Ανάπτυξη Μεθόδων για Αποδοτική Πολυθραυσματική Απόδοση Σκηνών

Κωνσταντίνος Παπακώστας

Μεταπτυχιακή Εργασία Εξειδίκευσης

— ♦ —

Ιωάννινα, Ιούλιος 2022

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

ΑΝΑΠΤΥΞΗ ΜΕΘΟΔΩΝ ΓΙΑ ΑΠΟΔΟΤΙΚΗ ΠΟΛΥΘΡΑΥΣΜΑΤΙΚΗ ΑΠΟΔΟΣΗ

Η Μεταπτυχιακή Διπλωματική Εργασία

υποβάλλεται στην ορισθείσα

από τη Συνέλευση

του Τμήματος Μηχανικών Η/Υ και Πληροφορικής

Εξεταστική Επιτροπή

από τον

Κωνσταντίνο Παπακώστα

ως μέρος των υποχρεώσεων για την απόκτηση του

ΔΙΠΛΩΜΑΤΟΣ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΣΤΗ ΜΗΧΑΝΙΚΗ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΜΕ ΕΙΔΙΚΕΥΣΗ

ΣΤΑ ΠΡΟΗΓΜΕΝΑ ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ

Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Ιωάννινα 2022

Εξεταστική Επιτροπή:

- **Ιωάννης Φούντος**, Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων (Επιβλέπων)
- **Βασίλειος Δημακόπουλος**, Αναπλ. Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
- **Βασίλειος Τενέντες**, Επικ. Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων

Ευχαριστιες

Θα ήθελα να ευχαριστήσω τον κ.Φούντο για την υπομονή του και τις γνώσεις που προσέφερε πάνω στο αντικείμενο, για να καταλήξουμε σε ένα αξιόλογο αποτέλεσμα. Τέλος, ένα ευχαριστώ στους κυρίους Γρηγόρη Τσοπουρίδη και Ανδρέα Βασιλάκη που βοήθησαν μέχρι το τέλος για την επίλυση προβλημάτων που προέκυψαν.

Περιεχομενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κατάλογος Αλγορίθμων

Περιληψη

Κωνσταντίνος Παπακώστας, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2022.
ΑΝΑΠΤΥΞΗ ΜΕΘΟΔΩΝ ΓΙΑ ΑΠΟΔΟΤΙΚΗ ΠΟΛΥΘΡΑΥΣΜΑΤΙΚΗ ΑΠΟΔΟΣΗ.
Επιβλέπων: Ιωάννης Φούντος, Καθηγητής.

Στόχος της συγκεκριμένης εργασίας είναι η κατασκευή αποδοτικών μεθόδων για πολυθραυσματική απόδοση. Το αντικείμενο της εργασίας είναι η απόδοση 3D σκηνής με πολλαπλά αντικείμενα τα οποία μπορούν να έχουν μερική ή πλήρη διαφάνεια. Το τελικό αποτέλεσμα είναι μια εικόνα που αποδίδει τη σκηνή από μια συγκεκριμένη οπτική γωνία. Ονομάζεται πολυθραυσματική γιατί το χρώμα κάθε εικονοστοιχείου της τελικής εικόνας προέρχεται όχι μόνο από το κοντινότερο προς την κάμερα θραύσμα (θραύσμα είναι το τμήμα ενός αντικειμένου που προβάλλεται νοητά στο αντίστοιχο εικονοστοιχείο) αλλά από όλα τα θραύσματα που αφορούν το εικονοστοιχείο αυτό. Με χρήση του σύγχρονου υλικού για γραφικά μπορεί να συλλεχθεί η πληροφορία για όλα τα θραύσματα, να ταξινομηθούν σύμφωνα με το βάθος στη σκηνή και μετά να υπολογιστεί το ιδανικό αποτέλεσμα. Η μέθοδος αυτή ονομάζεται OIT (Order Independent Transparency) είναι αργή, έχει μεγάλες απαιτήσεις σε μνήμη και δεν ενδείκνυται για γραφικά πραγματικού χρόνου. Μια προσέγγιση της μεθόδου αυτής είναι Weighted Blended Transparency όπου το κάθε θραύσμα συνεισφέρει στο χρώμα του εικονοστοιχείου με ένα βάρος που είναι συνάρτηση του βάθους και του συντελεστή διαφάνειας χωρίς να γίνει ταξινόμηση των θραυσμάτων. Στην εργασία αυτή έχουμε μελετήσει την προσθήκη ενός νευρωνικού δικτύου αντί της συνάρτησης βάρους που έχει εκπαιδευθεί ώστε να προσεγγίζει τον συντελεστή με τον οποίο το θραύσμα συμμετέχει στην βέλτιστη μέθοδο (OIT).

Η υλοποίηση των αλγορίθμων έγινε με χρήση WebGL και Javascript και το νευρωνικό υλοποιήθηκε σε python με χρήση του tensorflow.

Extended Abstract

Konstantinos Papakostas, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2022.
DEVELOPMENT OF METHODS FOR EFFICIENT MULTI-FRAGMENT RENDERING.
Advisor: Ioannis Fudos, Professor.

The goal of this work is to construct efficient methods for multi-fragment rendering. The scope of the work is to render a 3D scene with multiple objects which can be partially or fully transparent. The final result is an image that renders the scene from a specific viewpoint. It is called multi-fragment because the color of each pixel of the final image is derived not only from the closest fragment to the camera (a fragment is the part of an object that is mentally projected onto the corresponding pixel) but from all the fragments that relate to that pixel. Using modern graphics hardware, information about all fragments can be collected, sorted according to depth in the scene, and then the ideal result can be calculated. This method called OIT (Order Independent Transparency) is slow, has large memory requirements and is not suitable for real-time graphics. An approximation of this method is Weighted Blended Transparency where each fragment contributes to the pixel color with a weight that is a function of depth and transparency factor without sorting the fragments. In this work we have studied the addition of a neural network instead of the weight function trained to approximate the factor at which the fragment participates in the optimal method (OIT).
The implementation of the algorithms was done using WebGL and Javascript and the neural was implemented in python using the tensorflow libraries.

Κεφαλαιο 1

Εισαγωγη

1.1 Ορισμός του προβλήματος

Το πρόβλημα που πραγματεύεται η εργασία την αναπαράσταση 3D σκηνής με επικαλυπτόμενα πολύγωνα αποδίδοντας ρεαλιστική διαφάνεια και συγχρόνως αποδοτική με βάση το βάθος των αντικειμένων στην σκηνή. Οι μέθοδοι αυτοί αφορούν κυρίως εκδόσεις των shader σε επίπεδο web όπου οι δυνατότητες του hardware είναι περιορισμένες. Πιο συγκεκριμένα η εργασία συσχετίζει και αντικείμενα του τομέα των νευρωνικών δικτύων για την εύρεση συναρτήσεων για την προσέγγιση της τέλειας διαφάνειας.

1.2 Στόχοι

Στόχος της συγκεκριμένης εργασίας είναι η κατασκευή αποδοτικών μεθόδων για πολυθραυσματική απόδοση. Το αντικείμενο της εργασίας είναι η απόδοση 3D σκηνής με πολλαπλά αντικείμενα τα οποία μπορούν να έχουν μερική ή πλήρη διαφάνεια. Το

τελικό αποτέλεσμα είναι μια εικόνα που αποδίδει τη σκηνή από μια συγκεκριμένη οπτική γωνία.

Ειδικότερα η προσέγγιση αλγορίθμων όπως το Depth Peeling με αλγορίθμους ταχύτερους βασιζόμενοι σε συναρτήσεις βάρους προσεγγίζοντας την διαφάνεια των θραυσμάτων (fragments).Η διαδικασία βασίζεται στην εξαγωγή των θραυσμάτων της Depth Peeling μεθόδου.

Επιπλέον, η μέθοδος που βασιστήκαμε για την βελτίωση του αποτελέσματος ονομάζεται Weighted Blended Transparency, η οποία είναι μία προσεγγιστική μέθοδος υπολογισμού της διαφάνειας των θραυσμάτων συνυπολογίζοντας την διαφάνεια των γειτονικών τους θραυσμάτων.

1.3 Σχετικές Εργασίες

Άλλες σχετικές εργασίες είναι ο A-Buffer [2] που είναι μια τεχνική γραφικών υπολογιστών που εισήχθη το 1984, η οποία αποθηκεύει ανά pixel λίστες δεδομένων θραυσμάτων (συμπεριλαμβανομένων πληροφοριών μικροπολυγώνων) σε ένα λογισμικό ραστεροποίησης, το REYES (Render Everything You Ever Saw), αρχικά σχεδιασμένο για anti-aliasing αλλά υποστηρίζοντας επίσης τη διαφάνεια. Το 2001 παρουσιάστηκε μια τεχνική OIT επιταχυνόμενη από το υλικό, η οποία χρησιμοποιεί τον buffer βάθους για να αποκολλά ένα στρώμα εικονοστοιχείων σε κάθε πέρασμα. Με τους περιορισμούς στο υλικό γραφικών η γεωμετρία της σκηνής έπρεπε να αποδοθεί πολλές φορές, όπου το 2008 παρουσιάστηκε το dual depth deeling [1] βελτιώνοντας την απόδοση του depth peeling, εξακολουθώντας να έχει τον περιορισμό της απόδοσης πολλών περασμάτων. Τέλος το 2013 δημοσιεύθηκε η τεχινκή Weighted-Blended OIT [7] και χρησιμοποιεί μια συνάρτηση βάρους και δύο buffers για το χρώμα των εικονοστοιχείων και το κατώφλι αποκάλυψης των εικονοστοιχείων για το τελικό πέρασμα της σύνθεσης. Έχει ως αποτέλεσμα μια προσεγγιστική εικόνα με αξιοπρεπή ποιότητα σε πολύπλοκες σκηνές.

1.4 Δομή της Διατριβής

Η διατριβή περιέχει τα εξής κεφάλαια:

- Στο Κεφάλαιο 2 αναφέρουμε τις έννοιες, τις μεθόδους και τις ορολογίες του αντικειμένου.
- Στο Κεφάλαιο 3 παρουσιάζουμε τις μεθόδους για πολυθραυσματική απόδοση.
- Στο Κεφάλαιο 4 παρουσιάζονται τα αποτελέσματα μαζί και με την πειραματική αξιολόγηση.
- Τέλος στο Κεφάλαιο 5 παρουσιάζουμε τα συμπεράσματα και μελλοντικές κατευθύνσεις της εργασίας.

Κεφαλαιο 2

Υποβαθρο

2.1 Εισαγωγή στους Όρους της Μεταπτυχιακής Εργασίας

Στην ενότητα αυτή θα παρουσιάσουμε τις βασικές έννοιες που χρειάζονται για την κατανόηση του προβλήματος που παρουσιάζεται στην παρούσα μεταπτυχιακή εργασία. Πιο συγκεκριμένα θα ασχοληθούμε με μεθόδους σε shaders για παραγωγή ρεαλιστικού αποτελέσματος και συγχρόνως αποδοτικούς. Στη συνέχεια παρουσιάζονται οι όροι των γραφοσκιαστών (**shaders**), των θραυσμάτων (**fragments**) και τέλος την έννοια της ανεξάρτητης από την σειρά διαφάνεια **order-independent transparency**.

