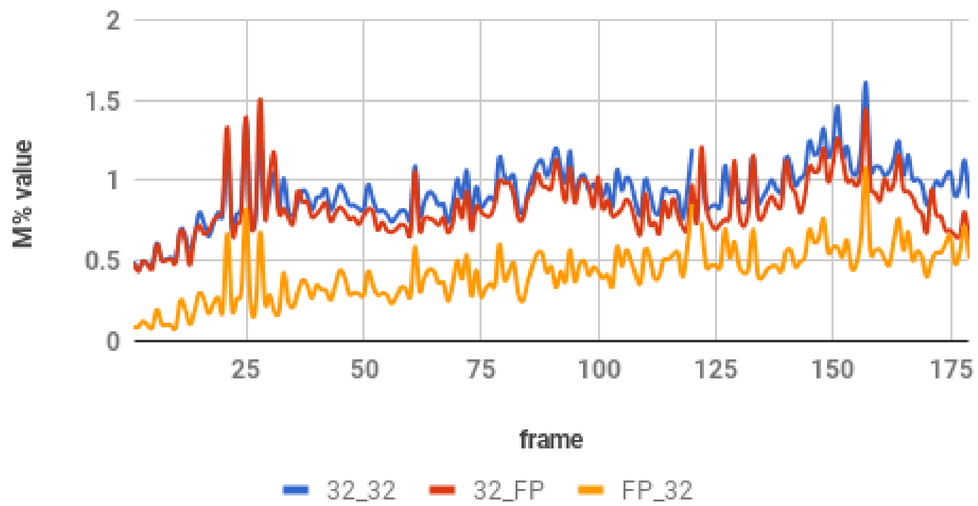


Figura H.6: Grafic valori medii M%, formate 32

BUPT

Anexa I

Grafice valori medii M% pentru obiecte

În această anexă sunt prezentate grafice cu valori medii per obiect ale metricii M%, pentru toată biblioteca de obiecte folosite. Fiecare astfel de grafic reprezintă valorile obținute pentru metrica M% asupra imaginilor alterate, relativ la cele de referință folosind combinația specificată de format de reprezentare a numerelor reale.

Pentru a spori claritatea graficelor prezentate în această anexă, un număr de aproximativ 20 de valori extreme (aproximativ 1%) au fost omise de la reprezentare, pentru fiecare grafic.

