

Πανεπιστήμιο Πατρών, Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

ART GALLERY

Ηλίας Ουζούνης
up1083749

3 Μαρτίου 2024

Περιεχόμενα

1 Cylindrical Room	3
2 Αντικείμενα και σκιές	6
3 FBOs	7
4 Artstyles	8
4.1 Floyd-Steinberg Dithering	8
4.2 Painterly Rendering	8
4.3 Fish Eye Effect	10
4.4 Toon Shading	10
4.5 Chromatic Aberration	10
5 Bump Rendering	12
6 Parallax Mapping	15

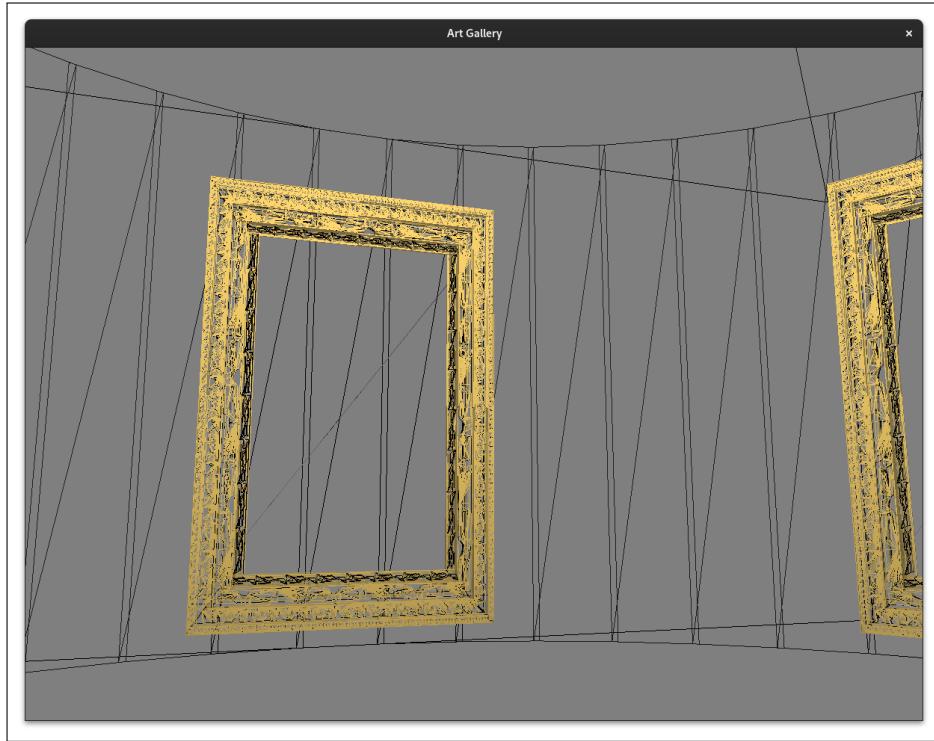
GENERAL APPROACH

Για αυτό το πρότζεκτ, υλοποιήθηκε μια πολύ object oriented προσέγγιση. Όλα τα αντικείμενα στην σκηνή (πίνακες, καρέκλες, ακόμα και τοίχοι - πατώματα) κάνουν inherit από την κλάση Object η οποία χειρίζεται όλα τα rendering, textures, uniforms κλπ. Αυτό έδινε μια ευελιξία στις ιδιότητες των αντικειμένων όσο άλλαζε το πρότζεκτ από ερώτημα σε ερώτημα. Επιπλέον ήταν ένας τρόπος να διατηρηθεί ο κώδικας καιθαρός και οργανωμένος καθώς το μέγεθος των αρχείων αυξανόταν συνεχώς.

Εκτός από τα αντικείμενα, όλα τα FBOs κληρονομούν από μία κλάση FBO. Έτσι όλα τα FBOs που χρειάστηκαν είχαν παρόμοια μορφή και ήταν πιο εύκολη η εναλλαγή μεταξύ τους.

1 CYLINDRICAL ROOM

Ένα τέλεια κυλινδικό δωμάτιο είναι αδύνατον να υλοποιηθεί στο OpenGL καθώς αυτό απαιτεί απεριόριστο αριθμό vertices. Για να προσεγγιστεί ένας κύλινδρος, επιλέγουμε πολλά σημεία ισοκατανεμημένα στο περίγυρο ενός κύκλου και τα ενώνουμε, δημιουργώντας ένα πολύγωνο με πολλές πλευρές.



Σχήμα 1.1: Wireframe του κυλινδικού δωματίου

Για το ταβάνι και το πάτωμα, στην αρχή είχα φτιάξει πολλά τρίγωνα από τις κορυφές που ορίστηκαν από τον κύκλο και το κέντρο του. Αυτό δημιουργούσε πολλά λεπτά τρίγωνα που συνέχιλναν στο κέντρο του δωματίου. Αυτή η προσέγγιση ήταν πιο δύσκολη στα UVs για τα textures του ταβανιού και του πατώματος. Αντί αυτού, επέλεξα να φτιάξω ένα μεγάλο παραλληλόγραμμο που καλύπτει το δωμάτιο. Παρόλο που θα προεξέχει, αυτό δεν θα φαίνεται καθώς θα είναι χρυμμένο από τους τοίχους. Αυτό έκανε τα UVs πολύ πιο εύκολα και τα textures πιο ρεαλιστικά.

Οι πίνακες τώρα είναι ισομοιρασμένοι στον κύκλο και έχουν τοποθετηθεί κοντά στον τοίχο. Επέλεξα να έχω 6 τελικά, 1 για κάθε διαφορετικό shading style και έναν default. Ενδιαφέρουσα είναι η εναλλαγή δωματίου 'μπαίνοντας' μέσα στον κάθε πίνακα. Αυτό επιτυγχάνεται θεωρώντας ένα bounding box για τον παίχτη και ελέγχοντας πότε συγκρούεται με τον πίνακα. Το bounding box του παίχτη είναι ένα παραλληλεπίπεδο που ορίζεται με 8 σημεία και δεν