Στα γραφικά υπολογιστών, ο γραφοσκιαστής είναι ένα πρόγραμμα που υπολογίζει τα κατάλληλα επίπεδα φωτός, σκότους και χρώματος κατά την απόδοση μιας τρισδιάστατης σκηνής - μια διαδικασία γνωστή ως σκίαση. Οι γραφοσκιαστές έχουν εξελιχθεί για να εκτελούν μια ποικιλία εξειδικευμένων λειτουργιών στα ειδικά εφέ γραφικών υπολογιστών και στην επεξεργασία βίντεο, καθώς και σε υπολογισμούς γενικής χρήσης σε μονάδες επεξεργασίας γραφικών. Οι παραδοσιακοί γραφοσκιαστές υπολογίζουν τα εφέ απόδοσης στο υλικό γραφικών με υψηλό βαθμό ευελιξίας. Οι περισσότεροι γραφοσκιαστές είναι προγραμματισμένοι για μια μονάδα επεξεργασίας γραφικών (GPU), αν και αυτό δεν αποτελεί αυστηρή απαίτηση. Οι γλώσσες

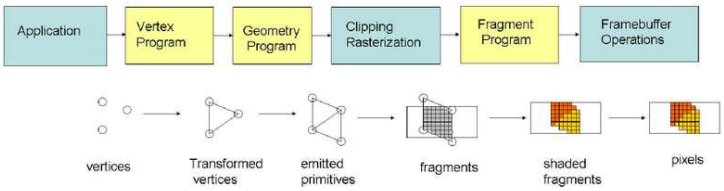
σκίασης (shader language) χρησιμοποιούνται για τον προγραμματισμό της διαδικασίας απόδοσης (rendering pipeline) της GPU. Η θέση και το χρώμα όλων των εικονοστοιχείων, των κορυφών ή των υφών που χρησιμοποιούνται για την κατασκευή μιας τελικής απεικονιζόμενης εικόνας μπορούν να μεταβληθούν χρησιμοποιώντας αλγορίθμους που ορίζονται σε έναν γραφοσκιαστή και μπορούν να τροποποιηθούν από εξωτερικές μεταβλητές ή υφές που εισάγονται από το πρόγραμμα υπολογιστή που καλεί τον γραφοσκιαστή.

Υπάρχουν αρκετά είδη γραφοσκιαστών με τα πιο διαδεδομένα να είναι οι Pixel Shader (Fragment Shader), Vertex Shader και Tessellation Shader. Στην παρούσα εργασία θα χρειαστούμε τους Vertex και Fragment Shaders.

Οι Fragment Shader, υπολογίζουν το χρώμα και άλλα χαρακτηριστικά κάθε θραύσματος. Τα απλούστερα είδη fragment shader εξάγουν ένα εικονοστοιχείο της οθόνης ως τιμή χρώματος, είναι επίσης πιο σύνθετοι γραφοσκιαστές με πολλαπλές εισόδους και εξόδους. Μπορούν να μεταβάλλουν το βάθος του θραύσματος (για Z-buffering), ή να εξάγουν περισσότερα από ένα χρώματα εάν υπάρχουν πολλαπλές επιφάνειες απόδοσης. Στα τρισδιάστατα γραφικά, ένας fragment shader από μόνος του δεν μπορεί να παράγει ορισμένα είδη σύνθετων εφέ, επειδή λειτουργεί σε ένα μόνο θραύσμα, χωρίς να γνωρίζει τη γεωμετρία της σκηνής (δηλαδή τα δεδομένα κορυφών). Ωστόσο, οι fragment shader έχουν γνώση των συντεταγμένων της οθόνης που σχεδιάζεται και μπορούν να πάρουν δείγμα της οθόνης και των κοντινών εικονοστοιχείων, εάν τα περιεχόμενα ολόκληρης της οθόνης περάσουν ως υφή στον σκιαστή. Αυτή η τεχνική μπορεί να επιτρέψει μια μεγάλη ποικιλία από δισδιάστατα εφέ μετα-επεξεργασίας, όπως θόλωση ή ανίχνευση/ενίσχυση ακμών για σκιαστές καρτούν. Οι σκιαστές εικονοστοιχείων μπορούν επίσης να εφαρμοστούν σε ενδιάμεσα στάδια σε οποιεσδήποτε δισδιάστατες εικόνες ή υφές στο pipeline, ενώ οι σκιαστές κορυφών απαιτούν πάντα μια τρισδιάστατη σκηνή. Για παράδειγμα, ένας shader εικονοστοιχείων είναι το μόνο είδος shader που μπορεί να λειτουργήσει ως μετα-επεξεργαστής ή φίλτρο για μια ροή βίντεο αφού αυτή έχει ραστεροποιηθεί.

Οι vertex shader είναι το πιο καθιερωμένο και κοινό είδος σκιαστών 3D και εκτελούνται μία φορά για κάθε κορυφή που δίνεται στον επεξεργαστή γραφικών. Ο σκοπός είναι να μετασχηματίσουν την τρισδιάστατη θέση κάθε κορυφής στον εικονικό χώρο στη συντεταγμένη 2D στην οποία εμφανίζεται στην οθόνη (καθώς και μια τιμή βάθους για τον Z-buffer. Οι vertex shader μπορούν να χειριστούν ιδιότητες όπως η θέση, το χρώμα και οι συντεταγμένες υφής, αλλά δεν μπορούν να δημιουρ-

γήσουν νέες κορυφές. Η έξοδος του vertex shader πηγαίνει στο επόμενο στάδιο του pipeline, το οποίο είναι είτε ένας geometry shader αν υπάρχει, είτε ο rasterizer.



Σχήμα 2.1: Διαδικασία απόδοσης τρισδιάστατης εικόνας.

Στα γραφικά υπολογιστών, ένα θραύσμα είναι τα δεδομένα που είναι απαραίτητα για τη δημιουργία ενός εικονοστοιχείου ενός αρχικού σχεδίου στον buffer του frame.

Τα δεδομένα αυτά μπορεί να περιλαμβάνουν, μεταξύ άλλων, τα εξής:

- Θέση του ράστερ.
- Βάθος.
- Παρεμβαλλόμενα χαρακτηριστικά (χρώμα, συντεταγμένες υφής κ.λ.π.).
- Στένσιλ.
- Άλφα.
- ID παραθύρου.

Καθώς σχεδιάζεται μια σκηνή, τα primitives σχεδίασης ραστεροποιούνται σε θραύσματα τα οποία υφίστανται υφή και συνδυάζονται με τον υπάρχοντα buffer καρέ. Ο τρόπος με τον οποίο ένα θραύσμα συνδυάζεται με τα δεδομένα που βρίσκονται ήδη στον frame buffer εξαρτάται από διάφορες ρυθμίσεις. Σε μια τυπική περίπτωση, ένα θραύσμα μπορεί να απορριφθεί εάν είναι πιο μακριά από το pixel που βρίσκεται ήδη σε αυτή τη θέση. Εάν είναι πιο κοντά από το υπάρχον pixel, μπορεί να αντικαταστήσει αυτό που ήδη υπάρχει, ή, εάν χρησιμοποιείται η μείξη άλφα, το χρώμα του pixel μπορεί να αντικατασταθεί με ένα μείγμα του χρώματος του θραύσματος και του υπάρχοντος χρώματος του pixel, όπως στην περίπτωση σχεδίασης ενός ημιδιαφανούς αντικειμένου.

Το order-independent transparency είναι ένα σύνολο από τεχνικές για την απόδοση διαφάνειας σε μία τρισδιάστατη σκηνή, που δεν απαιτεί απόδοση γεωμετρίας

σε ταξινομημένη σειρά για την σύνθεση του άλφα. Συνήθως, η τρισδιάστατη γεωμετρία με διαφάνεια αποδίδεται με ανάμειξη όλων των επιφανειών σε έναν buffer. Κάθε επιφάνεια αποκρύπτει το υπάρχον χρώμα και προσθέτει λίγο από το δικό της χρώμα ανάλογα με την τιμή άλφα, μια αναλογία της διαπερατότητας του φωτός. Η σειρά με την οποία αναμειγνύονται οι επιφάνειες επηρεάζει τη συνολική απόκρυψη ή ορατότητα κάθε επιφάνειας. Για ένα σωστό αποτέλεσμα, οι επιφάνειες πρέπει να αναμειγνύονται από την πιο απομακρυσμένη προς την πιο κοντινή ή από την πιο κοντινή προς την πιο απομακρυσμένη, ανάλογα με τη λειτουργία σύνθεσης άλφα. Η ταξινόμηση μπορεί να επιτευχθεί με την απόδοση της γεωμετρίας σε ταξινομημένη σειρά, για παράδειγμα με ταξινόμηση των τριγώνων κατά βάθος, αλλά μπορεί να πάρει σημαντικό χρόνο, να μην παράγει πάντα λύση και η υλοποίηση είναι πολύπλοκη. Αντ' αυτού, η ανεξάρτητη από τη σειρά διαφάνεια ταξινομεί τη γεωμετρία ανά pixel, μετά τη ραστεροποίηση. Για ακριβή αποτελέσματα αυτό απαιτεί την αποθήκευση όλων των θραυσμάτων πριν από την ταξινόμηση και τη σύνθεση.

2.2 Μέθοδοι που μελετήθηκαν

Για τους σκοπούς της εργασίας αυτής μελετήθηκαν αρκετές μέθοδοι για την απόδοση διαφάνειας, από τις οποίες υλοποιήθηκαν οι δύο σε WebGL (OpenGL ES 3.0) και javascript.

Αρχικά η πρώτη μέθοδος είναι η dual depth peeling [1], η βασική ιδέα είναι ότι για κάθε πέρασμα ξεφλουδίζουμε ένα στρώμα από μπροστά και ένα στρώμα από πίσω.

Διατηρούμε χειροκίνητα ένα buffer βάθους για την τιμή βάθους μπροστά και πίσω, ώστε για κάθε πέρασμα να γνωρίζουμε πόσο βαθιά είμαστε για την αποκόλληση. Απορρίπτουμε όλα τα θραύσματα που βρίσκονται εκτός του τρέχοντος εύρους βάθους. Σχεδιάζουμε όλα τα θραύσματα εντός αυτού του εύρους στον depth buffer και αφήνουμε το πλησιέστερο επόμενο, ώστε να ξέρουμε ποιο θραύσμα να σχεδιάσουμε για το επόμενο πέρασμα. Τέλος σχεδιάζουμε θραύσματα που έχουν την ίδια τιμή βάθους με το τρέχον μπροστινό ή πίσω βάθος στο μπροστινό και πίσω buffer χρώματος.