BUPT

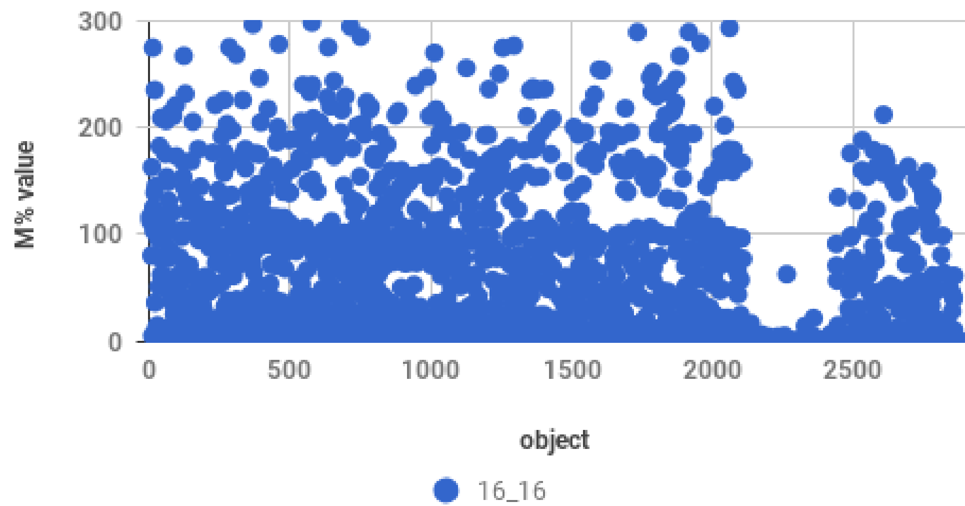


Figura I.1: Grafic valori medii M%: format 16_16

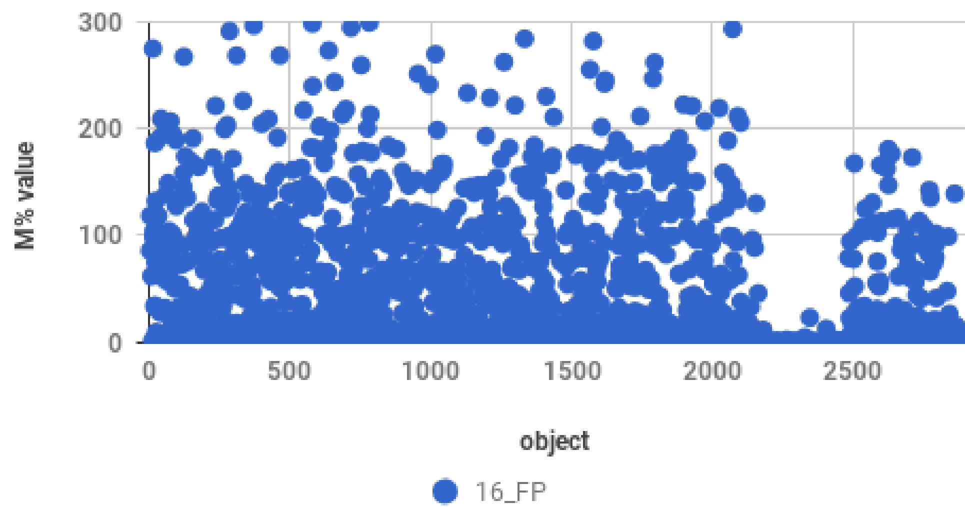


Figura I.2: Grafic valori medii M%: format 16_FP

BUPT

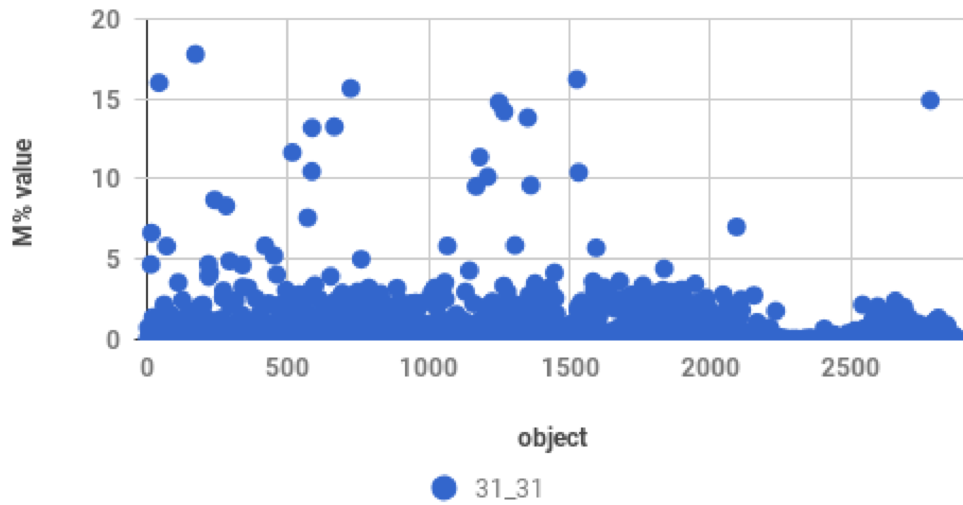


Figura I.3: Grafic valori medii M%: format 32_32

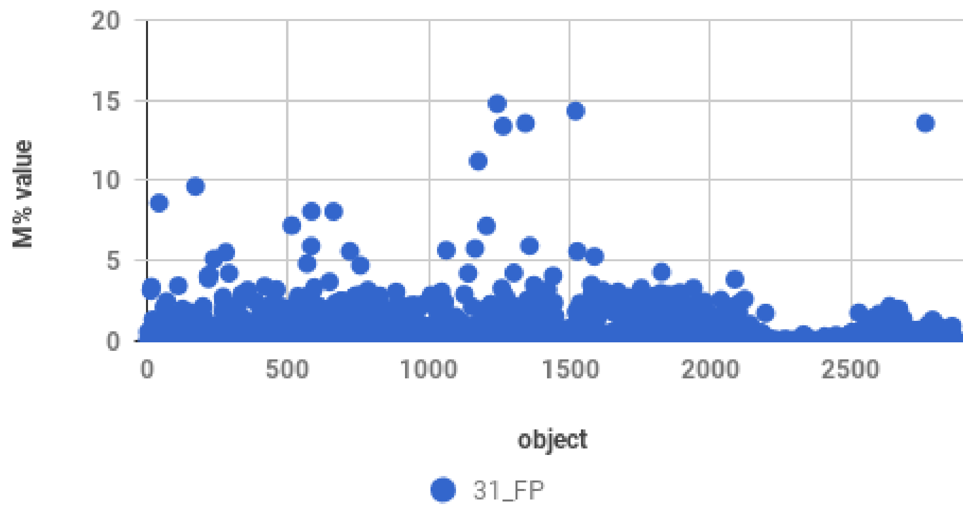


Figura I.4: Grafic valori medii M%: format 32_FP

BUPT

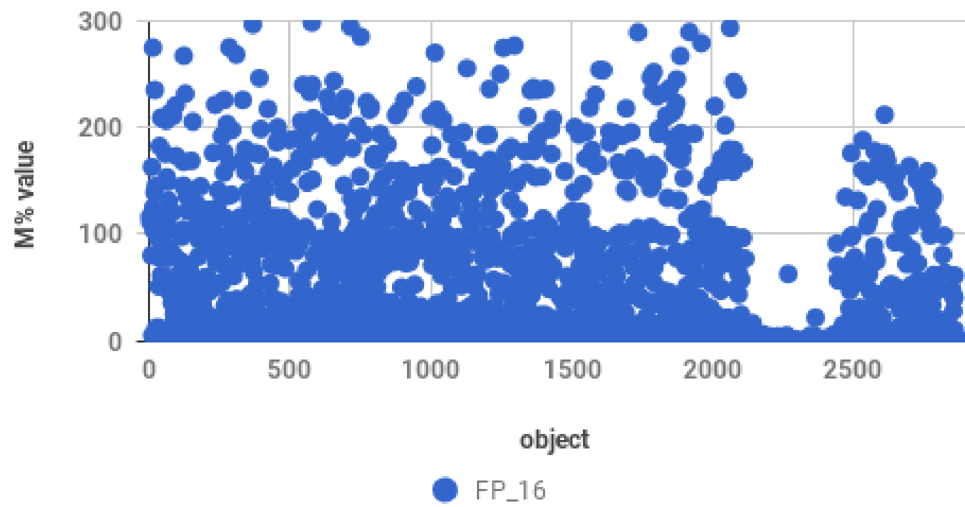


Figura I.5: Grafic valori medii M%: format FP_16

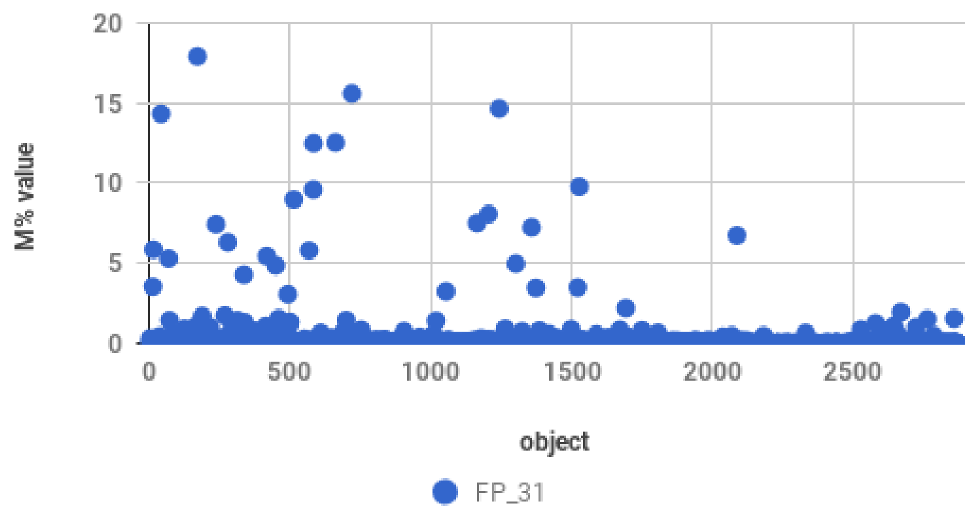


Figura I.6: Grafic valori medii M%: format FP_32

BUPT

Anexa J

Listă de publicații

J.1 Lucrări indexate ISI publicate în România

1. **Ovidiu Sicoe**, Alexandru Amaricăi, Mircea Popa, ”Generation of floating point 2D translation operators for FPGA,” 2015 IEEE 10th Jubilee International Symposium on Applied Computational Intelligence and Informatics, Timisoara, 2015, pp. 289-294. doi: 10.1109/SACI.2015.7208215

2. **Ovidiu Sicoe**, Mircea Popa, ”Generation of floating point 2D scaling operators for FPGA,” 2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, 2016, pp. 131-136. doi: 10.1109/SACI.2016.7507354

J.2 Lucrări indexate ISI publicate în străinătate

1. **Ovidiu Sicoe**, Mircea Popa. “A Mixed Fixed Point and Floating Point Graphics Pipeline,” In Proceedings of the 1st International Conference on Applied Physics, System Science and Computers (APSAC2016), September 28-30, Dubrovnik, Croatia, 2016, pp. 146-151. doi: 10.1007/978-3-319-53934-8_17

2. **Ovidiu Sicoe**, Mircea Popa. “Comparison metrics for the output of a mixed fixed point and floating point graphics pipeline,” In Proceedings of Jubilee 25th Telecommunications forum TELFOR 2017, Belgrade, Serbia, 2017, . . .

BUPT

J.3 Altele

1. Alexandru Amaricăi, Oana Boncalo, **Ovidiu Sicoe**. "FPGA implementations of low precision floating point multiply-accumulate," In International Journal of Microelectronics and Computer Science, Vol. 4, nr 4, 2013, pp. 159-163.