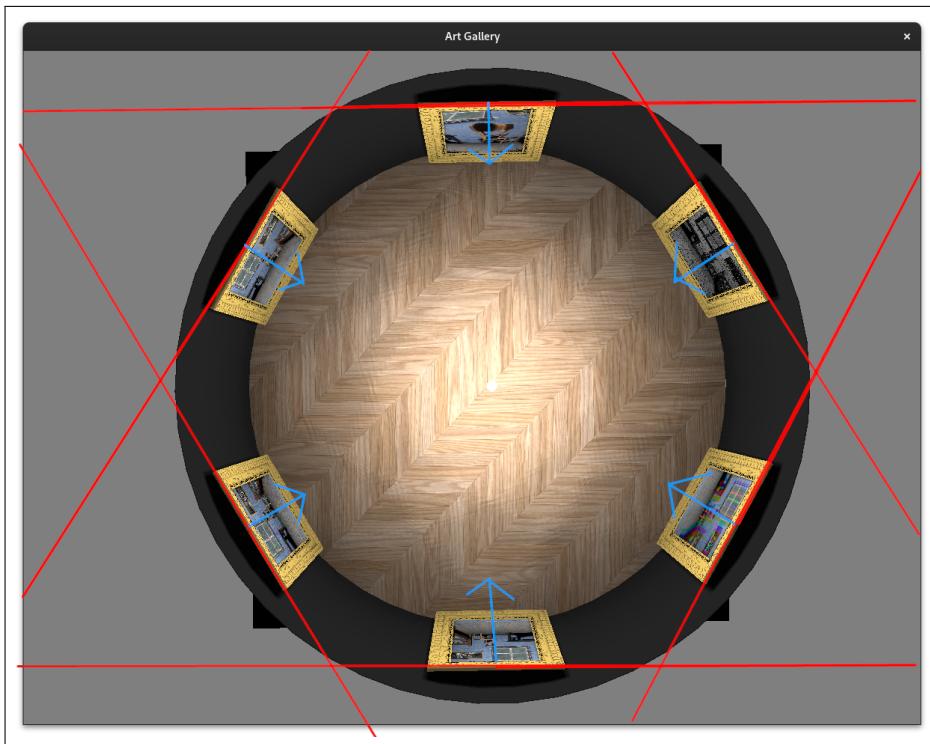


Σχήμα 1.2: Ταβάνι και πάτωμα. Είναι σκιασμένα γιατί το φως μπλοκάρεται από τους τοίχους

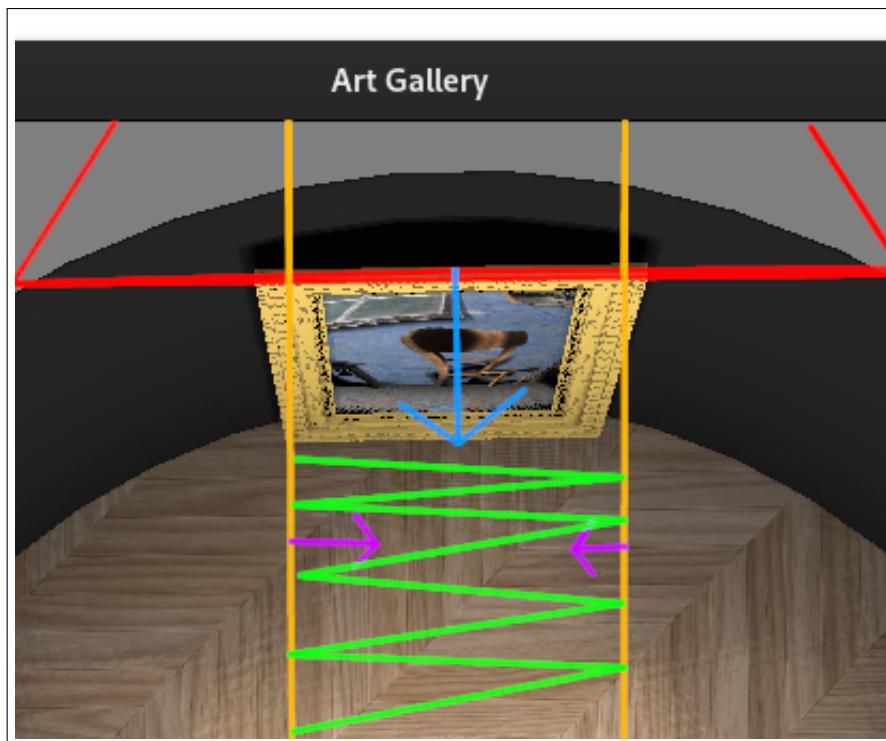
περιστρέφεται μαζί με τον παίχτη, απλά μετακινείται. Για να ελέγξουμε πότε αυτό το παραλληλεπίπεδο συγχρούεται με τον πίνακα αρχικά ελέγχουμε αν το y που βρίσκεται ο παίχτης είναι κατάλληλο. Έπειτα βρίσκουμε το επίπεδο που ο κάθε πίνακας ορίζει από το **normal** του. Τότε, παίρνουμε την εξίσωση του επιπέδου $dot(normal, point) + d = 0$ και βάζουμε τις κορυφές του παραλληλεπίπεδου. Αν για κάποιες κορυφές το $dot(normal, point) + d$ είναι θετικό ενώ για κάποιο άλλο σημείο αρθρικό, σημαίνει ότι 2 σημεία του bounding box βρίσκονται σε αντίθετα μέρη του επιπέδου, άρα τέμνει το επίπεδο του πίνακα. Αν όλα είναι ομόσημα σημαίνει πως βρίσκεται αποκλιστικά στην μία πλευρά του επιπέδου και σίγουρα δεν ακουμπάει τον πίνακα.

Αυτή η τεχνική δουλεύει αρκετά καλά από μόνη της γιατί τα επίπεδα βρίσκοντα εντός του δωματίου μόνο εκεί που υπάρχουν οι πίνακες. Ετσι δεν υπάρχει περίπτωση να τέμνει ο παίχτης το επίπεδο κάπου πολύ μακριά του πίνακα και να μεταφέρεται στο άλλο δωμάτιο αναπάντεχα. Όμως, κοντά στα όρια του πίνακα έχουμε εσφαλμένες συγχρούσεις. Για να καλύψουμε και αυτή την περίπτωση προσθέτουμε μία ακόμα συνθήκη. Περιστρέφοντας το **normal** του πίνακα κατά $+90$ και -90 μοίρες, παίρνουμε δύο κάθετα επίπεδα στον πίνακα. Για να θεωρηθεί η σύγχρουση σωστή πρέπει το **position** του παίχτη να βρίσκεται στην πράσινη περιοχή, δηλαδή να δίνει θετικές τιμές αν το βάλουμε στην εξίσωση των 2 επιπέδων.

Όταν γίνεται τελικά η σύχρουση, ο παίχτης μεταφέρεται στο κατάλληλο δωμάτιο και του δίνεται μια αρχική ταχύτητα, σαν να πηδάει έξω από τον πίνακα.



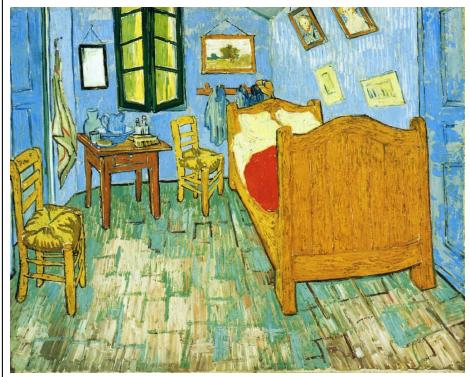
Σχήμα 1.3: Τα επίπεδα που ορίζουν τα normals των πινάκων



Σχήμα 1.4: Τα δύο κάθετα επίπεδα και η έγκυρη περιοχή

2 Αντικείμενα και σκιές

Κάθε στυλ shading αντιστοιχεί σε ένα κυβικό δωμάτιο. Απεικονίζουν μια abstract ερμηνεία του γνωστού πίνακα του Vincent Van Gogh, Bedroom in Arles.