Στην συνέχεια μελετήσαμε την μέθοδο του A-Buffer [2], όπου η μέθοδος επεκτείνει τον αλγόριθμο depth-buffer (ή Z Buffer). Καθώς η μέθοδος depth buffer μπορεί

να χρησιμοποιηθεί μόνο για αδιαφανές αντικείμενο αλλά όχι για διαφανές αντικείμενο, η μέθοδος A-Buffer παρέχει πλεονέκτημα σε αυτό το σενάριο. Αν και η μέθοδος A-Buffer απαιτεί περισσότερη μνήμη, αλλά τα διαφορετικά χρώματα επιφάνειας μπορούν να συντίθενται σωστά με τη χρήση της. Η βασική δομή δεδομένων στον A-Buffer είναι ο buffer συσσώρευσης. Κάθε θέση στον A-Buffer έχει 2 πεδία, ένα πεδίο για το βάθος και ένα πεδίο για δεδομένα σχετικά με την επιφάνεια. Ένα πεδίο βάθους αποθηκεύει έναν θετικό ή αρνητικό πραγματικό αριθμό. Ένα πεδίο δεδομένων επιφάνειας μπορεί να αποθηκεύει πληροφορίες έντασης επιφάνειας ή έναν δείκτη σε μια συνδεδεμένη λίστα επιφανειών που συμβάλλουν στη συγκεκριμένη θέση εικονοστοιχείου. Εάν η τιμή του βάθους είναι μη αρνητική τότε ο αριθμός που αποθηκεύεται σε αυτή τη θέση είναι το βάθος της μίας μοναδικής επιφάνειας που επικαλύπτει την αντίστοιχη περιοχή εικονοστοιχείων, αντιθέτως η συνεισφορά πολλαπλών επιφανειών στην ένταση του εικονοστοιχείου υποδεικνύεται με αρνητική τιμή του πεδίου του βάθους.

Τέλος η μέθοδος που μελετήσαμε και υλοποιήσαμε είναι η Weighted Blended Order-Independent Transparency. Αυτή η μέθοδος αποδίδει μη διαθλαστική, μονόχρωμη μετάδοση μέσω επιφανειών που έχουν οι ίδιες χρώμα, χωρίς να απαιτείται διαλογή ή νέα χαρακτηριστικά υλικού. Στην πραγματικότητα, μπορεί να υλοποιηθεί με ένα απλό shader για οποιαδήποτε GPU που υποστηρίζει blending.

Η βασική ιδέα της μεθόδου Weighted Blended είναι ο ακριβής υπολογισμός της κάλυψης του φόντου από τις διαφανείς επιφάνειες. Ο αλγόριθμος επιβάλλει έναν ευρετικό συντελεστή μεταξύ των διαφανών επιφανειών που αυξάνεται με την απόσταση από την κάμερα.

Κεφαλαιο 3

Μεθοδοι για Πολυθραυσματικη αποδοση

Στο κεφάλαιο αυτό θα εμβαθύνουμε περισσότερο στις τεχνικές πολυθραυσματικής απόδοσης που μελετήσαμε και χρησιμοποιήσαμε.

3.1 Dual Depth Peeling

3.1.1 Εισαγωγή

Η ακριβής απόδοση ημιδιαφανών υλικών με μη ομοιόμορφα χρώματα RGBA απαιτεί σύνθεση θραυσμάτων σε σειρά βάθους. Το Depth Peeling είναι μία ισχυρή λύση με βάση την εικόνα σε αυτό το πρόβλημα, η οποία συλλαμβάνει ένα στρώμα θραυσμάτων κάθε φορά που η γεωμετρία αποδίδεται (geometry pass). Εκτός από την ανεξάρτητη από τη σειρά διαφάνεια, το Depth Peeling είναι χρήσιμη για τη δημιουργία πολυεπίπεδων εικόνων βάθους και την ανίχνευση δεσμίδων ακτίνων. Το σημαντικότερο μειονέκτημα του αρχικού αλγορίθμου Depth Peeling είναι ότι απαιτεί ένα πέρασμα γεωμετρίας ανά στρώμα. Έτσι, για να αποτυπωθούν όλα τα θραύσματα μιας σκηνής με πολυπλοκότητα βάθους N, το Depth Peeling απαιτεί N περάσματα

γεωμετρίας. Αυτό είναι απαγορευτικό για εφαρμογές που τρέχουνε στην GPU. Το Dual Depth Peeling μειώνει τον αριθμό των περασμάτων γεωμετρίας από N σε N/2+1 εφαρμόζοντας τη μέθοδο του Depth Peeling από μπροστά και από πίσω ταυτόχρονα. Το ένα επιπλέον πέρασμα προκύπτει από το γεγονός ότι ο αλγόριθμός μας πρέπει να εργαστεί σε δύο διαδοχικά στρώματα σε μια φορά. Εκτός από το Dual Depth Peeling, περιγράφουμε επίσης πώς τα αποφλοιωμένα θραύσματα μπορούν να αναμειχθούν on the fly με βάση την εξίσωση ανάμειξης άλφα [3].

Το Depth Peeling ξεκινάει με την κανονική απόδοση της σκηνής με ένα τεστ βάθους, το οποίο επιστρέφει τα πλησιέστερα θραύσματα στο μάτι. Σε ένα δεύτερο πέρασμα, ο depth buffer από το προηγούμενο πέρασμα δεσμεύεται σε έναν fragment shader. Για να αποφευχθούν οι κίνδυνοι ανάγνωσης-τροποποίησης-εγγραφής, η πηγή και ο depth buffer προορισμού ανταλλάσσονται σε κάθε πέρασμα (ping ponging) [4][6]. Εάν το βάθος θραύσματος είναι μικρότερο ή ίσο με το σχετικό βάθος από το προηγούμενο πέρασμα, τότε αυτό το θραύσμα απορρίπτεται και επιστρέφεται από τον έλεγχο βάθους το επόμενο στρώμα από κάτω.

Εννοιολογικά, το Dual Depth Peeling εκτελεί το Depth Peeling τόσο από πίσω προς τα εμπρός όσο και από μπροστά προς τα πίσω, αποκολλώντας ένα στρώμα από μπροστά και ένα στρώμα από πίσω σε κάθε πέρασμα. Αυτό θα ήταν εύκολο να υλοποιηθεί αν η GPU διέθετε πολλαπλούς depth buffers και αν κάθε buffer συσχετιζόταν με ένα συγκεκριμένο texture. Θα υπήρχε μια υφή RGB για ανάμειξη από πίσω προς τα εμπρός και μία RGBA υφή για ανάμειξη από μπροστά προς τα πίσω. Ωστόσο, στο περίπτωση του Σχήματος 3.2, το στρώμα στη μέση θα αποκολλήθηκε και θα αναμειγνύονταν τόσο στις υφές ανάμειξης από εμπρός προς τα πίσω όσο και στις υφές ανάμειξης από πίσω προς τα εμπρός. Επιλύουμε αυτό το ζήτημα δουλεύοντας με ένα παράθυρο δύο διαδοχικών στρωμάτων. Επιπλέον, αυτό καθιστά δυνατή την υλοποίηση μιας δοκιμής βάθους minmax με συνδυασμό ανάμειξης στον ίδιο σκιαστή (”on the fly blending”). Εμείς υλοποιούμε ένα min-max depth buffer με ένα texture RG32F, χρησιμοποιώντας min blending για να τη σύγκριση βάθους. Ουσιαστικά απενεργοποιούμε τον depth buffer του υλικού και χρησιμοποιούμε blending για να εκτελέσουμε το τμήμα ανάγνωσης-τροποποίησης-εγγραφής της προσαρμοσμένοy τεστ βάθους. Τα ξεφλουδισμένα θραύσματα από τα εμπρόσθια στρώματα πρέπει να περάσουν από την εξίσωση under-blending, και τα θραύσματα από τα πίσω στρώματα μέσω της εξίσωσης over-blending. Τέλος, οι δύο αναμεμειγμένες εικόνες για τις μπροστινές και τις πίσω κατευθύνσεις αναμειγνύονται μεταξύ τους

με τη χρήση under-blending. [1]

Σχήμα 3.1: Εφαρμογή του Dual Depth Peeling στο βάθος.

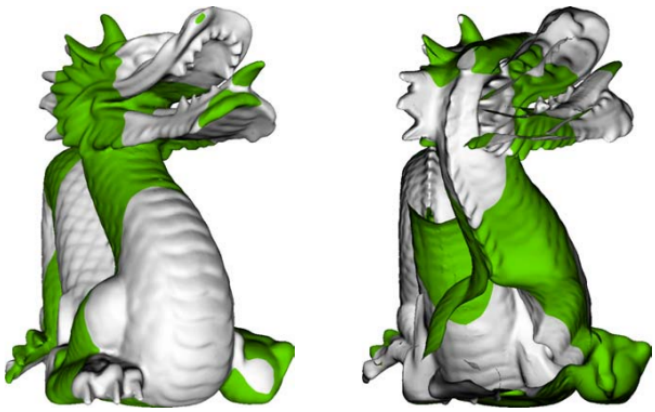
3.1.2 Λειτουργία του αλγορίθμου

Ο αλγόριθμος λειτουργεί σε σύνολα διαδοχικών στρωμάτων, δύο στρώματα από μπροστά και δύο στρώματα από το πίσω μέρος. Στο πρώτο πέρασμα, το εξωτερικό σύνολο αρχικοποιείται στα ελάχιστα και μέγιστα βάθη και κανένα χρώμα δεν αποκολλάται ακόμη. Στο δεύτερο πέρασμα, ο shader επεξεργάζεται τα θραύσματα που ταιριάζουν με τα βάθη του προηγούμενου εξωτερικού συνόλου (Σχήμα 3.1) και εκτελεί το Depth Peeling στα στρώματα εντός του εξωτερικού συνόλου. Όταν το μπροστινό και το πίσω μέτωπο συναντώνται, τα θραύσματα μπορούν να υποβληθούν σε επεξεργασία είτε ως μπροστινά ή πίσω θραύσματα. Τα επεξεργαζόμαστε ως μπροστινά θραύσματα όπως στον ακόλουθο ψευδοκώδικα:

```
if (fragDepth == nearestDepth) {
    //process front fragment
} else {
    //process back fragment
}
```

Η τιμή βάθους που εγγράφεται στον depth buffer είναι άμεσα διαθέσιμη στον fragment shader ως gl_FragCoord.z. Επομένως, θέλουμε να συγκρίνουμε το gl_FragCoord.z με το βάθος που είναι αποθηκευμένο στον depth buffer στο gl_FragCoord.xy. Επειδή τόσο το gl_FragCoord.z όσο και ο depth buffer μας είναι 32-bit float, μπορούμε να τα συγκρίνουμε.

Υπάρχουν δύο τρόποι για να εκτελεστεί η ανάμειξη άλφα. Ο πιο συνηθισμένος τρόπος είναι η σύνθεση θραυσμάτων από πίσω προς τα εμπρός χρησιμοποιώντας



Σχήμα 3.2: Μπροστά και πίσω peels της σκηνής.