2. Alexandru Amaricăi, **Ovidiu Sicoe**, Oana Boncalo. "On the Redundant Representation of Partial Remainders in Radix-4 SRT Dividers," In Journal of Circuits, Systems and Computers, Vol. 26, issue 6, June 2017, pp. 1-13. doi: 10.1142/S0218126617500979

BUPT

Listă de figuri

3.1 Rețea de poligoane [71] 32
3.2 Transformări de Coordonate [72] 33
3.3 Suprafață descrisă print-o rețea de poligoane [71] 33
3.4 Rețea de triunghiuri [72] 34
3.5 Translația într-un spațiu bidimensional 36
3.6 Scalarea într-un spațiu bidimensional 38

4.1 Operator de translație cu șase unități de procesare [75] 43
4.2 Operator de translație cu două unități de procesare [75] 45
4.3 Operator de translație cu o unitate de procesare [75] 46
4.4 Operator de scalare cu șase înmulțitoare [76] 48
4.5 Operator de scalare cu două înmulțitoare [76] 50
4.6 Operator de scalare cu un înmulțitor [76] 51

5.1 Stagiile pipeline-ului descris de OpenGL ES 1.1 [72] 55
5.2 Arhitectura mediului de rulare 56
5.3 Exemplu de obiect 3D reprezentat printr-un fișier .obj 58
5.4 Conversie patrulater-triunghiuri 60
5.5 Schema logică a aplicației principale 62
5.6 Suprafețe multiple 63
5.7 Set de imagini comparate 64
5.8 Arhitectura modulului de analiză a datelor 66
5.9 Puncte de schimbare a formatului în pipeline-ul grafic 67
5.10 Diagramă de clase numere reale 68
5.11 Diagramă de secvență a celor două puncte de schimbare a formatului 69
5.12 Schema logică de rulare a modulelor software create 71

6.1 Histogramă distribuție vârfuri 74
6.2 Set de imagini cu diferență mare 76
6.3 Set de imagini cu diferență mică 77

BUPT

A.1 Histogramă distribuție triunghiuri 91
A.2 Histogramă distribuție dimensiune volum 91
A.3 Histogramă distribuție raport x/y 92
A.4 Histogramă distribuție raport y/z 92
A.5 Histogramă distribuție raport z/x 93

B.1 Suită de cadre de referință(FP_FP) 95
B.2 Suită de cadre alterate(16_16) 96
B.3 Suită de cadre diferență(FP_FP vs 16_16) 97

C.1 Suită de cadre de referință(FP_FP) 99
C.2 Suită de cadre alterate(format 32_32) 100
C.3 Suită de cadre diferență(FP_FP vs 32_32) 101

D.1 Imagine de referință 103
D.2 Imagine alterată 104
D.3 Imagine diferență 105

E.1 Imagine de referință 107
E.2 Imagine alterată 108
E.3 Imagine diferență 109

F.1 Grafic valori MSE: obiect 0074, format 16_16 111
F.2 Grafic valori PSNR: obiect 0074, format 16_16 111
F.3 Grafic valori MSE: obiect 0074, format 32_32 112
F.4 Grafic valori PSNR: obiect 0074, format 32_32 112
F.5 Grafic valori MSE: obiect 0081, format 16_16 113
F.6 Grafic valori PSNR: obiect 0081, format 16_16 113
F.7 Grafic valori MSE: obiect 0081, format 32_32 114
F.8 Grafic valori PSNR: obiect 0081, format 32_32 114
F.9 Grafic valori MSE: obiect 0115, format 16_16 115
F.10 Grafic valori PSNR: obiect 0115, format 16_16 115
F.11 Grafic valori MSE: obiect 0115, format 32_32 116
F.12 Grafic valori PSNR: obiect 0115, format 32_32 116
F.13 Grafic valori MSE: obiect 0538, format 16_16 117
F.14 Grafic valori PSNR: obiect 0538, format 16_16 117
F.15 Grafic valori MSE: obiect 0538, format 32_32 118
F.16 Grafic valori PSNR: obiect 0538, format 32_32 118
F.17 Grafic valori MSE: obiect 1886, format 16_16 119
F.18 Grafic valori PSNR: obiect 1886, format 16_16 119
F.19 Grafic valori MSE: obiect 1886, format 32_32 120
F.20 Grafic valori PSNR: obiect 1886, format 32_32 120

BUPT

G.1 Grafic valori M%: obiect 0074, format 16_16 122
G.2 Grafic valori M%: obiect 0074, format 32_32 122
G.3 Grafic valori M%: obiect 0081, format 16_16 123
G.4 Grafic valori M%: obiect 0081, format 32_32 123
G.5 Grafic valori M%: obiect 0115, format 16_16 124
G.6 Grafic valori M%: obiect 0115, format 32_32 124
G.7 Grafic valori M%: obiect 0538, format 16_16 125
G.8 Grafic valori M%: obiect 0538, format 32_32 125
G.9 Grafic valori M%: obiect 1886, format 16_16 126
G.10Grafic valori M%: obiect 1886, format 32_32 126

H.1 Grafic valori medii MSE, formate 16 128
H.2 Grafic valori medii MSE, formate 32 128
H.3 Grafic valori medii PSNR, formate 16 129
H.4 Grafic valori medii PSNR, formate 32 129
H.5 Grafic valori medii M%, formate 16 130
H.6 Grafic valori medii M%, formate 32 130