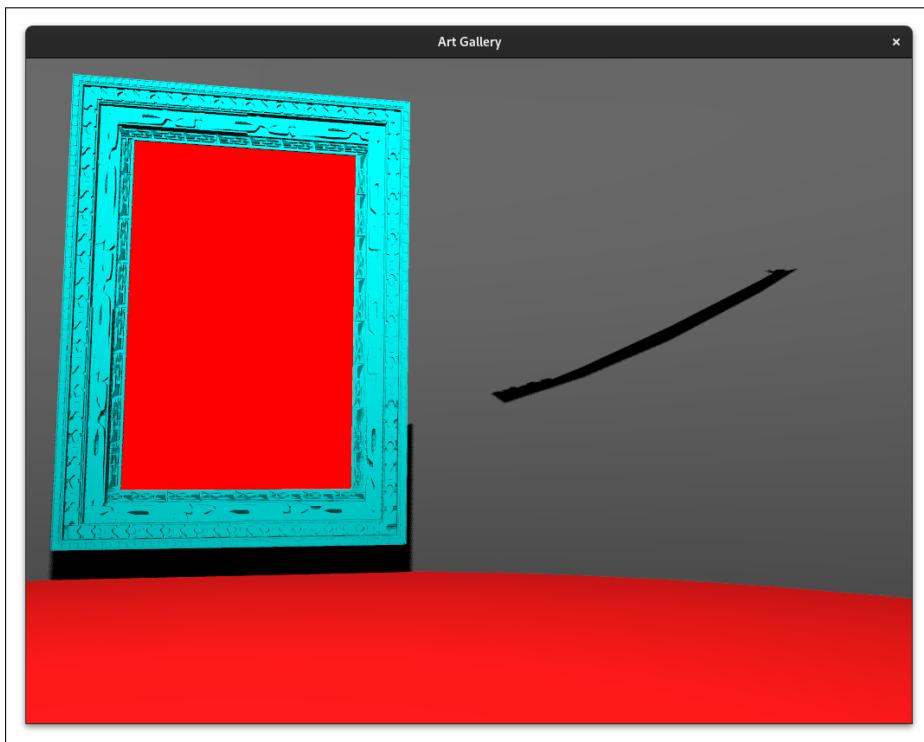


Σχήμα 2.1: Bedroom in Arles



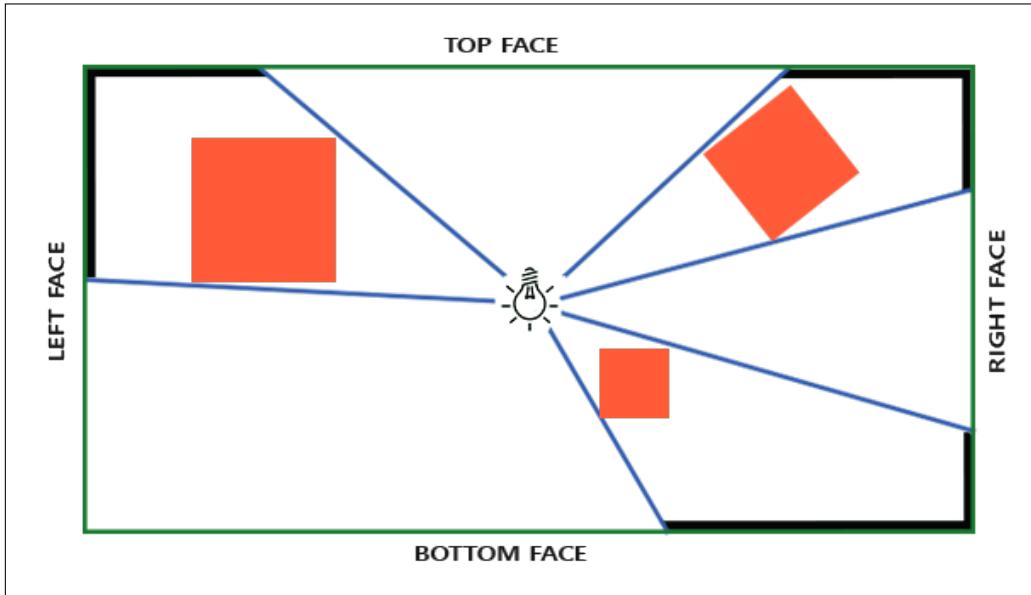
Σχήμα 2.2: Bedroom in OpenGL

Για την σκίαση των αντικειμένων χρειαζόταν να φύγω από το orthographic projection shading που κάναμε στο εργαστήριο καθώς δεν έδινε τα επιθυμητά αποτελέσματα. Οι πίνακες θα άφηναν σκιά μόνο στο πάτωμα παρόλο που η πηγή είναι μία λάμπα στο ταβάνι. Στην αρχή δοκίμασα να αλλάξω το orthographic projection σε perspective projection. Αυτό άφηνε πολλά artifacts και δεν ήταν η σωστή προσέγγιση.



Σχήμα 2.3: Artifacts από το perspective projection

Η σωστή προσέγγιση ήταν η χρήση point shadows [1]. Σε αυτή τη μέθοδο αντί για 1 depth map χρησιμοποιούμε 6, μία για κάθε πλευρά ενός κύβου όπου γίνεται η προβολή. Χρειάζεται αλλαγή στον depth shader για να προστεθεί και ένας geometry shader που κάνει αυτή την μετάφραση σε 6 depth maps. Για να βρούμε την σκίαση κάθε σημείου, παίρνουμε την προβολή του πάνω σε αυτόν τον κύβο.