την εξίσωση ανάμειξης

$$C_{dst} = A_{src} \times C_{src} + (1 - A_{src}) \times C_{dst} \tag{3.1}$$

Αυτή η εξίσωση δεν χρησιμοποιεί το κανάλι άλφα του color buffer προορισμού. Όταν αποκολλάμε στρώματα θραυσμάτων από μπροστά και κάνουμε σύνθεση από μπροστά προς τα πίσω, η εξίσωση ανάμειξης άλφα πρέπει να τροποποιηθεί. Ένα χρώμα θραύσματος (C_1, A_1) που αναμειγνύεται πάνω σε ένα χρώμα (C_2, A_2) πάνω σε ένα χρώμα φόντου C_0, παράγει το ακόλουθο χρώμα C_2':

$$C_1\text{'} = A_1 \times C_1 + (1 - A_1) \times C_0$$

$$C_2\text{'} = A_2 \times C_2 + (1 - A_2) \times C_1\text{'}$$

$$C_2\text{'} = A_2 \times C_2 + (1 - A_2) \times (A_1 \times C_1 + (1 - A_1) \times C_0)$$

$$C_2\text{'} = A_2 \times C_2 + (1 - A_2) \times A_1 \times C_1 + (1 - A_2) \times (1 - A_1) \times C_0$$

Η παραπάνω εξίσωση δείχνει ότι τα θραύσματα μπορούν να αναμειχθούν με σειρά από μπροστά προς τα πίσω με τις ακόλουθες ξεχωριστές εξισώσεις ανάμειξης:

$$C_{dst} = A_{dst} \times (A_{src} \times C_{src}) + C_{dst} \tag{3.2}$$

$$A_{dst} = (1 - A_{src}) \times A_{dst} \tag{3.3}$$

```
glEnable(GL_BLEND);
glBlendEquation(GL_FUNC_ADD);
glBlendFuncSeparate(GL_DST_ALPHA, GL_ONE, GL_ZERO, GL_ONE_MINUS_SRC_ALPHA);
```

Τέλος, τα θραύσματα που αναμειγνύονται συντίθενται με το χρώμα φόντου C_{bg} χρησιμοποιώντας έναν fragment shader που εφαρμόζει το $C_{dst} = A_{dst} \times C_{bg} + C_{dst}$.

3.2 Weighted Blended OIT

3.2.1 Εισαγωγή

Αυτή η τεχνική αποδίδει μη διαθλαστική, μονόχρωμη μετάδοση μέσω επιφανειών με χρώμα, χωρίς να απαιτείται ταξινόμηση ή νέα χαρακτηριστικά στο hardware. Στην πραγματικότητα, μπορεί να υλοποιηθεί με έναν απλό shader για οποιαδήποτε GPU που υποστηρίζει blending για την απόδοση με περισσότερα από 8 bits ανά κανάλι [7][8].

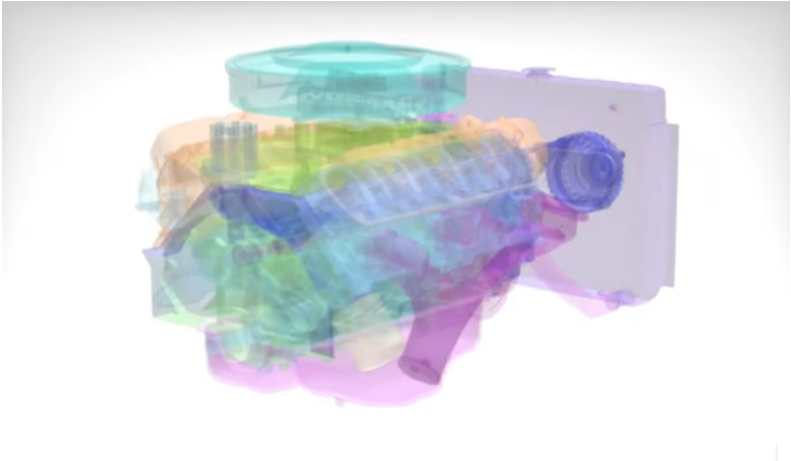
Λειτουργεί καλύτερα σε GPU με πολλαπλούς render targets και texture κινητής υποδιαστολής, όπου είναι ταχύτερη από τη ταξινομημένη διαφάνεια (sorted transparency) και αποφεύγει τα artifacts και το popping για τα συστήματα σωματιδίων. Επίσης, απαιτεί μικρότερο bandwidth ακόμη και από έναν 4-deep RGBA8 K-buffer και επιτρέπει την ανάμειξη σωματιδίων χαμηλής ανάλυσης με επιφάνειες πλήρους ανάλυσης, όπως το γυαλί.

Για την περίπτωση μικτής ανάλυσης, το μέγιστο κόστος μνήμης παραμένει αυτό του render target υψηλότερης ανάλυσης, αλλά το κόστος του bandwidth μειώνεται με βάση την αναλογία των επιφανειών χαμηλής ανάλυσης.

Η βασική ιδέα της μεθόδου Weighted Blended είναι ο ακριβής υπολογισμός της κάλυψης του φόντου από τις διαφανείς επιφάνειες, αλλά μόνο η προσέγγιση του φωτός που σκεδάζεται προς την κάμερα από τις ίδιες τις διαφανείς επιφάνειες. Ο αλγόριθμος επιβάλλει έναν ευρετικό συντελεστή μεταξύ των διάφανων επιφανειών που αυξάνεται με την απόσταση από την κάμερα.

3.2.2 Λειτουργία του αλγορίθμου

Το χρώμα των συνδυασμένων επιφανειών μερικής κάλυψης κυριαρχείται πάντα από τις επιφάνειες με την υψηλότερη κάλυψη, ανεξάρτητα από το πού εμφανίζονται στην ταξινόμηση βάθους. Επίσης, πολλαπλές επιφάνειες με διαφορετικά χρώματα και παρόμοια κάλυψη θα αποδώσουν το μέσο χρώμα τους, ανεξάρτητα από τη διάταξη. Σκεφτείτε την περίπτωση ενός λευκού σύννεφου που διέρχεται μπροστά από ένα



Σχήμα 3.3: Απόδοση σχεδίου CAD με την τεχνική αυτή.

σκούρο σύννεφο. Επιθυμούμε ένα αποτέλεσμα που να είναι πιο κοντά στο λευκό σύννεφο σε αυτή την περίπτωση, και το αντίθετο αν τα βάθη τους ανταλλάσσονταν [9].

Για την επίτευξη αυτού του στόχου, έχουμε την μέθοδο ανάμειξης με βάρη για τα χρώματα που μειώνονται με την απόσταση από την κάμερα:

$$C_f = \frac{\sum_{i=1}^{n} C_i \times w(z_i, a_i)}{\sum_{i=1}^{n} a_i \times w(z_i, a_i)} \left(1 - \prod_{i=1}^{n}(1 - a_i)\right) + C_0 \times \prod_{i=1}^{n}(1 - a_i) \tag{3.4}$$

Υποθέτουμε ότι το $|z|$ είναι μια τιμή στο χώρο της κάμερας που είναι μηδέν στο κέντρο της προβολής και μειώνεται μακριά από την κάμερα προς το -∞. Οποιαδήποτε μονότονα φθίνουσα, μη μηδενική συνάρτηση του $|z|$ στο αναμενόμενο εύρος βάθους, όπως w(z,α) = $|z|$-k ή w(z,α) = z - z_{far} + ε, δίνει στις πλησιέστερες επιφάνειες υψηλότερα βάρη. Δημιουργείται έτσι ένα στοιχείο απόκρυψης μεταξύ διάφανων επιφανειών. Μπορεί επίσης να επιλέξει κανείς μια συνάρτηση της ομοιογενούς τιμή του clip-space z αντί για μια γραμμική του χώρου της κάμερας $|z|$. Ανεξάρτητα από τη συνάρτηση βάρους, η καθαρή κάλυψη του φόντου παραμένει ακριβώς σωστή και το χρώμα θα είναι πάντα μια παρεμβολή των χρωμάτων από τις διαφανείς επιφάνειες - ποτέ δεν προεκτείνεται πέρα από αυτές, οπότε το αποτέλεσμα είναι πάντα αληθοφανές. Η συνάρτηση βάρους λειτουργεί σαν ένας εκτιμητής της απόκρυψης κάθε επιφάνειας, επιτρέποντας σε μια επιφάνεια να μετριάσει τη δική της συνεισφορά υποθέτοντας μια ομοιόμορφη κατανομή των άλλων επιφανειών μεταξύ αυτής και του θεατή, χωρίς ρητή πληροφορία σχετικά με αυτές τις επιφάνειες. Όταν αυτή η εκτίμηση είναι εσφαλμένη (π.χ. επειδή δεν υπάρχουν άλλες επιφάνειες), ο όρος

κανονικοποίησης διορθώνει την καθαρή συνεισφορά. Με αυτόν τον τρόπο, είναι μια στατική ευρετική προσέγγιση της πραγματικής συνάρτησης ορατότητας, συγκρίσιμη με τον στοχαστικό εκτιμητή ορατότητας του Enderton.[5] Σύμφωνα με την παραδοχή της ομοιόμορφης πυκνότητας του μέσου, η σωστή ευρετική της ορατότητας θα ήταν η εκθετική πτώση σύμφωνα με το νόμο Beer-Lambert (όπως στην ατμοσφαιρική προοπτική). Σκηνές με μικρό αριθμό διακριτών επιφανειών ή συστάδες καπνού, δεν έχουν ομοιόμορφη πυκνότητα. Επιπλέον, η τιμή μιας πραγματικής εκθετικής συνάρτησης θα υπερχείλιζε τη διαθέσιμη ακρίβεια σε έναν buffer κινητής υποδιαστολής 16 bit κοντά στην κάμερα, και να υπολείπονται σε ακρίβεια μακριά από την κάμερα. Έτσι, συνιστούμε ορθολογικά πολυωνυμικά βάρη που παρουσιάζουν πιο αργή ασυμπτωτική ανάπτυξη. Εάν η συνάρτηση βάρους εξαρτάται αποκλειστικά από το z, μια επιφάνεια πολύ κοντά στην κάμερα με τόσο χαμηλή κάλυψη ώστε να είναι ανεπαίσθητη κατά τη διάρκεια της διατεταγμένης σύνθεσης μπορεί στη συνέχεια να χρωματίσει ανεπιθύμητα απομακρυσμένες και πιο αδιαφανείς επιφάνειες. Συνεπώς, συνιστούμε η συνάρτηση βάρους να μειώνεται επίσης με την κάλυψη. Παρόλο που χρησιμοποιούμε προπολλαπλασιασμένη κάλυψη, σύμφωνα με την οποία τα σωματίδια χαμηλής κάλυψης έχουν ήδη πολύ μικρή συνεισφορά στο χρώμα, πολλές συναρτήσεις βάρους δίνουν πολύ μεγάλες τιμές κοντά στην κάμερα. Το γινόμενο του βάρους και της κάλυψης μπορεί ακόμα να είναι πολύ μεγάλο, εκτός αν το βάρος πέφτει ρητά με πολύ χαμηλές τιμές του α. Για τα επιμέρους αποτελεσμάτα στην παρούσα εργασία δίνουμε συναρτήσεις βάρους που σχεδιάσαμε για αυτές τις σκηνές. Συνιστούμε επίσης τέσσερις γενικές συναρτήσεις βάρους κατάλληλες για αυθαίρετες σκηνές με μεγάλο εύρος βάθους. Τις σχεδιάσαμε ώστε να λειτουργούν καλά για 16-bit συσσωρευτικούς buffers κινητής υποδιαστολής με 0,1 ≤ |z| ≤ 500:

$$w(z,a) = a \times max\left[10^{-}2, min\left[3\times10^{3}, \frac{10}{(10^{-}5 + (|z|/5)^{2} + (|z|/200)^{6})}\right]\right] \tag{3.5}$$