I.1 Grafic valori medii M%: format 16_16 132
I.2 Grafic valori medii M%: format 16_FP 132
I.3 Grafic valori medii M%: format 32_32 133
I.4 Grafic valori medii M%: format 32_FP 133
I.5 Grafic valori medii M%: format FP_16 134
I.6 Grafic valori medii M%: format FP_32 134

BUPT

Listă de tabele

4.1 Rezultate sinteză operator translație [75] 47
4.2 Rezultate sinteză operator scalare [76] 52

5.1 Formate folosite 70

6.1 Caracteristici obiecte tridimensionale 74
6.2 Valori MSE și PSNR pentru B.3 79
6.3 Valori MSE și PSNR pentru C.3 79
6.4 Valori M% pentru B.3 81
6.5 Valori M% pentru C.3 82
6.6 Valori extreme pentru metrici 83

BUPT

Bibliografie

[1] J. Nickolls and W. J. Dally, “The GPU computing era,” *IEEE Micro*, vol. 30, no. 2, pp. 56–69, 2010.

[2] J. D. Bruguera and T. Lang, “Floating-point fused multiply-add: Reduced latency for floating-point addition,” in *17th IEEE Symposium on Computer Arithmetic (ARITH-17 2005), 27-29 June 2005, Cape Cod, MA, USA*, pp. 42–51, 2005.

[3] S. Kestur, J. D. Davis, and E. S. Chung, “Towards a universal FPGA matrix-vector multiplication architecture,” in *2012 IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2012, 29 April - 1 May 2012, Toronto, Ontario, Canada*, pp. 9–16, IEEE, 2012.

[4] M. K. Jaiswal and R. C. C. Cheung, “Area-efficient architectures for double precision multiplier on fpga, with run-time-reconfigurable dual single precision support,” *Microelectronics Journal*, vol. 44, no. 5, pp. 421–430, 2013.

[5] H. Parizi, A. Niktash, A. H. Kamalizad, and N. Bagherzadeh, “A reconfigurable architecture for wireless communication systems,” in *Third International Conference on Information Technology: New Generations (ITNG 2006), 10-12 April 2006, Las Vegas, Nevada, USA*, pp. 250–255, IEEE Computer Society, 2006.

[6] F. Bensaali, A. Amira, and R. Sotudeh, “Floating-point matrix product on FPGA,” in *2007 IEEE/ACS International Conference on Computer Systems and Applications (AICCSA 2007), 13-16 May 2007, Amman, Jordan*, pp. 466–473, 2007.

[7] Y. Chang, J. Wei, W. Guo, and J. Sun, “A high performance, area efficient tta-like vertex shader architecture with optimized floating point arithmetic unit for embedded graphics applications,” *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 37, no. 6-7, pp. 725–738, 2013.

BUPT

[8] Y. Dou, S. Vassiliadis, G. Kuzmanov, and G. Gaydadjiev, “64-bit floating-point FPGA matrix multiplication,” in *Proceedings of the ACM/SIGDA 13th International Symposium on Field Programmable Gate Arrays, FPGA 2005, Monterey, California, USA, February 20-22, 2005*, pp. 86–95, 2005.

[9] B. Holanda, R. Pimentel, J. Barbosa, R. Camarotti, A. Silva-Filho, L. João, V. Souza, J. Ferraz, and M. Lima, “An fpga-based accelerator to speed-up matrix multiplication of floating point operations,” in *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May 2011 - Workshop Proceedings*, pp. 306–309, 2011.

[10] Z. Jovanovic and V. M. Milutinovic, “Fpga accelerator for floating-point matrix multiplication,” *IET Computers & Digital Techniques*, vol. 6, no. 4, pp. 249–256, 2012.

[11] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. S. Hemmert, “Embedded floating-point units in fpgas,” in *Proceedings of the ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, FPGA 2006, Monterey, California, USA, February 22-24, 2006* (S. J. E. Wilton and A. DeHon, eds.), pp. 12–20, ACM, 2006.

[12] D. M. Russinoff, “Computation and Formal Verification of SRT Quotient and Square Root Digit Selection Tables,” *IEEE Trans. Computers*, vol. 62, no. 5, pp. 900–913, 2013.

[13] P. Montuschi and L. Ciminiera, “Over redundant digit sets and the design of digit-by-digit division units,” *IEEE Trans. Computers*, vol. 43, no. 3, pp. 269–277, 1994.

[14] P. Kornerup, “Digit selection for srt division and square root,” *IEEE Trans. Computers*, vol. 54, no. 3, pp. 294–303, 2005.

[15] A. A. Ibrahem, H. Elsimary, and A. E. Salama, “Fpga implementation of fast radix 4 division algorithm,” in *IWSOC*, pp. 69–72, IEEE Computer Society, 2004.

[16] M. D. Ercegovac and T. Lang, “On-the-fly conversion of redundant into conventional representations,” *IEEE Trans. Computers*, vol. 36, no. 7, pp. 895–897, 1987.