Σχήμα 2.4: Shadow Mapping Example

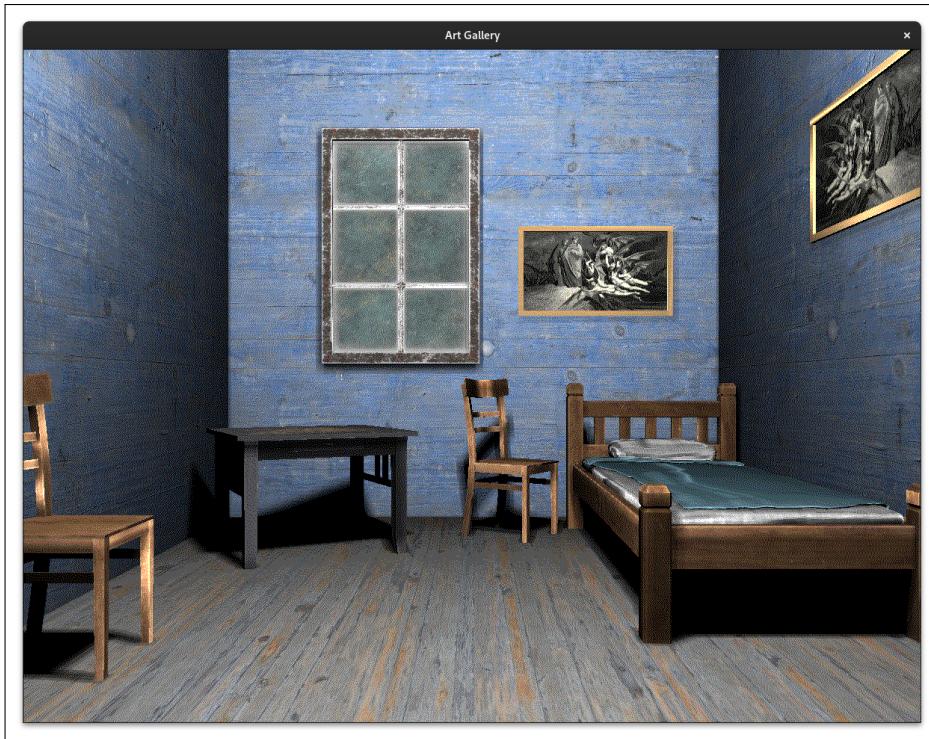
3 FBOs

Η εφαρμογή των διαφορετικών artstyles θα γίνεται με post processing. Τότε, χρειάζεται να αποθηκεύεται το κάθε rendered frame σε ένα texture όπου θα γίνεται τυχόν επεξεργασία και έπειτα θα εμφανίζεται στην οθόνη. Για αυτόν τον σκοπό χρειαζόμαστε ένα sceneFBO όπου έχει στο color attachement ένα texture όπου αποθηκεύεται το rendered frame. Έπειτα, αυτό το texture δίνεται στον default framebuffer και γίνεται rendered πάνω σε ένα plane που καλύπτει όλη την οθόνη. Αυτό το texture μπορεί να υποστεί επεξεργασία στην CPU πριν γίνει rendered ή καθώς γίνεται rendered στους shaders. Για κάθε artstyle υπάρχει και ένας shader που του αντιστοιχεί και χρησιμοποιείται για το rendering του texture στο plane ενώ όπου χρειάζεται υπάρχουν και συναρτήσεις C++ για προεπεξεργασία.

4 ARTSTYLES

4.1 Floyd-Steinberg Dithering

Το Floyd-Steinberg Dithering είναι μία μέθοδος για να μειώσουμε τον αριθμό των χρωμάτων σε μία εικόνα, χωρίς να χάσουμε πολύ ποιότητα. Ο αλγόριθμος αυτός [2], για κάθε pixel της εικόνας προσθέτει το error που προκύπτει από την αποκλιση του χρώματος των προηγούμενων pixels και έπειτα κβαντίζει το χρώμα στην πλησιέστερη τιμή. Με αυτόν τον τρόπο, οι αποκλίσεις του κάθε pixel μοιράζονται σε όλον τον πίνακα και τα σφάλματα κβάντισης είναι δύσκολο να παρατηρηθούν, ειδικά από απόσταση. Δεν χρειάζεται κάποια περεταίρω επεξεργασία οπότε γίνεται rendered στο plane με τον default shader.



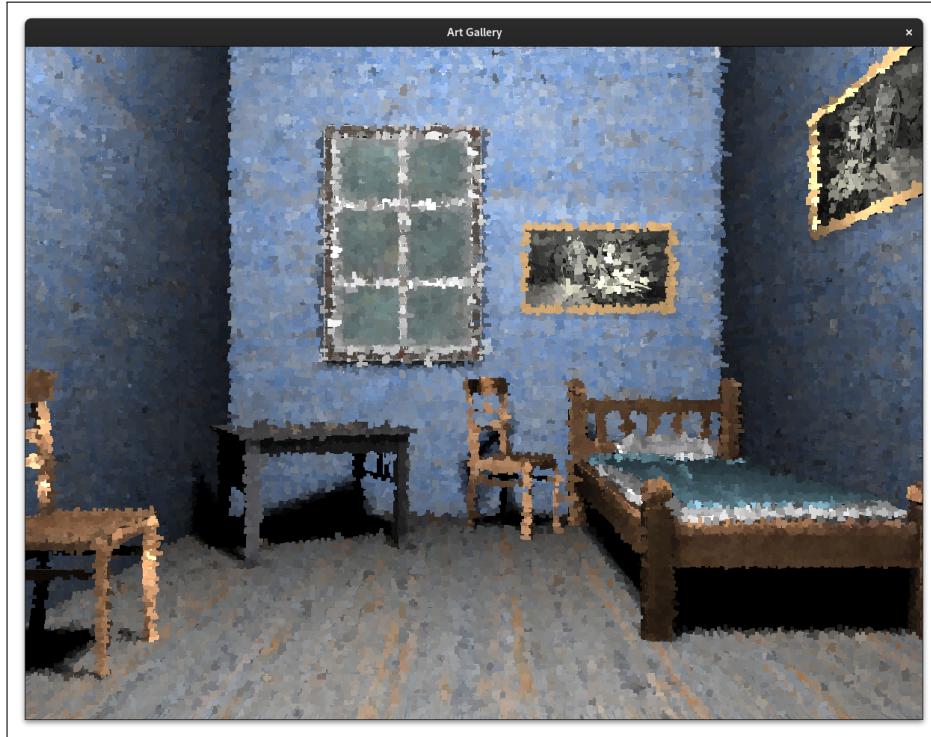
Σχήμα 4.1: Floyd-Steinberg Dithering με 3 χρώματα ανά κανάλι (27 σύνολο)

Επειδή όμως ο αλγόριθμος είναι σειριακός, αφού χρειάζεται τα προηγούμενα pixels για να λειτουργήσει, δεν είναι κατάλληλος για να γίνει στους shaders. Για αυτό, φορτώνεται το texture στην CPU και γίνεται η επεξεργασία εκεί. Επειδή όμως η διαδικασία είναι πολύ αργή, χρειάστηκε πολλές βελτιστοποιήσεις για να είναι ικανοποιητική η ταχύτητα. Παραδείγματα είναι η εκτενή χρήση bitwise operations όπου ήταν δυνατή και η κατάλληλη μέθοδος φόρτωσης-εκφόρτωσης του texture. (Φόρτωση σε GL_BGRA αντί για GL_RGB [3]).

4.2 Painterly Rendering

Το Painterly Rendering είναι μια προσπάθεια να αποδοθεί μια εικόνα με τον τρόπο που θα την ζωγράφιζε ένας ζωγράφος. Συνδιάζει τα ερωτήματα 4.b & 4.c καθώς μοιάζουν μεταξύ τους σε αποτέλεσμα. Αυτή είναι μία ακόμα τεχνική που απαιτεί επεξεργασία στην CPU. Φορτώνουμε το texture στην CPU και εκτελούμε τον αλγόριθμο. [4] Ο αλγόριθμός εφαρμόζεται σε πολλά στάδια με πολλά iterations. Σε κάθε iteration επιλέγεται ένα τυχαίο βήμα για το x και για το y και δημιουργούμε ένα grid από επιλεγμένα pixels. Σε αυτά δίνουμε και ένα τυχαίο offset για να μην μοιάζει τόσο ομοιόμορφο. Seed για τις τυχαίες τιμές είναι οι τιμές χρωμάτων κάποιων pixels ώστε να είναι ίδιο το αποτέλεσμα όσο είναι στάσιμος ο παίχτης αλλά να αλλάζει όσο κουνιέται. Έχουμε επιλέξει τώρα κάποια pixels τα οποία θα λειτουργήσουν ως η βάση του Painterly rendering. Σκοπός μας είναι να μοιράσουμε το χρώμα τους και στα γειτονικά pixels ώστε να φαίνεται σαν να έχουν χρωματιστεί με μία βούρτσα, όλα μαζί. Ανάλογα με την διαφορά χρώματος των γειτονικών pixels επιλέγουμε και το μέγεθος της βούρτσας. Αυτό γιατί

ένας ζωγράφος όπου υπάρχουν μεγάλες εναλλαγές. Χρωματίζουμε τώρα τα pixels γύρω από το επιλεγμένο με το χρώμα του. Δεν χρειάζεται κάποια περεταίρω επεξεργασία οπότε γίνεται rendered στο plane με τον default shader.