$$w(z,a) = a \times max\left[10^{-}2, min\left[3\times10^{3}, \frac{10}{(10^{-}5 + (|z|/10)^{3} + (|z|/200)^{6})}\right]\right] \tag{3.6}$$

$$w(z,a) = a \times max\left[10^{-}2, min\left[3\times10^{3}, \frac{0.03}{(10^{-}5 + (|z|/200)^{4})}\right]\right] \tag{3.7}$$

$$w(z,a) = a \times max[10^{-2}, 3\times10^{3} \times (1-d(z))^{3}] \tag{3.8}$$

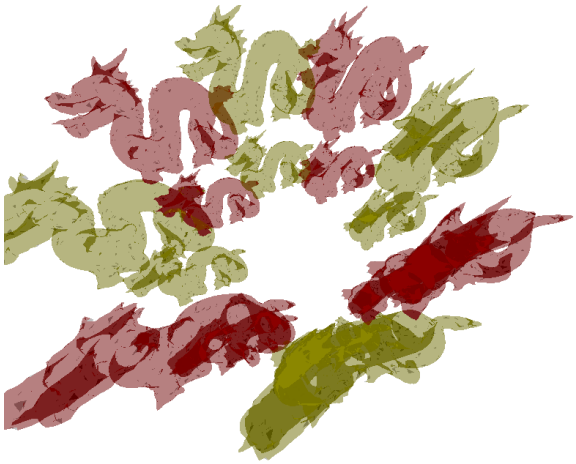
Όλοι οι εκθέτες μπορούν να αξιολογηθούν με επαναλαμβανόμενα γινόμενα - δεν απαιτείται πραγματική εκθετικοποίηση. Στην εξίσωση 5, d(z) είναι η τιμή στο gl_FragCoord.z κατά την εκτέλεση του OpenGL fragment shader, όπου όλες οι τιμές z είναι αρνητικές στο χώρο της κάμερας.

$$d(z) = \frac{(z_{near} \times z_{far})/z - z_{far}}{z_{near} - z_{far}} \tag{3.9}$$

3.2.3 Τροποποιήσεις του αλγορίθμου

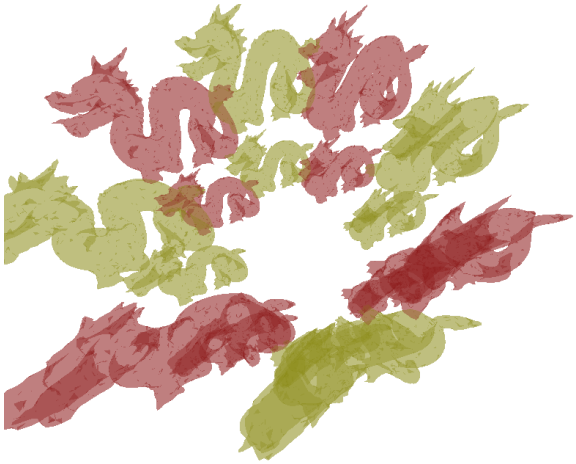
Έπειτα από σχετικές δοκιμές βελτίωσης του αλγορίθμου καταλήξαμε ότι σε συνδυασμό με την παρακάτω συνάρτηση βάρους, παίρνουμε το αποτέλεσμα που φαίνεται στο σχήμα 3.4(γ)

$$w(z,a) = a \times max\left[10^{-2}, min\left[8, 0.3 \times 10^5 + \left[\frac{1-z}{200}\right]^4\right]\right] \tag{3.10}$$



(α) Απόδοση με Dual Depth Peeling

(β) Απόδοση Weighted Blended με χρήση της συνάρτησης 3.1



(γ) Απόδοση Weighted Blended με χρήση της συνάρτησης 3.7

Σχήμα 3.4: Σύγκριση μεταξύ μεθόδων

3.3 Deep Weighted Blended

3.3.1 Εισαγωγή

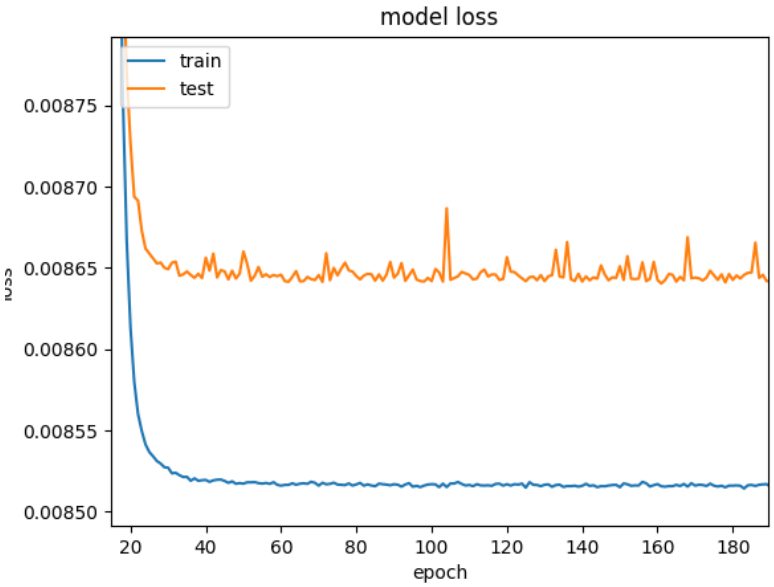
Η τεχνική αυτή είναι μία πιο εξελιγμένη τεχνική σε σχέση με την τεχνική βασισμένη στην στο Weighted Blended. Στην πραγματικότητα εκπαιδεύουμε ένα νευρωνικό δίκτυο με βάση την πληροφορία των fragments του Dual Depth Peeling με σκοπό να δημιουργήσουμε μια συνάρτηση βάρους που θα χρησιμοποιηθεί για σύνθεσή της τελικής εικόνας του Weighted Blended.

Η βασική ιδέα της μεθόδου Deep Weighted Blended είναι η δημιουργία μίας καλύτερης συνάρτησης βάρους που θα υπολογίζει τον συντελεστή του Dual Depth Peeling, και η χρήση της συνάρτησης αυτής στους fragment shaders του Weighted Blended.

3.3.2 Περιγραφή της μεθόδου

Η μέθοδος αυτή βασίζεται στην εκπαίδευση ενός νευρωνικού για την προσέγγιση του συντελεστή του Dual Depth Peeling (Εξίσωση 3.1). Για αρχή παράγουμε παράγουμε μία σκηνή με την μέθοδο Dual Depth Peeling από την οποία θα εξάγουμε τα απαραίτητα δεδομένα για την εκπαίδευση του δικτύου μας, τα οποία αναλύουμε στο επόμενο κεφάλαιο. Στην συνέχεια εξάγουμε το βάθος της σκηνής, το σύνολο των fragment για κάθε pixel και το άθροισμα των άλφα για να κάνουμε την πρόβλεψη του συντελεστή του Deep Weighted Blended. Τέλος, την σκηνή που έχουμε παράξει από το Weighted Blended την πολλαπλασιάζουμε με τους συντελεστές που έχουμε ήδη προβλέψει με το νευρωνικό.

Για το νευρωνικό χρησιμοποιήσαμε ένα απλό MLP που αποτελείται από 4 επίπεδα, το επίπεδο εισόδου που έχει 3 εισόδους(features) και 2 ακόμα κρυμμένα επίπεδα με 10 και 8 εισόδους αντίστοιχα και το τελευταίο επίπεδο που είναι η έξοδος. Η συνάρτηση ενεργοποίησης είναι η ReLU (Rectified Linear Unit), για optimizer χρησιμοποιήσαμε adam και για loss function την mse. Για την εκπαίδευση χρησιμοποιήθηκαν 150 εποχές και batch size 128. Για την εκπαίδευση χρησιμοποιήθηκε ένα σύνολο 1.544.955 στοιχείων από τα οποία το 33% (509.835) χρησιμοποιήθηκε για το τεστ. Κάθε γραμμή από τα στοιχεία αυτά περιείχε το (1-Ai), το γραμμικοποιημένο βάθος, το άθροισμα των άλφα και τον σύνολο των fragment για κάθε pixel.



Σχήμα 3.5: Σφάλμα εκπαίδευσης για 200 εποχές

Τα δεδομένα εξήχθησαν από τους shader σε αρχεία κειμένου και τα συνθέθηκαν σε ένα csv που περιείχε μία στήλη για το κάθε feature και η πρόβλεψη έγινε με για τον όρο (1-Ai).

Με learning rate 0.0001 βλέπουμε στο Σχήμα 3.5 οτι το train και το test loss να φτάνουνε σε τιμές πολύ κοντά στο μηδέν, ενώ φαίνεται από την 100ή εποχή και μετά δεν αλλάζει το σφάλμα. Αυτό το αποτέλεσμα θα μπορούσε να αλλάξει αν τα δεδομένα που λάβουμε περιέχουν σκηνή από διαφορετικές οπτικές και σκηνές με διαφορετικά μοντέλα στο χώρο με τυχαίες τιμές διαφάνειας. Τα δεδομένα εξήχθησαν από την ίδια σκηνή αλλά από 3 διαφορετικές γωνίες λήψης.

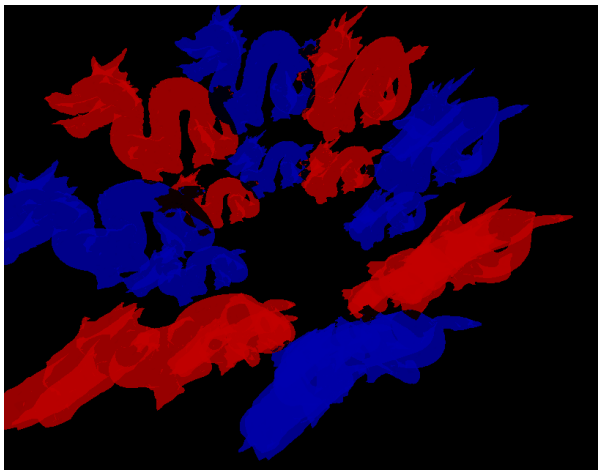
Κεφαλαιο 4

Αποτελεσματα

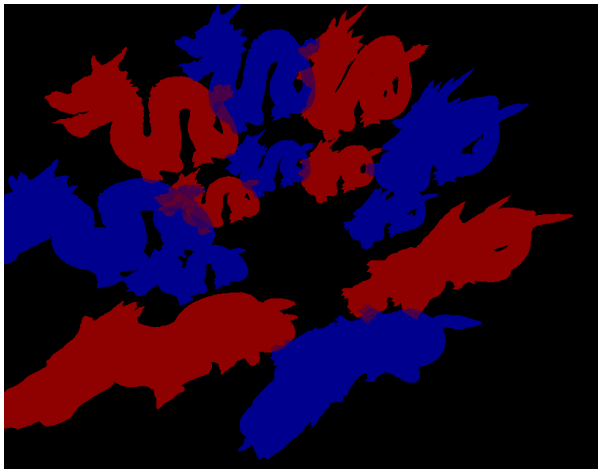
4.1 Πειραματική Αξιολόγηση

Για τα πειραματικά αποτελέσματα χρησιμοποιήσαμε μία σκηνή με 16 μοντέλα, με 11102 πολύγωνα το κάθε ένα.