[17] T. N. Pham and E. E. S. Jr., “Design of radix-4 srt dividers in 65 nanometer cmos technology,” in *ASAP*, pp. 105–108, IEEE Computer Society, 2006.

[18] N. Burgess and C. N. Hinds, “Design of the ARM VFP11 divide and square root synthesisable macrocell,” in *IEEE Symposium on Computer Arithmetic*, pp. 87–96, IEEE Computer Society, 2007.

BUPT

[19] M. Anane, H. Bessalah, M. Issad, N. Anane, and H. Salhi, “Higher radix and redundancy factor for floating point SRT division,” *IEEE Trans. VLSI Syst.*, vol. 16, no. 6, pp. 774–779, 2008.

[20] A. Nannarelli, “Radix-16 combined division and square root unit,” in Antelo *et al.* [80], pp. 169–176.

[21] H. R. Srinivas, K. K. Parhi, and L. A. Montalvo, “Radix 2 division with over-redundant quotient selection,” *IEEE Trans. Computers*, vol. 46, no. 1, pp. 85–92, 1997.

[22] H. R. Srinivas and K. K. Parhi, “A fast radix-4 division algorithm and its architecture,” *IEEE Trans. Computers*, vol. 44, no. 6, pp. 826–831, 1995.

[23] F. de Dinechin, J. Detrey, O. Cret, and R. Tudoran, “When fpgas are better at floating-point than microprocessors,” in *Proceedings of the ACM/SIGDA 16th International Symposium on Field Programmable Gate Arrays, FPGA 2008, Monterey, California, USA, February 24-26, 2008*, p. 260, 2008.

[24] Z. Guo, W. A. Najjar, F. Vahid, and K. A. Vissers, “A quantitative analysis of the speedup factors of fpgas over processors,” in *Proceedings of the ACM/-SIGDA 12th International Symposium on Field Programmable Gate Arrays, FPGA 2004, Monterey, California, USA, February 22-24, 2004*, pp. 162–170, 2004.

[25] N. Kapre and A. DeHon, “Optimistic parallelization of floating-point accumulation,” in *18th IEEE Symposium on Computer Arithmetic (ARITH-18 2007), 25-27 June 2007, Montpellier, France*, pp. 205–216, 2007.

[26] J. Detrey and F. de Dinechin, “A parameterized floating-point exponential function for fpgas,” in *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology, FPT 2005, 11-14 December 2005, Singagore* (G. J. Brebner, S. Chakraborty, and W. Wong, eds.), pp. 27–34, IEEE, 2005.

[27] J. Detrey and F. de Dinechin, “Parameterized floating-point logarithm and exponential functions for fpgas,” *Microprocessors and Microsystems*, vol. 31, no. 8, pp. 537–545, 2007.

[28] F. de Dinechin and B. Pasca, “Floating-point exponential functions for dsp-enabled fpgas,” in *Proceedings of the International Conference on Field-Programmable Technology, FPT 2010, 8-10 December 2010, Tsinghua University, Beijing, China* (J. Bian, Q. Zhou, P. Athanas, Y. Ha, and K. Zhao, eds.), pp. 110–117, IEEE, 2010.

BUPT

[29] J. Detrey and F. de Dinechin, “Floating-point trigonometric functions for fpgas,” in *FPL 2007, International Conference on Field Programmable Logic and Applications, Amsterdam, The Netherlands, 27-29 August 2007* (K. Bertels, W. A. Najjar, A. J. van Genderen, and S. Vassiliadis, eds.), pp. 29–34, IEEE, 2007.

[30] F. de Dinechin, M. Istoan, and G. Sergent, “Fixed-point trigonometric functions on fpgas,” *SIGARCH Computer Architecture News*, vol. 41, no. 5, pp. 83–88, 2013.

[31] F. de Dinechin, B. Pasca, O. Cret, and R. Tudoran, “An fpga-specific approach to floating-point accumulation and sum-of-products,” in *2008 International Conference on Field-Programmable Technology, FPT 2008, Taipei, Taiwan, December 7-10, 2008* (T. A. El-Ghazawi, Y. Chang, J. Huang, and P. Saha, eds.), pp. 33–40, IEEE, 2008.

[32] F. de Dinechin and B. Pasca, “Designing custom arithmetic data paths with flopoco,” *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.

[33] F. de Dinechin and B. Pasca, “Designing custom arithmetic data paths with flopoco,” *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.

[34] F. de Dinechin, “The arithmetic operators you will never see in a microprocessor,” in Antelo *et al.* [80], pp. 189–190.

[35] N. Brisebarre, F. de Dinechin, and J. Muller, “Integer and floating-point constant multipliers for fpgas,” in *19th IEEE International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2008, July 2-4, 2008, Leuven, Belgium*, pp. 239–244, IEEE Computer Society, 2008.