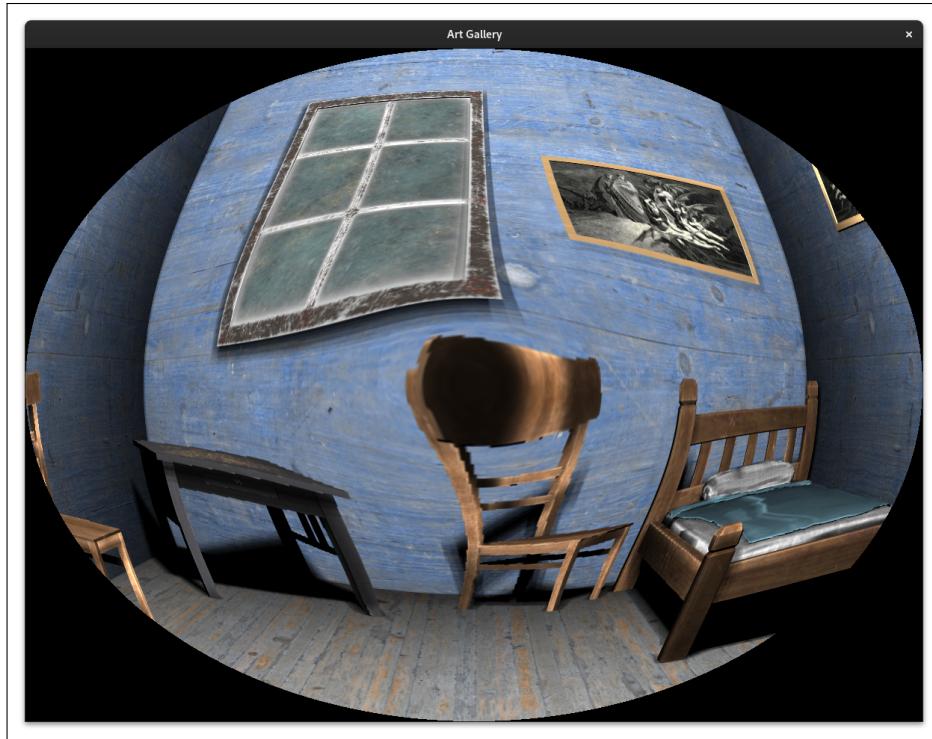


Σχήμα 4.2: Painterly Rendering με 15 iterations

Αυτή η τεχνική είναι ικανοποιητική αλλά στις ακμές των αντικειμένων φαίνεται ότι ξεφεύγει. Ένας πραγματικός ζωγράφος θα ήταν ακόμα πιο προσεκτικός σε εκείνες τις περιοχές. Δοκίμασα, με ένα edge detection να βρω αυτές τις ακμές και να χωρίσω τον πίνακα σε περιοχές με έναν dfs αλγόριθμο. Ο Painterly αλγόριθμος θα επιτρεπόταν να αλλάξει το χρώμα μόνο σε pixels που βρίσκονται στην ίδια περιοχή σαν έναν ζωγράφο που ζωγραφίζει εντός των γραμμών. Αυτή η προσέγγιση έδινε λίγο καλύτερα αποτελέσματα αλλά με μεγάλο υπολογιστικό κόστος που δεν άξιζε τον κόπο.

4.3 Fish Eye Effect

Το fish eye effect είναι ουσιαστικά μια διαστρεύλωση των UV values των pixels στην εικόνα. Θέλουμε τα pixels στο κέντρο να μοιάζουν μεγαλύτερα από αυτά στα άκρα ώστε να πετύχουμε το εφέ ενός κυρτού φακού, όπως τα μάτια ενός ψαριού. Προσθέτουμε ένα offset στις UV του κάθε fragment ανάλογα με μια δύναμη της απόστασης του από το κέντρο. Έτσι τα μακρινά pixels θα διαστρεβλωθούν πολύ ενώ τα κεντρικά λίγο. Επίσης αν ένα pixel είναι πολύ μακριά από το κέντρο, χρωματίζεται μαύρο γιατί θα έπαιρνε τιμές εκτός του (0, 1) με την διαστρεύλωση.



Σχήμα 4.3: Fish Eye Effect

4.4 Toon Shading

Το Toon Shading είναι μια τεχνική σκίασης που μοιάζει με κόμικς. Αντί για ομαλές μεταβάσεις χρωμάτων, έχουμε έντονες αλλαγές μεταξύ grayscale τιμών. Επίσης, δίνεται έμφαση στα περιγράμματα βάφοντας τα με μαύρο που δίνει την εντύπωση ότι είναι σχεδιασμένα με μολύβι. Για το edge detection χρησιμοποιήθηκαν τα Sobel filters [5]

$$sx = \begin{bmatrix} 1.0 & 2.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \\ -1.0 & -2.0 & -1.0 \end{bmatrix} \quad sy = \begin{bmatrix} 1.0 & 0.0 & -1.0 \\ 2.0 & 0.0 & -2.0 \\ 1.0 & 0.0 & -1.0 \end{bmatrix}$$

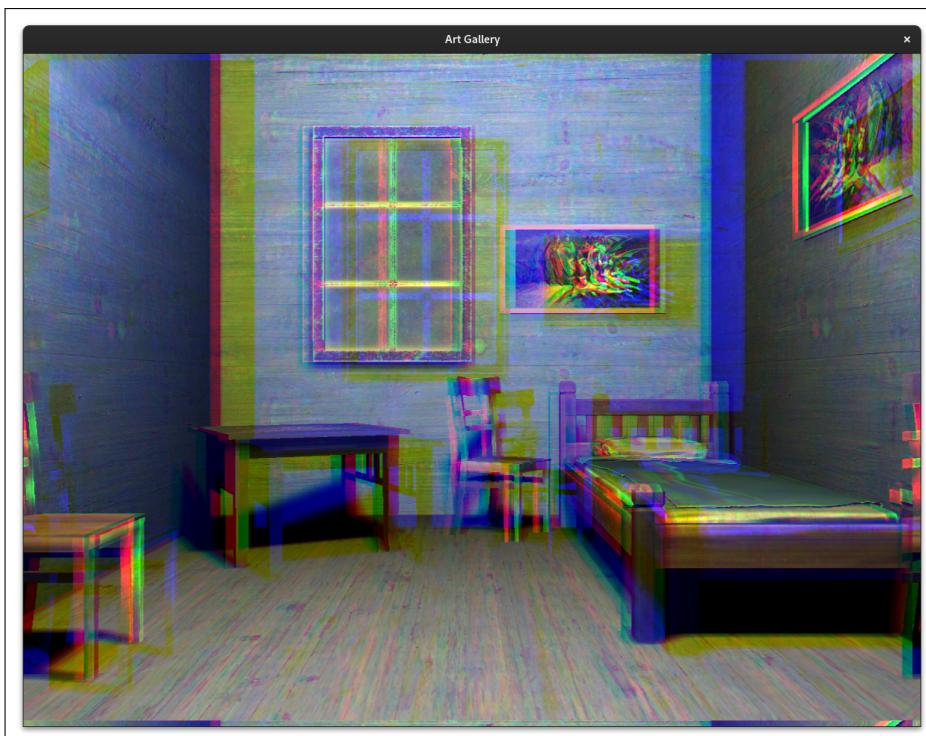
Το Gradient Magnitude το περνάμε μετά από μία smoothscale function για να μειωθεί ο θόρυβος. Κάθε fragment που είναι πάνω από το threshold για edge χρωματίζεται μαύρο. Τα υπόλοιπα κβαντίζονται σε 6 grayscale τιμές.