Η εκπαίδευση έγινε σε ένα MLP με ένα hidden layer. Τα δεδομένα που χρησιμοποιήσαμε για την εκπαίδευση ήταν ο όρος A_dst (Συνάρτηση 3.3) του κάθε fragment, το σύνολο των fragment για κάθε pixel, το άθροισμα των a και τέλος το γραμμικοποιημένο βάθος της σκηνής. Δεδομένα μαζεύτηκαν από διάφορες οπτικές της σκηνής και για τυχαία άλφα. Τέλος παρατηρήθηκε ότι με πολύ υψηλές τιμές του άλφα στον αλγόριθμο του Weighted Blended, μετά την ανακατασκευή της εικόνας παρατηρήθηκαν μαύρα artifacts στα χρώματα. Όπως θα δούμε στη συνέχεια δοκιμές έγιναν για τυχαίες τιμές του άλφα και τυχαίες τιμές χρωμάτων.



(α) Απόδοση με Dual Depth Peeling και διαφάνεια 0.75



(β) Απόδοση Weighted Blended και διαφάνεια 0.75

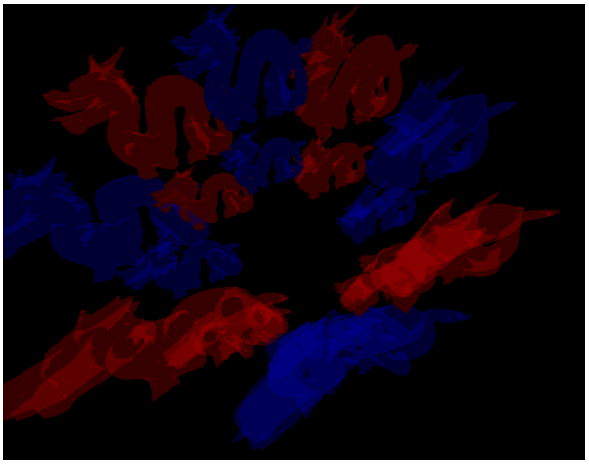
Σχήμα 4.1: Παραδείγματα μεθόδων

4.2 Ποιοτικά και Ποσοτικά Αποτελέσματα

Αρχικά στην υποενότητα αυτή θα παρουσιάσουμε τα αποτελέσματα για σταθερό χρώμα και άλφα και θα τα συγκρίνουμε με τις σκηνές του Depth Peeling και Weighted Blended. Τα χρώματα για το weighted blended φαίνονται λιγότερο φωτεινά γιατί στην εξίσωση του χρώματος πολλαπλασιάζεται και η συνάρτηση βάρους που είναι ανάλογη της τιμής του άλφα και του βάθους.

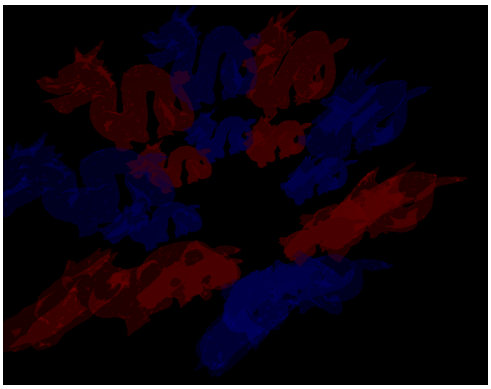
```
float w = weight(depthF, vColor.a);
...
accumAlpha = color.a*w;
accumColor = vec4(color.rgb*accumAlpha*w, w);
```

Όπως θα δούμε στα επόμενα παραδείγματα το MSE μειώνεται για την μέθοδο Deep Weighted Blended σε σύγκριση με την αρχική εικόνα του Weighted Blended αλγορίθμου. Για τα παραδείγματα με τυχαίο χρώμα και διαφάνεια βλέπουμε ότι το οπτικό αποτέλεσμα είναι αρκετά πιο κοντά στο αποτέλεσμα που θα είχε η Dual Depth Peeling μέθοδος.



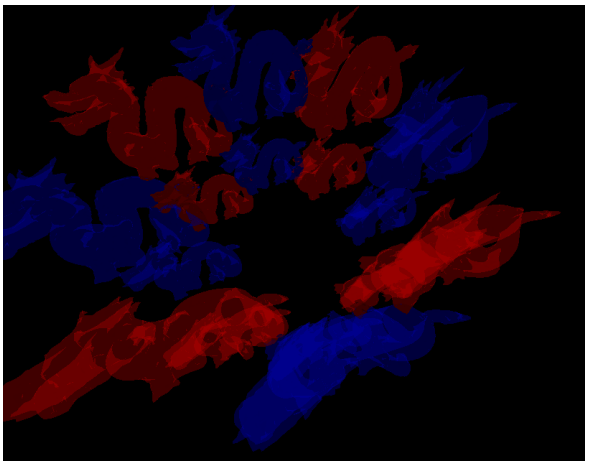
(α) Απόδοση με Dual Depth Peeling και διαφάνεια 0.25

(β) Απόδοση Weighted Blended και διαφάνεια 0.25 με MSE 15.95

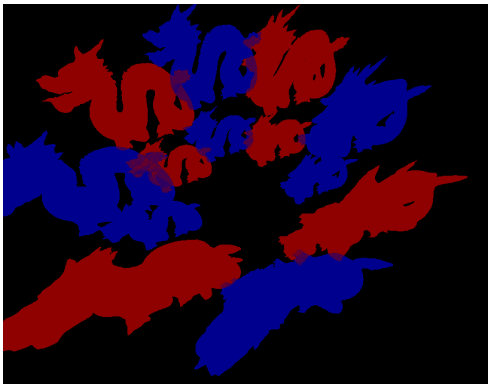


(γ) Αποτέλεσμα Deep Weighted Blended με MSE 11.79

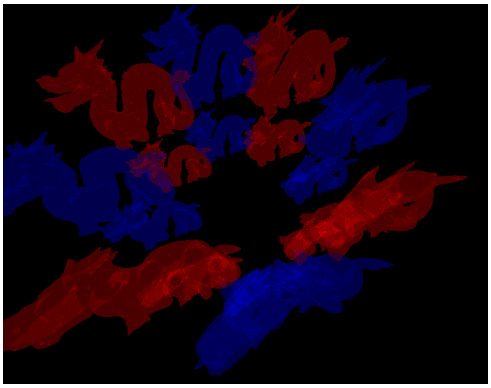
Σχήμα 4.2: Αποτελέσματα για διαφάνεια 0.25



(α) Απόδοση με Dual Depth Peeling και διαφάνεια 0.50

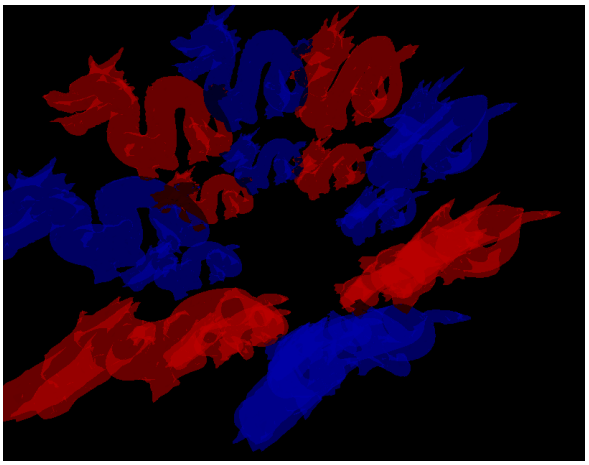


(β) Απόδοση Weighted Blended και διαφάνεια 0.50 με MSE 16.95

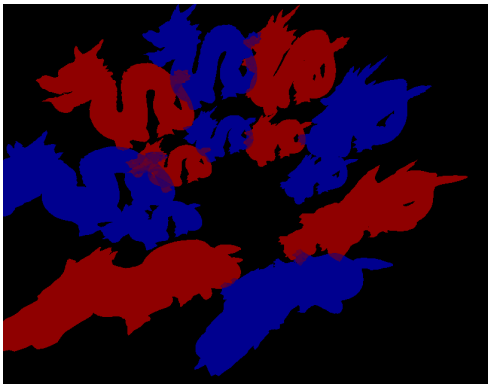


(γ) Αποτέλεσμα Deep Weighted Blended με MSE 10.56

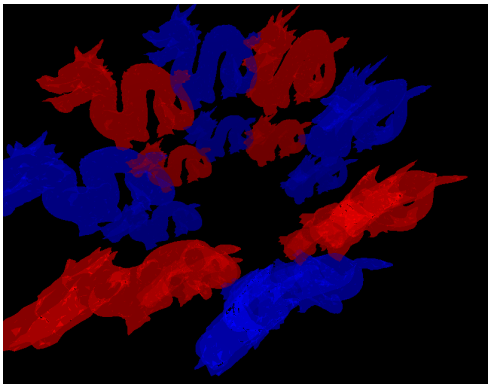
Σχήμα 4.3: Αποτελέσματα για διαφάνεια 0.50



(α) Απόδοση με Dual Depth Peeling και διαφάνεια 0.75

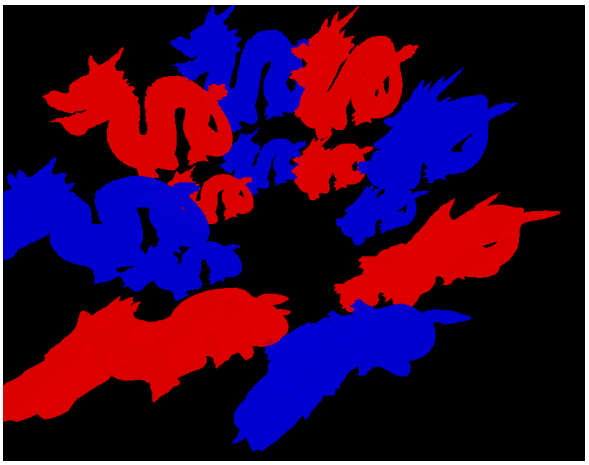


(β) Απόδοση Weighted Blended και διαφάνεια 0.75 με MSE 13.61

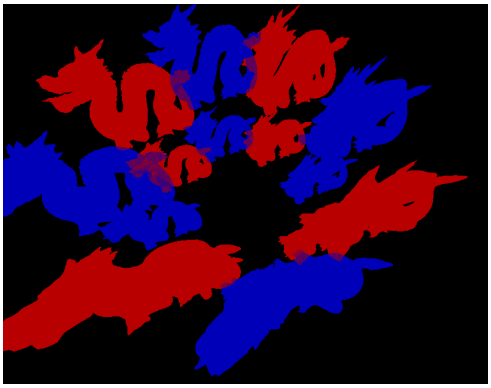


(γ) Αποτέλεσμα Deep Weighted Blended με MSE 11.26

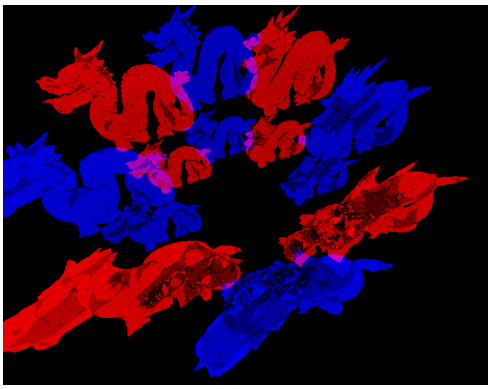
Σχήμα 4.4: Αποτελέσματα για διαφάνεια 0.75