[36] V. Lefèvre, “Multiplication by an Integer Constant,” Research Report RR-4192, INRIA, 2001.

[37] M. G. Arnold and S. Collange, “A dual-purpose real/complex logarithmic number system ALU,” in *19th IEEE Symposium on Computer Arithmetic, ARITH 2009, Portland, Oregon, USA, 9-10 June 2009* (J. D. Bruguera, M. Cornea, D. D. Sarma, and J. Harrison, eds.), pp. 15–24, IEEE Computer Society, 2009.

[38] F. de Dinechin and B. Pasca, “Large multipliers with fewer DSP blocks,” in Danek *et al.* [81], pp. 250–255.

[39] S. Banescu, F. de Dinechin, B. Pasca, and R. Tudoran, “Multipliers for floating-point double precision and beyond on fpgas,” *SIGARCH Computer Architecture News*, vol. 38, no. 4, pp. 73–79, 2010.

BUPT

[40] F. de Dinechin, C. Klein, and B. Pasca, “Generating high-performance custom floating-point pipelines,” in Danek *et al.* [81], pp. 59–64.

[41] F. de Dinechin, H. D. Nguyen, and B. Pasca, “Pipelined FPGA adders,” in *International Conference on Field Programmable Logic and Applications, FPL 2010, August 31 2010 - September 2, 2010, Milano, Italy* [82], pp. 422–427.

[42] F. de Dinechin, M. Joldes, B. Pasca, and G. Revy, “Multiplicative square root algorithms for fpgas,” in *International Conference on Field Programmable Logic and Applications, FPL 2010, August 31 2010 - September 2, 2010, Milano, Italy* [82], pp. 574–577.

[43] F. de Dinechin, M. Joldes, and B. Pasca, “Automatic generation of polynomial-based hardware architectures for function evaluation,” in *21st IEEE International Conference on Application-specific Systems Architectures and Processors, ASAP 2010, Rennes, France, 7-9 July 2010*, pp. 216–222, IEEE, 2010.

[44] J. Pool, A. Lastra, and M. Singh, “Power-gated arithmetic circuits for energy-precision tradeoffs in mobile graphics processing units,” *J. Low Power Electronics*, vol. 7, no. 2, pp. 148–162, 2011.

[45] R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” *IEEE Trans. Computers*, vol. 31, no. 3, pp. 260–264, 1982.

[46] V. M. D. Barrio, C. González, J. Roca, A. Fernández, and R. Espasa, “ATTILA: a cycle-level execution-driven simulator for modern GPU architectures,” in *2006 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2006, March 19-21, 2006, Austin, Texas, USA, Proceedings*, pp. 231–241, IEEE Computer Society, 2006.

[47] U. I. Minhas, S. Bayliss, and G. A. Constantinides, “GPU vs FPGA: A comparative analysis for non-standard precision,” in *Reconfigurable Computing: Architectures, Tools, and Applications - 10th International Symposium, ARC 2014, Vilamoura, Portugal, April 14-16, 2014. Proceedings* (D. Goehringer, M. D. Santambrogio, J. M. P. Cardoso, and K. Bertels, eds.), vol. 8405 of *Lecture Notes in Computer Science*, pp. 298–305, Springer, 2014.

[48] N. Corporation, “Board Specification Tesla C1060 Computing Processor Board.” `http://www.nvidia.com/docs/IO/43395/BD-04111-001_v05.pdf`.

[49] Xilinx, “Virtex-6 Family Overview.” `https://www.xilinx.com/support/documentation/data_sheets/ds150.pdf`.

BUPT

[50] Xilinx, "LogiCORE IP Floating-PointOperator v5.0." https://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf.

[51] P. K. Krause, "ftc - floating precision texture compression," *Computers & Graphics*, vol. 34, no. 5, pp. 594–601, 2010.

[52] X. Hao and A. Varshney, "Variable-precision rendering," in *Proceedings of the 2001 Symposium on Interactive 3D Graphics, SI3D 2001, Chapel Hill, NC, USA, March 26-29, 2001* (J. F. Hughes and C. H. Séquin, eds.), pp. 149–158, ACM, 2001.

[53] D. Kanter, "A Look Inside Apple's Custom GPU for the iPhone." http://www.realworldtech.com/apple-custom-gpu/.

[54] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *The IEEE/ACM International Conference on Computer-Aided Design, ICCAD'13, San Jose, CA, USA, November 18-21, 2013* (J. Henkel, ed.), pp. 48–54, IEEE, 2013.

[55] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate xor/xnor-based adders for inexact computing," in *2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013)*, pp. 690–693, Aug 2013.

[56] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012* (P. Groeneveld, D. Sciuto, and S. Hassoun, eds.), pp. 820–825, ACM, 2012.

[57] Standard Performance Evaluation Corporation, "Standard Performance Evaluation Corporation (SPEC) CPU2006." http://www.spec.org/cpu2006.

