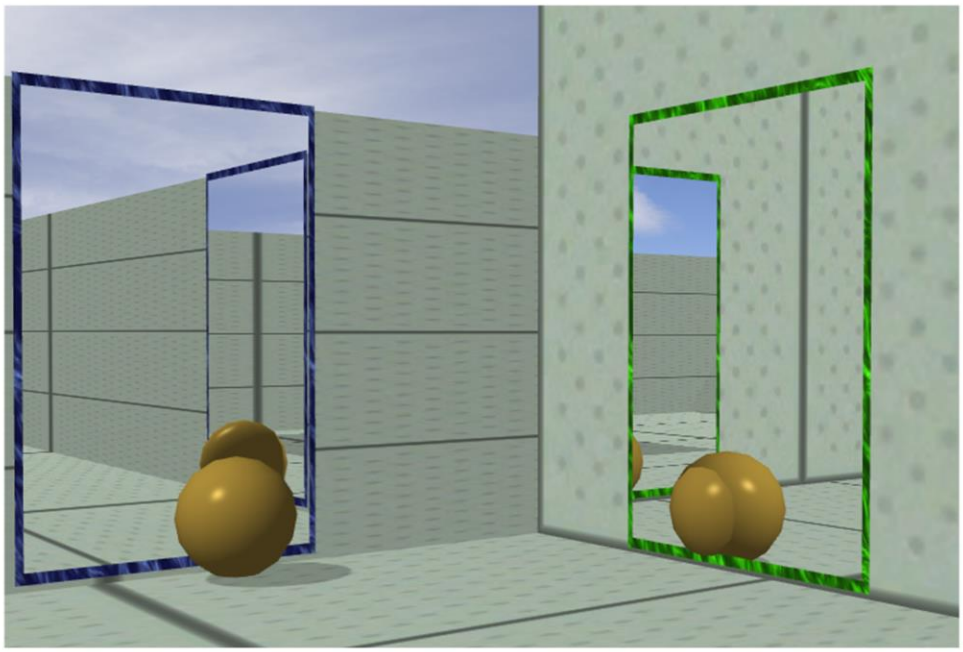


Απαλλακτική εργασία

PORTALS I

ΓΙΩΡΓΟΣ ΠΕΤΡΑΚΗΣ | ΑΜ: 1066516 | Γραφικά και Εικονική Πραγματικότητα | 13/01/2022