4.5 Chromatic Aberration

Με το Chromatic Aberration αποδίδεται ένα trippy εφέ στην εικόνα. Τα αντικείμενα φαίνονται διπλά και μοιάζουν να κουνιούνται αλλά και να είναι στάσιμα ταυτόχρονα. Αυτό επιτυγχάνεται παίρνοντας διαφορετικά UV values για κάθε κανάλι χρώματος. Το κάθε κανάλι αντιστοιχείται σε μία περιοδική συνάρτηση με διαφορετική φάση και για αυτό φαίνονται τα αντικείμενα να κουνιούνται. Το εφέ είναι καλύτερο και πιο ξεκάθαρο σε βίντεο.



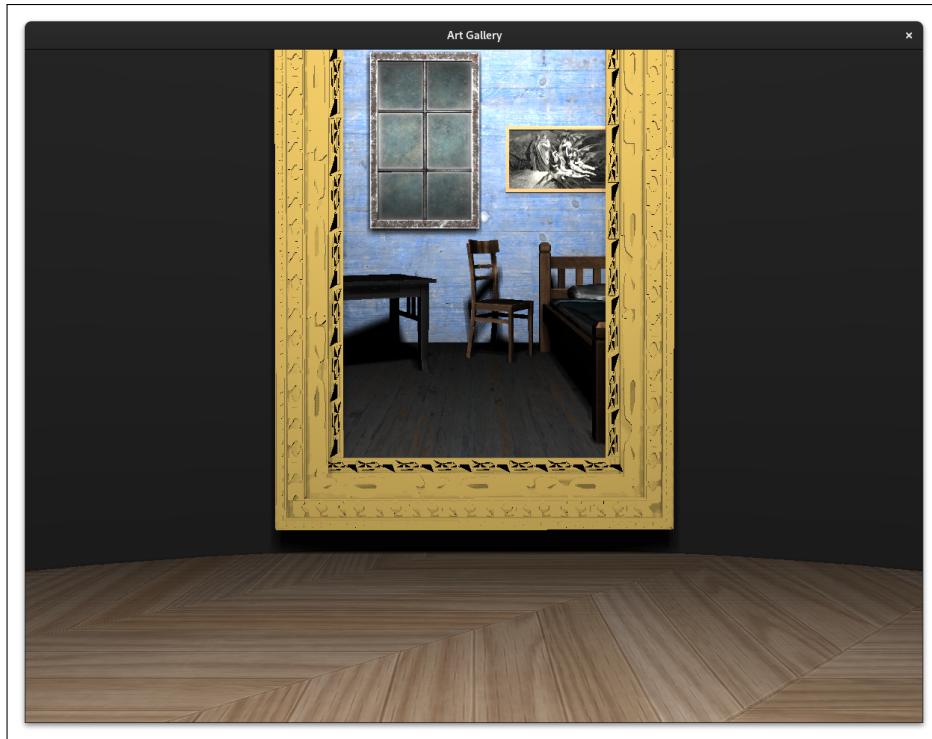
Σχήμα 4.4: Toon Shading



Σχήμα 4.5: Chromatic Aberration

5 BUMP RENDERING

Προς το παρόν, οι πίνακες είναι ένα μαύρο επίπεδο. Θέλουμε να είναι ένα preview του δωματίου που τους αντιστοιχεί. Χρειαζόμαστε δηλαδή ένα texture που να απεικονίζει την σκηνή σε κάθε δωμάτιο. Πριν το mainloop τότε κάνουμε ένα render pass για κάθε δωμάτιο και αποθηκεύουμε την σκηνή σε ένα texture. Για αυτό ωστε χρειαστούμε ένα νέο FBO στο οποίο ωστε μπορούμε να αλλάξουμε το color attachment texture κάθε φορά. Το texture μπορούμε να το επεξεργαστούμε κανονικά, όπως στο mainloop για να φαίνεται και το εφέ του κάθε δωματίου. Έπειτα, αυτό το texture ωστε αποθηκεύεται ως το diffuse texture κάθε πίνακα.



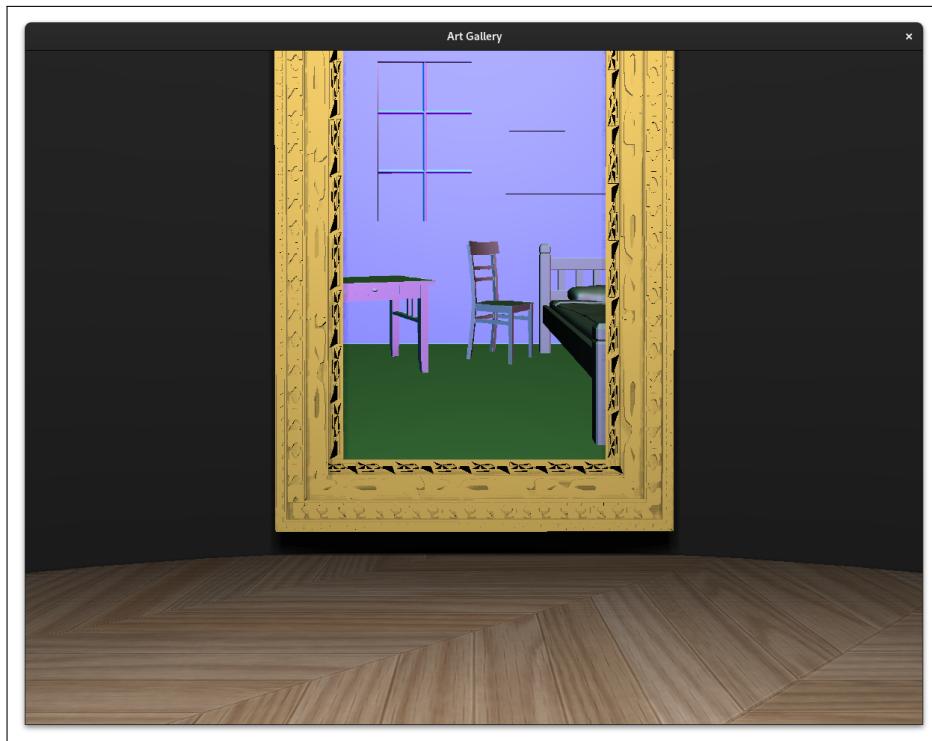
Σχήμα 5.1: Preview of the deafult room

Για να αποδώσουμε τις λεπτομέρειες των επιφανειών, μπορούμε να αποθηκεύουμε και τα normals της σκηνής σε ένα bump texture και να χρησιμοποιούμε τα normals από εκεί στον fragment shader αντί για την παρεμβολή των normals των vertices που μας περιορίζουν αρκετά. Χρειαζόμαστε έναν νέο FBO με μεταβλητό color attachment texture στο οποίο αντί για το χρώμα της σκηνής ωστε αποθηκεύουμε το normal του κάθε fragment.