(α) Απόδοση με Dual Depth Peeling και διαφάνεια 0.85



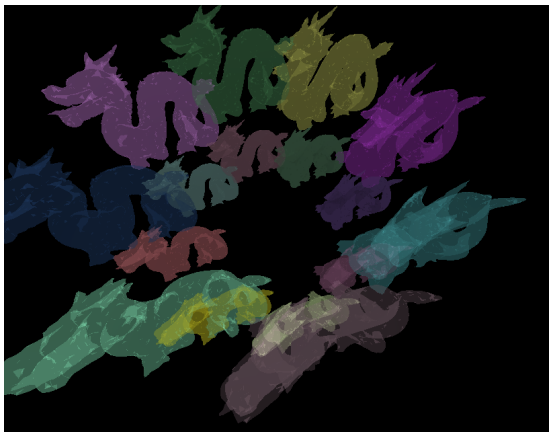
(β) Απόδοση Weighted Blended και διαφάνεια 0.85



(γ) Αποτέλεσμα Deep Weighted Blended

Σχήμα 4.5: Αποτελέσματα για διαφάνεια 0.85

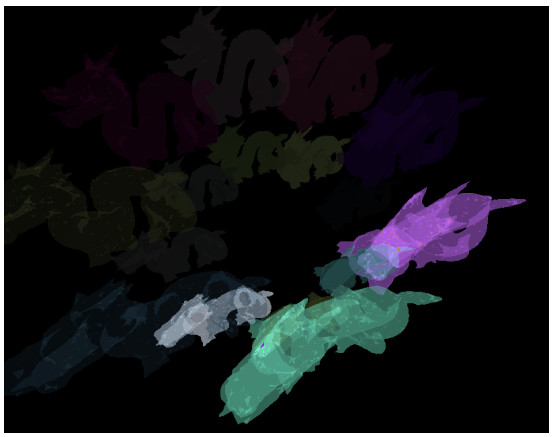
(α) Απόδοση με Weighted Blended και διαφάνεια 0.75 με τυχαίο χρώμα ανά μοντέλο



(β) Απόδοση Deep Weighted Blende

Σχήμα 4.6: Αποτελέσματα για διαφάνεια 0.75 και τυχαίο χρώμα

(α) Απόδοση με Weighted Blended και διαφάνεια 0.75 με τυχαίο χρώμα και διαφάνεια ανά μοντέλο



(β) Απόδοση Deep Weighted Blended

Σχήμα 4.7: Αποτελέσματα για τυχαίο χρώμα και διαφάνεια

Όπως βλέπουμε στα παραπάνω σχήματα το αποτέλεσμα της μεθόδου μας προσεγγίζει το αποτέλεσμα του Dual Depth Peeling με το μέσο σφάλμα να είναι χαμηλότερο για κάθε ένα από τα αποτελέσματα σε σύγκριση με τα αποτελέσματα της Weighted Blended Transparency, για τα Σχήματα 4.2, 4.3, 4.4. Στο Σχήμα 4.5 παρατηρούμε ότι για πολύ υψηλή τιμή του άλφα, 0.85 στην περίπτωσή μας, υπάρχουν μαύρα artifacts στα κόκκινα μοντέλα με τα μπλε μοντέλα να μην προσεγγίζουν σε διαφάνεια την αρχική εικόνα του Dual Depth Peeling. Στο Σχήμα 4.6 για σταθερό συντελεστή διαφάνειας (0.75 άλφα) και τυχαίο χρώμα ανά μοντέλο βλέπουμε ότι

οπτικά η διαφάνεια προσεγγίζεται κανονικά με την μέθοδό μας, σε αντίθετη περίπτωση στο Σχήμα 4.7 βλέπουμε ότι με τυχαίο συντελεστή διαφάνειας για κάθε μοντέλο και τυχαίο χρώμα το αποτέλεσμα της μεθόδου μας προσεγγίζει την διαφάνεια με κάποια όμως μωβ χρώματος artifacts στο πράσινο μοντέλο. Για την εξάλειψη των artifacts συνίσταται η εκπαίδευση ενός πιο σύνθετου νευρωνικού μοντέλου με δεδομένα από περισσότερες σκηνές, τυχαίες τιμές του άλφα και ενδεχομένως διαφορετικά μοντέλα.

Πίνακας 4.1: Πίνακας λάθους σε σύγκρηση με Dual Depth Peeling με χρήση της συνάρτησης λάθους MSE.

	Weighted Blending	Deep Weighted Blending
0.25 Transparency	15.95647	11.79113
0.50 Transparency	16.35498	10.50012
0.75 Transparency	13.61550	11.26541

Τέλος, στον πίνακα βλέπουμε ότι και για τα τρία αποτελέσματα με σταθερό χρώμα και σταθερό άλφα το μέσο σφάλμα έχει μειωθεί και στις τρεις περιπτώσεις που σημαίνει ότι η μέθοδός προσεγγίζει τον Dual Depth Peeling αλγόριθμο. Για τα αποτελέσματα που περιείχαν artifacts δεν έγιναν μετρήσεις γιατί φαίνονται ξεκάθαρες οπτικές αλλοιώσεις.

Κεφαλαιο 5

Συμπερασματα και Μελλοντικες Κατευθυνσεις

5.1 Συμπεράσματα

Από τα παραδείγματα συμπεραίνουμε ότι με την μέθοδο μας μπορέσαμε να προσεγγίσουμε ένα αποτέλεσμα πιο κοντά στο αποτέλεσμα της τέλειας διαφάνειας. Με την δυνατότητα συλλογής περισσότερων δεδομένων, χρήση πιο σύνθετου δικτύου και καλύτερης εκπαίδευσης του νευρωνικού μπορούμε να προσεγγίσουμε και μεγαλύτερες τιμές διαφάνειας χωρίς οπτικές αλλοιώσεις καθώς και πιο πολύπλοκες σκηνές. Τέλος, μία από τις μελλοντικές εξελίξεις της μεθόδου αυτής θα ήταν η δυνατότητα να τρέχει ταυτόχρονα σε επίπεδο shader για παραγωγή της βελτιωμένης σκηνής σε πραγματικό χρόνο.

Bibliography

[1] Louis Bavoil and Kevin Myers. "Order independent transparency with dual depth peeling". In: *NVIDIA OpenGL SDK* 1 (2008), p. 12.

[2] Loren Carpenter. "The A-buffer, an antialiased hidden surface method". In: *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. 1984, pp. 103–108.

[3] Eric Enderton et al. "Stochastic transparency". In: *IEEE transactions on visualization and computer graphics* 17.8 (2010), pp. 1036–1047.

[4] Cass Everitt. "Interactive order-independent transparency". In: *White paper, nVIDIA* 2.6 (2001), p. 7.

[5] Norman P Jouppi—Compaq, Chun-Fa Chang, and UNC Chapel Hill. "Z3: An Economical Hardware Technique for High-Quality Antialiasing and Transparency". In: ().

[6] Abraham Mammen. "Transparency and antialiasing algorithms implemented with the virtual pixel maps technique". In: *IEEE Computer graphics and Applications* 9.4 (1989), pp. 43–55.

[7] Morgan McGuire and Louis Bavoil. "Weighted blended order-independent transparency". In: *Journal of Computer Graphics Techniques* 2.4 (2013).

[8] Houman Meshkin. "Sort-independent alpha blending". In: *GDC Talk* 2.4 (2007).

[9] Kevin Myers and Louis Bavoil. "Stencil routed A-buffer." In: *SIGGRAPH Sketches*. Citeseer. 2007, p. 21.

[10] Peter Young. "Coverage sampled antialiasing". In: *Technical report, NVIDIA* (2006).

Παραρτημα Α

Κωδικας για την μεθοδο του Dual Depth Peeling

Α.1 Κώδικας Αλγορίθμου

Στο σημείο αυτό βλέπουμε τον κώδικα για την απόδοση του Dual Depth Peeling. Αρχικοποιούμε τους buffer για τα βάθη των τομών.

```
function draw() {

    ////////////////////////////////
    // 1. Initialize min-max depth buffer
    ////////////////////////////////

    gl.bindFramebuffer( gl.DRAW_FRAMEBUFFER, blendBackBuffer );
    gl.clearColor( 0, 0, 0, 0 );
    gl.clear( gl.COLOR_BUFFER_BIT );

    gl.bindFramebuffer( gl.FRAMEBUFFER, depthPeelBuffers[ 0 ] );
    gl.drawBuffers( [ gl.COLOR_ATTACHMENT0 ] );
    gl.clearColor( DEPTH_CLEAR_VALUE, DEPTH_CLEAR_VALUE, 0, 0 );
    gl.clear( gl.COLOR_BUFFER_BIT );
```

```
gl.bindFramebuffer( gl.DRAW_FRAMEBUFFER, depthPeelBuffers[ 1 ] );
gl.clearColor( -MIN_DEPTH, MAX_DEPTH, 0, 0 );
gl.clear( gl.COLOR_BUFFER_BIT );

gl.bindFramebuffer( gl.DRAW_FRAMEBUFFER, colorBuffers[ 0 ] );
gl.drawBuffers( [ gl.COLOR_ATTACHMENT0, gl.COLOR_ATTACHMENT1 ] );
gl.clearColor( 0, 0, 0, 0 );
gl.clear( gl.COLOR_BUFFER_BIT );

gl.bindFramebuffer( gl.DRAW_FRAMEBUFFER, colorBuffers[ 1 ] );
gl.clearColor( 0, 0, 0, 0 );
gl.clear( gl.COLOR_BUFFER_BIT );

// draw depth for first pass to peel
gl.bindFramebuffer( gl.DRAW_FRAMEBUFFER, depthPeelBuffers[ 0 ] );
gl.drawBuffers( [ gl.COLOR_ATTACHMENT0 ] );
gl.blendEquation( gl.MAX );

gl.useProgram( depthPeelProgram );
gl.uniform1i( depthPeelDepthLocation, 3 );
gl.uniform1i( depthPeelFrontColorLocation, 4 );

gl.bindVertexArray( objectArray );

...
gl.bindBuffer( gl.ARRAY_BUFFER, matrixBuffer );
gl.bufferSubData( gl.ARRAY_BUFFER, 0, modelMatrixData );
gl.drawElementsInstanced( gl.TRIANGLES, indicesArray.length, gl.
    UNSIGNED_INT, 0, objects.length );//using this for object loaded
```