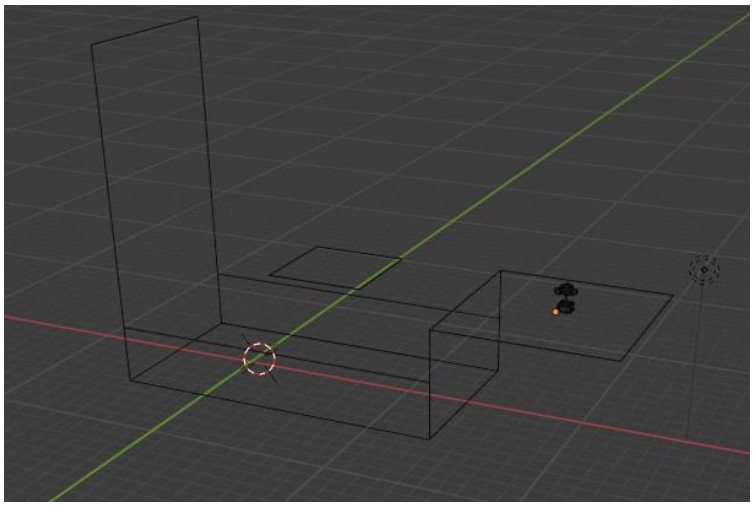
ΠΕΡΙΕΧΟΜΕΝΑ

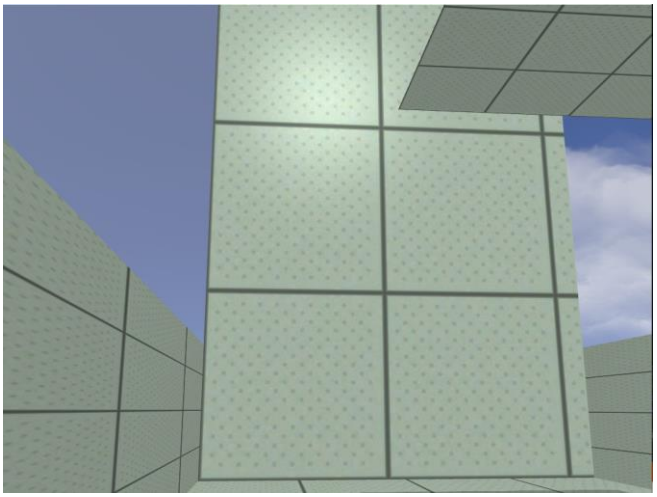
Σε κάθε μέρος της εργασίας γίνεται αναφορά τόσο στις δυσκολίες που αντιμετωπίστηκαν όσο και σε “bugs” που παρατηρήθηκαν και δεν κατάφερα να διορθώσω. Επίσης, παρατίθενται και προτάσεις βελτίωσης, οι οποίες λόγω πολυπλοκότητας και περιορισμού χρόνου, δεν συμπεριλήφθηκαν στο τελικό πρόγραμμα.

ΜΕΡΟΣ Α

1. Σκηνή και αντικείμενα

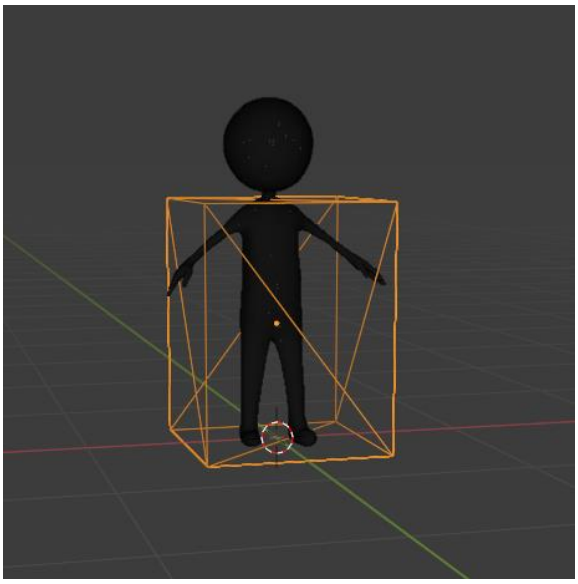
Για την δημιουργία της σκηνής χρησιμοποιήθηκε το λογισμικό “Blender”. Η σκηνή αποτελείται από 2 επίπεδα, 4 τοίχους, εκ των οποίων ο ένας είναι ψηλότερος, μια οροφή και ένα «τρόπαιο». Θεωρητικά, ο σκοπός του «παιχνιδιού» είναι με την χρήση των Portals ο παίκτης να ανέβει στο 2ο επίπεδο όπου βρίσκεται το βραβείο. Το UV-Mapping της σκηνής έγινε στο Blender με JPG και BMP textures από το παιχνίδι της Valve “Portal 1”. Παρακάτω φαίνεται η δομή της και ένα screenshot από το πρόγραμμα:





Για την φόρτωση της σκηνής στην OpenGL, δημιουργήθηκε μια νέα κλάση “Scene” στο πρόγραμμα, η οποία φορτώνει με την κλάση Drawable κάθε τοίχο της σκηνής ξεχωριστά, ώστε να μπορούμε αργότερα να εφαρμόσουμε collision detection και ray casting για την τοποθέτηση των portals. Η σκηνή είναι ανοιχτή από πάνω για να εφαρμόσουμε directional lighting και εκτός από την σκηνή κάνουμε render και ένα skybox.

Πέρα από την σκηνή και το τρόπαιο δεν φορτώνονται άλλα αντικείμενα εκτός από το μοντέλο του παίχτη και μία σφαίρα την οποία αυτός εκτοξεύει (μοντέλο “Sphere.obj”). Το μοντέλο του παίχτη είναι το stickman figure που φαίνεται στο παρακάτω screenshot.



Το «κουτί» που περιβάλλει τον παίχτη στην εικόνα (“Cube.obj” από lab7 με μερικές τροποποιήσεις) χρησιμοποιείται για το collision detection του παίχτη και φυσικά δεν γίνεται render. Θα γίνει συγκεκριμένη αναφορά σε αυτό στο κομμάτι των collisions.