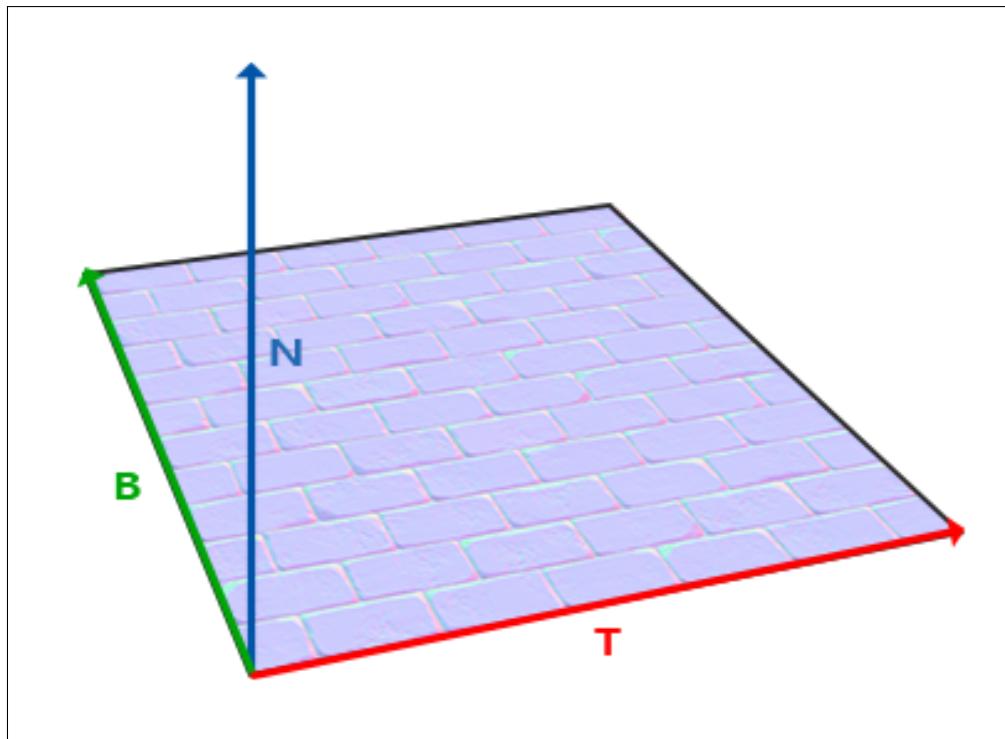
$$normal.x -> color.r | normal.y -> color.g | normal.z -> color.b$$

Επειδή όμως τα normals παίρνουν τιμές $(-1, +1)$ ενώ τα rgb τιμές παίρνουν $(0, 1)$ κάνουμε scale το normal κατά 0.5 και προσθέτουμε 0.5 ώστε να ταυτίζονται οι περιοχές τιμών. Όταν τα πάρουμε από το texture αντιστρέφουμε την διαδικασία.

Τα normal maps όμως απεικονίζουν τα normals του αντικειμένου μόνο σε ένα συγκεκριμένο orientation. Όταν το αντικείμενο περιστραφεί, τα normals από το normal map δίνουν πολύ λανθασμένα αποτελέσματα. Για την χρήση αυτών των normals χρειάζεται να μεταφέρουμε τους υπολογισμούς του phong shading από το world space στο tangent space ώστε να μην μας νοιάζει η περιστροφή του αντικειμένου [6]. Για κάθε τρίγωνο κάθε αντικειμένου χρειάζεται να υπλογίσουμε ένα TBN matrix που ορίζεται από τα normals του και τις UV συνιστώσες του. Αυτό δημιουργεί ένα νέο σύστημα συντεταγμένων όπου ωστε δουλέψουμε.



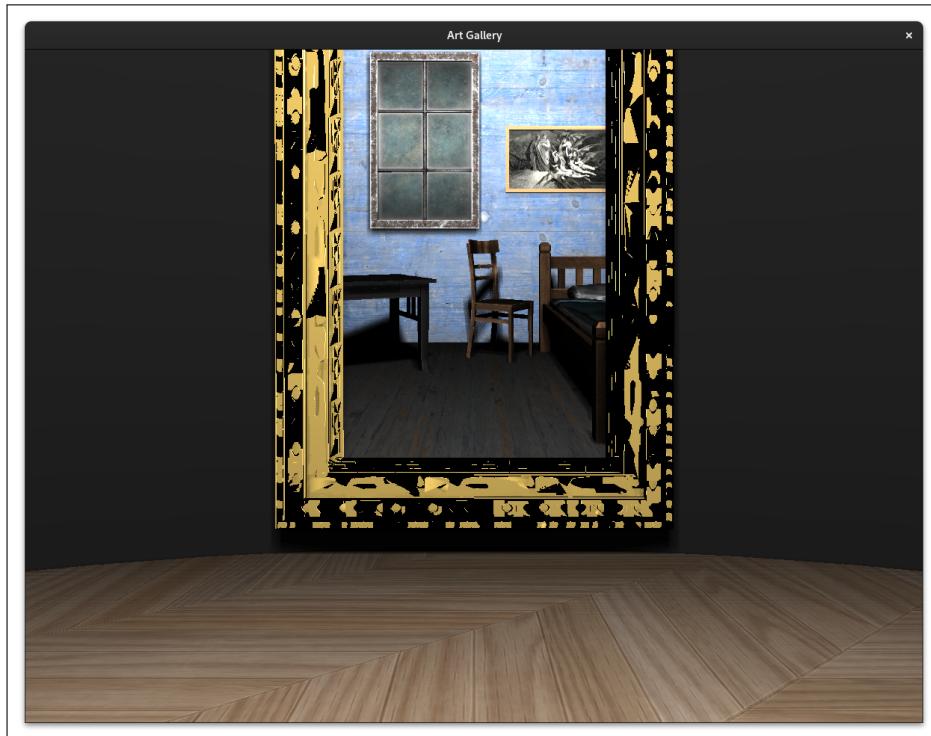
Σχήμα 5.2: Normal Map της σκηνής αν το βάλουμε ως diffuse texture



Σχήμα 5.3: Tangent space coordinates

Μεταφέρουμε τις μεταβλητές που χρειάζονται για το Phong shading στο νέο σύστημα συντεταγμένων πολλαπλασιάζοντας με το inverse transpose TBN matrix. Χρειαζόμαστε την θέση του fragment, την θέση του φωτός και την θέση του παρατηρητή. Έπειτα μπορούμε να κάνουμε τον υπολογισμούς όπως συνήθως. Επειδή δεν είχαν όλα τα αντικείμενα normal maps, με ένα flag αποφασίζει ο shader αν θα χρησιμοποιήσει το normal map ή τα normals του αντικειμένου.

Οι κορνίζες των πινάκων όμως είχαν μια δυσκολία στον υπολογισμό του TBN matrix. Για ορισμένα τρίγωνα οι υπολογισμοί του tangent vector και του bitangent vector έβγαιναν nan ή inf επειδή 2 κορυφές του είχαν τις ίδιες UV συντεταγμένες.