[58] D. Shin and S. K. Gupta, "A re-design technique for datapath modules in error tolerant applications," in *17th IEEE Asian Test Symposium, ATS 2008, Sapporo, Japan, November 24-27, 2008*, pp. 431–437, IEEE Computer Society, 2008.

[59] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE Trans. VLSI Syst.*, vol. 18, no. 8, pp. 1225–1229, 2010.

[60] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Trans. Computers*, vol. 62, no. 9, pp. 1760–1771, 2013.

BUPT

[61] E. Schkufza, R. Sharma, and A. Aiken, "Stochastic optimization of floating-point programs with tunable precision," in *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014* (M. F. P. O'Boyle and K. Pingali, eds.), pp. 53–64, ACM, 2014.

[62] E. Schkufza, R. Sharma, and A. Aiken, "Stochastic superoptimization," in *Architectural Support for Programming Languages and Operating Systems, ASPLOS '13, Houston, TX, USA - March 16 - 20, 2013* (V. Sarkar and R. Bodík, eds.), pp. 305–316, ACM, 2013.

[63] V. Camus, J. Schlachter, C. Enz, M. Gautschi, and F. K. Gurkaynak, "Approximate 32-bit floating-point unit design with 53% power-area product reduction," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference, Lausanne, Switzerland, September 12-15, 2016*, pp. 465–468, IEEE, 2016.

[64] B. Grigorian and G. Reinman, "Improving coverage and reliability in approximate computing using application-specific, light-weight checks," in *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014.

[65] D. Tsai and R. Yang, "An eigenvalue-based similarity measure and its application in defect detection," *Image Vision Comput.*, vol. 23, no. 12, pp. 1094–1101, 2005.

[66] E. A. Silva, K. Panetta, and S. S. Agaian, "Quantifying image similarity using measure of enhancement by entropy," *Mobile Multimedia/Image Processing for Military and Security Applications*, vol. 6579, p. 65790U, 2007.

[67] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[68] A. Horé and D. Ziou, "Image quality metrics: PSNR vs. SSIM," in *20th International Conference on Pattern Recognition, ICPR 2010, Istanbul, Turkey, 23-26 August 2010*, pp. 2366–2369, IEEE Computer Society, 2010.

[69] L. Zhang, L. Zhang, X. Mou, and D. Zhang, "FSIM: A feature similarity index for image quality assessment," *IEEE Trans. Image Processing*, vol. 20, no. 8, pp. 2378–2386, 2011.

[70] W. Lin and C. J. Kuo, "Perceptual visual quality metrics: A survey," *J. Visual Communication and Image Representation*, vol. 22, no. 4, pp. 297–312, 2011.

BUPT

[71] A. H. Watt, *3D Computer Graphics with Cdrom.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 3rd ed., 1999.

[72] T. K. G. Inc, "Opengl ES Common/Common-Lite Profile Specification." https://www.khronos.org/registry/gles/#specs11.

[73] J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, and K. Akeley, *Computer Graphics: Principles and Practice (3rd Edition).* Addison-Wesley, 2013.

[74] P. W. Markstein, "The new IEEE-754 standard for floating point arithmetic," in *Numerical Validation in Current Hardware Architectures, 6.1. - 11.1.2008* (A. A. M. Cuyt, W. Krämer, W. Luther, and P. W. Markstein, eds.), vol. 08021 of *Dagstuhl Seminar Proceedings*, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[75] O. Sicoe, A. Amaricai, and M. Popa, "Generation of floating point 2d translation operators for FPGA," in *10th IEEE Jubilee International Symposium on Applied Computational Intelligence and Informatics, SACI 2015, Timisoara, Romania, May 21-23, 2015*, pp. 289–294, IEEE, 2015.

[76] O. Sicoe and M. Popa, "Generation of floating point 2d scaling operators for FPGA," in *11th IEEE International Symposium on Applied Computational Intelligence and Informatics, SACI 2016, Timisoara, Romania, May 12-14, 2016*, pp. 131–136, IEEE, 2016.

[77] O. Sicoe, "Towards Implementation of Graphic Libraries On Reconfigurable Platforms," Master's thesis, Politehnica University of Timisoara, Timisoara, Romania, 2013.

[78] J. B. G. R.-P. Guy Eric Schalnat, Andreas Dilger *et al.*, "libpng." http://www.libpng.org/pub/png/libpng.html.

[79] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[80] E. Antelo, D. Hough, and P. Ienne, eds., *20th IEEE Symposium on Computer Arithmetic, ARITH 2011, Tübingen, Germany, 25-27 July 2011*, IEEE Computer Society, 2011.

[81] M. Danek, J. Kadlec, and B. E. Nelson, eds., *19th International Conference on Field Programmable Logic and Applications, FPL 2009, August 31 - September 2, 2009, Prague, Czech Republic*, IEEE, 2009.

BUPT

[82] *International Conference on Field Programmable Logic and Applications, FPL 2010, August 31 2010 - September 2, 2010, Milano, Italy*, IEEE, 2010.

BUPT