2. Φωτισμός και Σκίαση

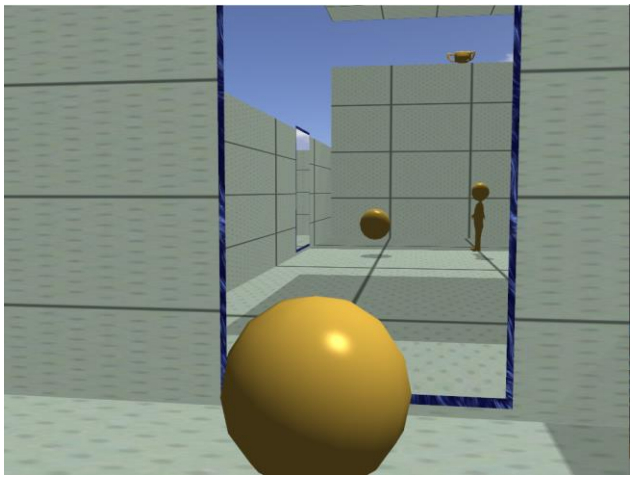
Ο φωτισμός και η σκίαση της σκηνής, του παίχτη και των αντικειμένων, γίνεται με παρόμοιο τρόπο όπως στα εργαστήρια 5 και 6.

Για τον φωτισμό χρησιμοποιήθηκε η κλάση “Light” από το εργαστήριο 5, αλλά ουσιαστικά μόνο για την δημιουργία του viewMatrix του φωτός. Χρησιμοποιείται ένα φως στη θέση (16,20,0), που φαίνεται και στην αρχική φωτογραφία της σκηνής από το blender, ώστε να έχουμε σκίαση και από τοίχους εκτός από αντικείμενα. Εκτός από την συνάρτηση phong() που χρησιμοποιούταν στο εργαστήριο έχουν δημιουργηθεί και μερικές ακόμα για να μπορούμε να έχουμε φωτισμό και σκίαση πάνω σε UV textures.

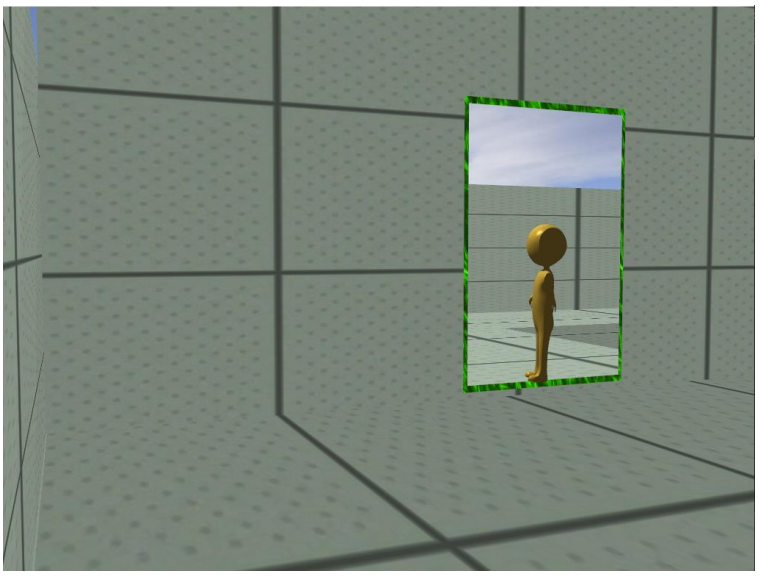
Ομοίως για την σκίαση χρησιμοποιήθηκε ο κώδικας του εργαστηρίου 6 με ελάχιστες παραλλαγές. Χρησιμοποιούμε ένα framebuffer object στο οποίο αποθηκεύουμε την απόσταση κάθε fragment από το φως με ένα νέο shader program “depthProgram” . Με την συνάρτηση ShadowCalculations() στους αρχικούς shaders καθορίζουμε για κάθε fragment ξεχωριστά αν φωτίζεται ή όχι και χρωματίζουμε κατάλληλα. Στον υπολογισμό χρησιμοποιούμε bias για την αντιμετώπιση του “shadow acne” και PCF για να έχουμε πιο «ομαλή» σκίαση. Με ένα depth texture