Σχήμα 5.4: Nan και inf τιμές στον υπολογισμό του TBN matrix

Για αυτό χρειάστηκε να θεσω ένα ακόμα flag αν θα κάνει ο shader τους υπολογισμούς στο world space αντί για το tangent space.

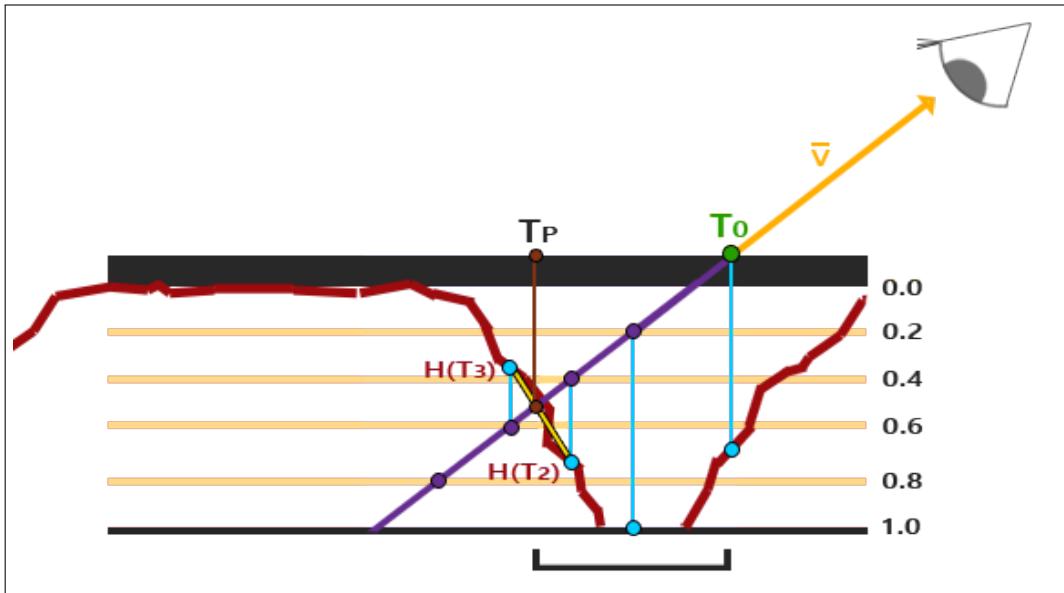
Για τον πίνακα με το fish eye effect έπρεπε να διεστρευλώσω και το normal map ώστε να υπάρχει σωστή αντιστοίχηση με το diffuse texture.

6 PARALLAX MAPPING

Για μεγαλύτερες λεπτομέρειες στις επιφάνειες χρησιμοποιείται κάποια μέθοδος **displacement mapping**. Με το parallax mapping [7] μπορούμε να προσομοιώσουμε γεωμετρικές λεπτομέρειες χωρίς όμως να αλλάζουμε το mesh του αντικειμένου.

Αρχικά αποθηκεύουμε πληροφορίες για το βάθος του κάθε αντικειμένου της σκηνής σε ένα displacement texture κάνοντας άλλο ένα render pass πριν το mainloop με ένα διαφορετικό FBO όπου στο color attachment αποθηκεύει την Z συνιστώσα του κάθε fragment. Έπειτα μπορούμε να χρησιμοποιήσουμε αυτό το texture για να δώσουμε στους πίνακες ένα αίσθημα βάθους που δεν υπάρχει.

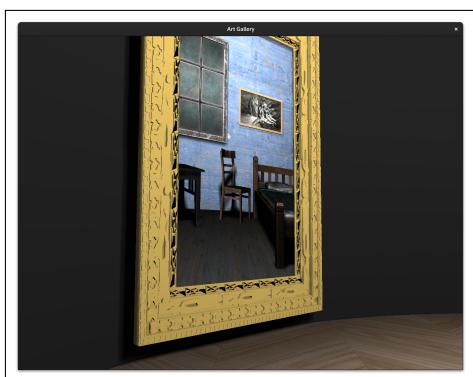
Η βασική αρχή του parallax mapping είναι η αλλαγή των UV coordinates ανάλογα με το βάθος του texture σε εκείνη την περιοχή. Δίνουμε την φυσικότητα του βάθους υπολογίζοντας άμα το αντικείμενο ήταν όντως ανάγλυφο, ποιο σημείο θα έβλεπε ο παρατηρητής. Στο κάθε fragment



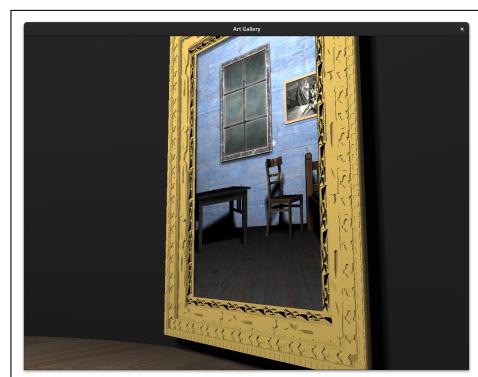
Σχήμα 6.1: Parallax Occlusion Mapping

βλέπουμε από το displacement texture το βάθος σε εκείνο το σημείο. Προεκτίνουμε το view direction vector όσο είναι το βάθος ή φτάσουμε σε ένα threshold. Αυτό είναι το νέο UV του fragment που θα χρησιμοποιηθεί. Επαναλαμβάνουμε αυτή την διαδικασία μέχρι το view vector να φτάσει κάτω από το depth value του UV. Τότε θεωρούμε UV για αυτό το fragment τον μέσο όρο των 2 τελευταίων UVs και το χρησιμοποιούμε για το texture lookup.

Με αυτήν την μέθοδο πετυχαίνουμε το εφέ του βάθους. Κοιτώντας από διαφορετικές γωνίες βλέπουμε τα αντικείμενα να κουνιούνται και πράγματα που δεν ήταν εμφανά στην αρχή γιατί τα μπλόκαρε κάτι άλλο να είναι ορατά. Σε βίντεο είναι πιο εμφανές το εφέ.



Σχήμα 6.2: Looking from the left



Σχήμα 6.3: Looking from the right

REFERENCES

- [1] LearnOpenGL Joey de Vries. Point shadows - advanced lighting | learnopengl. <https://learnopengl.com/Advanced-Lighting/Shadows/Point-Shadows>.
- [2] Wikipedia. Floyd-steinberg dithering - wikipedia. https://en.wikipedia.org/wiki/Floyd%20Steinberg_dithering.
- [3] User mhagain. Answer on khronos forum. <https://community.khronos.org/t/updating-textures-per-frame/75020/3>.
- [4] Amelia Carolina Sparavignal and Roberto Marazzato. Non-photorealistic image processing: an impressionist rendering. <https://arxiv.org/pdf/0911.4874.pdf>.
- [5] Wikipedia. Sobel operator - wikipedia. https://en.wikipedia.org/wiki/Sobel_operator.
- [6] LearnOpenGL Joey de Vries. Normal mapping | learnopengl. <https://learnopengl.com/Advanced-Lighting/Normal-Mapping>.
- [7] LearnOpenGL Joey de Vries. Parallax mapping | learnopengl. <https://learnopengl.com/Advanced-Lighting/Parallax-Mapping>.