Στην συνέχεια έχουμε την Ping-Pong λειτουργία του Dual Depth Peeling. Στην δικιά μας περίπτωση έχουμε αριθμό περασμάτων 4.

```
////////////////////////////////////
// 2. Dual Depth Peeling Ping-Pong
////////////////////////////////////

var readId, writeId;
var offsetRead, offsetBack;
```

```
//I NEED A [X][WIDTH*HEIGHT*4] SIZE ARRAY OF NUMBERS TO KEEP PIXELS
for ( var pass = 0; pass < NUM_PASS; pass ++ ) {

    readId = pass % 2;
    writeId = 1 - readId; // ping-pong: 0 or 1

    gl.bindFramebuffer( gl.DRAW_FRAMEBUFFER, depthPeelBuffers[ writeId
        ] );
    gl.drawBuffers( [ gl.COLOR_ATTACHMENT0 ] ); //C_A0 = RG and FLOAT
    //READ BUFFERS AND THEN THE PIXELS THEY GIVE
    //EXTRACTING DEPTH BUFFER INFORMATION
    ...
    gl.clearColor( DEPTH_CLEAR_VALUE, DEPTH_CLEAR_VALUE, 0, 0 );
    gl.clear( gl.COLOR_BUFFER_BIT );

    gl.bindFramebuffer( gl.DRAW_FRAMEBUFFER, colorBuffers[ writeId ] );
    gl.drawBuffers( [ gl.COLOR_ATTACHMENT0, gl.COLOR_ATTACHMENT1 ] );
        //C_A1 = RGBA and HALF_FLOAT

    //EXTRACTING COLOR BUFFER INFORMATION
    ...

    gl.clearColor( 0, 0, 0, 0 );
    gl.clear( gl.COLOR_BUFFER_BIT );

    gl.bindFramebuffer( gl.DRAW_FRAMEBUFFER, depthPeelBuffers[ writeId
        ] );
    gl.drawBuffers( [ gl.COLOR_ATTACHMENT0, gl.COLOR_ATTACHMENT1, gl.
        COLOR_ATTACHMENT2 ] ); //C_A2 = RGBA and HALF_FLOAT

    gl.blendEquation( gl.MAX );
    // update texture uniform
    offsetRead = readId * 3;
    gl.useProgram( depthPeelProgram );
    gl.uniform1i( depthPeelDepthLocation, offsetRead );
    gl.uniform1i( depthPeelFrontColorLocation, offsetRead + 1 );

    // draw geometry
    gl.bindVertexArray( objectArray );
    gl.drawElementsInstanced( gl.TRIANGLES, indicesArray.length, gl.
        UNSIGNED_INT, 0, objects.length ); //using this for object
```

```
            loaded

        // blend back color separately
        offsetBack = writeId * 3;
        gl.bindFramebuffer( gl.DRAW_FRAMEBUFFER, blendBackBuffer );
        gl.drawBuffers( [ gl.COLOR_ATTACHMENT0 ] );
        gl.blendEquation( gl.FUNC_ADD );
        gl.blendFuncSeparate( gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA, gl.ONE,
            gl.ONE_MINUS_SRC_ALPHA );
        gl.useProgram( blendBackProgram );
        gl.uniform1i( blendBackColorLocation, offsetBack + 2 );
        gl.bindVertexArray( quadArray );
        gl.drawArrays( gl.TRIANGLES, 0, 6 );

    }

    //////////////////////////
    // 3. Background color
    //////////////////////////
    gl.bindFramebuffer( gl.FRAMEBUFFER, null );
    gl.clearColor( 1,1,1,1);
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.blendFunc( gl.ONE, gl.ONE_MINUS_SRC_ALPHA );
    gl.useProgram( finalProgram );
    gl.uniform1i( finalFrontColorLocation, offsetBack + 1 );
    gl.bindVertexArray( quadArray );
    gl.drawArrays( gl.TRIANGLES, 0, 6 );

}
```

A.2 Shaders αλγορίθμου

Ο παρακάτω κώδικας χρησιμοποιείται για την δημιουργία κάθε peel του αλγορίθμου. Όπως φαίνεται και στο τέλος χρησιμοποιούμε φωτισμό για την δημιουργία της σκηνής.

```
#version 300 es
precision highp float;
```

```
precision highp sampler2D;

...

layout(location=0) out vec2 depth;  // RG32F, R - negative front depth,
    G - back depth
layout(location=1) out vec4 frontColor;
layout(location=2) out vec4 backColor;

void main() {

    // ------------------------
    // dual depth peeling
    // ------------------------

    float fragDepth = gl_FragCoord.z;   // 0 - 1

    ivec2 fragCoord = ivec2(gl_FragCoord.xy);
    vec2 lastDepth = texelFetch(uDepth, fragCoord, 0).rg;
    vec4 lastFrontColor = texelFetch(uFrontColor, fragCoord, 0);

    // depth value always increases
    // so we can use MAX blend equation
    depth.rg = vec2(-MAX_DEPTH);

    // front color always increases
    // so we can use MAX blend equation
    frontColor = lastFrontColor;

    // back color is separately blend afterwards each pass
    backColor = vec4(0.0);

    float nearestDepth = -lastDepth.x;
    float furthestDepth = lastDepth.y;
    float alphaMultiplier = 1.0 - lastFrontColor.a;


    if (fragDepth < nearestDepth || fragDepth > furthestDepth) {
        // Skip this depth since it's been peeled.
        return;
    }
```

```
if (fragDepth > nearestDepth && fragDepth < furthestDepth) {
    // This needs to be peeled.
    // The ones remaining after MAX blended for
    // all need-to-peel will be peeled next pass.
    depth.rg = vec2(-fragDepth, fragDepth);
    return;
}


//
    ------------------------------------------------------------------

// If it reaches here, it is the layer we need to render for this
    pass
//
    ------------------------------------------------------------------


vec3 position = vPosition.xyz;
vec3 normal = normalize(vNormal.xyz);
vec2 uv = vUV;

vec4 baseColor = vColor * texture(uTexture, uv);
vec3 eyeDirection = normalize(uEyePosition.xyz - position);
vec3 lightVec = uLightPosition.xyz - position;
vec3 lightDirection = normalize(lightVec);
vec3 reflectionDirection = reflect(-lightDirection, normal);
float nDotL = max(dot(lightDirection, normal), 0.0);
float diffuse = nDotL;
float ambient = 1.0;
float specular = pow(max(dot(reflectionDirection, eyeDirection),
    0.0), 20.0);

vec4 color = vec4((ambient + diffuse + specular) * baseColor.rgb,
    vColor.a);

// dual depth peeling
// write to back and front color buffer
if (fragDepth == nearestDepth) {
        frontColor.rgb += color.rgb * color.a * alphaMultiplier;
```

```
            frontColor.a = 1.0 - alphaMultiplier * (1.0 - color.a);
            //frontColor.a += (color.a); //used for extracting sum of
                alpha
    } else {
            backColor += color;
    }
}
```

Ο επόμενος shader χρησιμοποιείται για την σύνθεση των peels.

```
#version 300 es
...

void main() {
    ivec2 fragCoord = ivec2(gl_FragCoord.xy);
    vec4 frontColor = texelFetch(uFrontColor, fragCoord, 0);
    vec4 backColor = texelFetch(uBackColor, fragCoord, 0);
    float alphaMultiplier = 1.0 - frontColor.a;

    fragColor = vec4( frontColor.rgb + alphaMultiplier * backColor.rgb,
        frontColor.a + backColor.a );
    //fragColor = frontColor; //used for extracting (1-a), sum of
       alphas and linearized
}
```

Παραρτημα Β

Κωδικας της μεθοδου Weighted Blended

Β.1 Κώδικας αλγορίθμου

```
function draw() {

    ////////////////////////////////
    // Draw Objects
    ////////////////////////////////
    gl.bindFramebuffer(gl.FRAMEBUFFER, accumBuffer);
    gl.useProgram(accumProgram);
    gl.bindVertexArray(objectArray);

    gl.drawBuffers([gl.COLOR_ATTACHMENT0, gl.COLOR_ATTACHMENT1] );
    gl.readBuffer(gl.COLOR_ATTACHMENT0);//accumTarget
    ...

    gl.readBuffer(gl.COLOR_ATTACHMENT1);//accumAlphaTarget
    ...

    ...

    gl.bindBuffer(gl.ARRAY_BUFFER, matrixBuffer);
```

```
    gl.bufferSubData(gl.ARRAY_BUFFER, 0, modelMatrixData);
    gl.blendFuncSeparate(gl.ONE, gl.ONE, gl.ZERO, gl.ONE_MINUS_SRC_ALPHA);
    gl.clearColor(0,0,0,1);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.drawElementsInstanced( gl.TRIANGLES, indicesArray.length, gl.
        UNSIGNED_INT, 0, objects.length );

    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
    gl.useProgram(drawProgram);
    gl.bindVertexArray(quadArray);
    gl.clearColor(1,1,1,1); //background color
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.blendFunc(gl.ONE, gl.ONE_MINUS_SRC_ALPHA);
    gl.drawArrays(gl.TRIANGLES, 0, 6);

    requestAnimationFrame( draw );
}
```

B.2 Shaders μεθόδου

```
    #version 300 es
    precision highp float;

    ...

    float weight(float z, float a) {
        return pow(a, 1.0)*max(clamp(min(0.3/(1e-5 + pow((1.0-z)/200.0,4.0)
            ),8.0),1e-2,1.0),pow(10.0,-2.0));
    }

    float LinearizeDepth(float depth)
    {
        float z = depth * 2.0 - 1.0; // back to NDC
        return (2.0 * near * far) / (far + near - z * (far - near));
    }

    void main() {
        // ------------------------
        // Getting the depth values
        // ------------------------
```

```
    float fragDepth = gl_FragCoord.z;    // 0 - 1

    ivec2 fragCoord = ivec2(gl_FragCoord.xy);

    float depthF = LinearizeDepth(fragDepth);

    vec3 position = vPosition.xyz;
    vec3 normal = normalize(vNormal.xyz);
    vec2 uv = vUV;

    vec4 baseColor = vColor;
    vec3 eyeDirection = normalize(uEyePosition.xyz - position);
    vec3 lightVec = uLightPosition.xyz - position;
    vec3 lightDirection = normalize(lightVec);
    vec3 reflectionDirection = reflect(-lightDirection, normal);
    float nDotL = max(dot(lightDirection, normal), 0.0);
    float diffuse = nDotL;
    float ambient = 1.0;
    float specular = pow(max(dot(reflectionDirection, eyeDirection),
        0.0), 20.0);

    vec4 color = vec4((ambient + diffuse + specular) * baseColor.rgb,
        vColor.a);

    float w = weight(depthF, vColor.a);
    color.rgb *= color.a;

    accumAlpha = color.a*w;
    accumColor = vec4(color.rgb*accumAlpha*w, w);
}
```

Στο τέλος πολλαπλασιάζουμε το rgb με το accumAlpha αλλα και το w για να πάρουμε το επιθυμητό αποτέλεσμα.

Σύντομο Βιογραφικο

Ο Κωνσταντίνος Παπακώστας γεννήθηκε στα Ιωάννινα το 1994, Έλαβε το δίπλωμα του από το πανεπιστήμιο Ιωαννίνων με τίτλο Μηχανικός Ηλεκτρονικών Υπολογιστών και Πληροφορικής, το 2019. Αυτή την περίοδο βρίσκεται στο πρόγραμμα μεταπτυχιακών σπουδών του ίδιου τμήματος. Ο ερευνητικός του τομέας είναι τα γραφικά υπολογιστών.