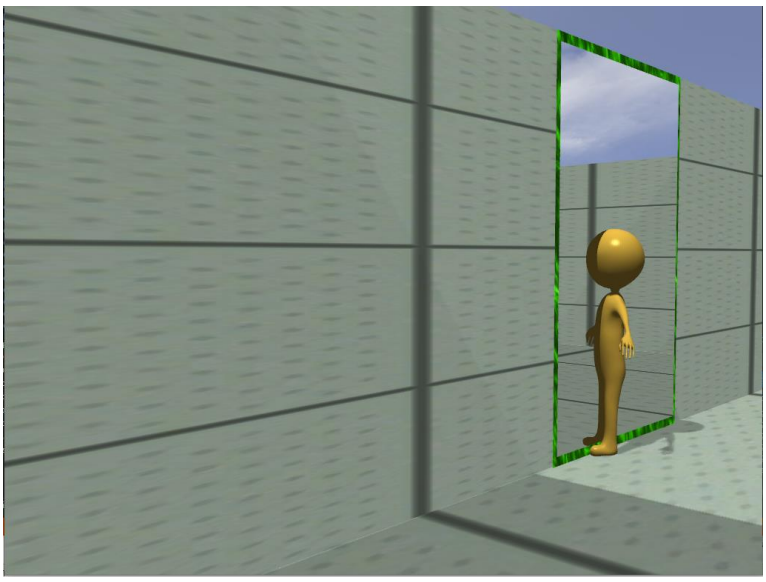
μεγέθους 4056x4056 έχουμε ικανοποιητικά αποτελέσματα χωρίς performance issues.

Μέσα στην συνάρτηση createDepthTexture(), εκτός από το rendering της σφαίρας και του παίχτη έχουμε δύο επιπλέον ελέγχους που εκτελούν ακόμα δύο renders. Η χρήση τους θα γίνει καλύτερα κατανοητή μετά την ανάλυση της λειτουργίας του “Smooth Portal”.



Η χρήση του depth texture είναι απλή. Σε κάθε επανάληψη της mainLoop() επαναδημιουργούμε το texture και το «περνάμε» στους shaders με χρήση του GL_TEXTURE5 της OpenGL, για να μην υπάρχει πρόβλημα με τα υπόλοιπα textures. Όπως φαίνεται στο στιγμιότυπο, λόγω του τρόπου που σχεδιάζουμε το Portal Image (θα αναλυθεί στο 2ο μέρος) οι σκιές εμφανίζονται σωστά και μέσα από τα Portals.





Στις 2 παραπάνω φωτογραφίες έχουμε «τηλεμεταφορά» ενός μέρους του παίχτη από μέρος της σκηνής που φωτίζεται σε μέρος που είναι σκιασμένο (αριστερά) και αντίστροφα (δεξιά). Στην πρώτη φωτογραφία το κομμάτι του παίχτη που εισέρχεται σε μία περιοχή που δε φωτίζεται φαίνεται να σκιάζεται. Στην δεύτερη, αντίστοιχα το κομμάτι που τηλεμεταφέρεται φωτίζεται αλλά και δημιουργεί σκιά στο πάτωμα από κάτω του.

Παρατηρήσεις και προβλήματα πάνω στη σκίαση

Το κυριότερο πρόβλημα που εμφανίστηκε κατά τη σκίαση είναι ότι για ορισμένες γωνίες του φωτός, οι σκιές των τοίχων της σκηνής εμφανίζονται μετατοπισμένες, σαν να αιωρούνται. Μετά από λίγη έρευνα [1], διαπίστωσα ότι το φαινόμενο αυτό ονομάζεται “Peter Panning” και οφείλεται στο bias που χρησιμοποιούμε για να αντιμετωπίσουμε το “shadow acne”. Δυστυχώς, ο ευκολότερος τρόπος να αντιμετωπιστεί αυτό το πρόβλημα είναι να αποφευχθεί η χρήση «δισδιάστατων» αντικειμένων όπως (οι τοίχοι στην συγκεκριμένη σκηνή) και να κάνουμε render μόνο τα back faces των αντικειμένων στο depth map. Όμως, η χρήση «συμπαγών» τοίχων θα δημιουργούσε πρόβλημα σε αρκετές λειτουργίες των Portals, όπως θα εξηγηθεί αργότερα. Για την καλύτερη δυνατή αντιμετώπιση του προβλήματος προσάρμοσα όσο το δυνατόν καλύτερα τα clipping planes του projection matrix του φωτός στη σκηνή, για να βελτιώσω την ακρίβεια του depth texture.

3. Ανίχνευση και χειρισμός συγκρούσεων

Για τις συγκρούσεις που έχουν να κάνουν με την αλληλεπίδραση των αντικειμένων ή του παίχτη με τα Portals θα γίνει σύντομη αναφορά στις μεθόδους ανίχνευσης και ο χειρισμός τους θα αναλυθεί στο 2^{ο} μέρος της αναφοράς. Οι περισσότερες συναρτήσεις για την ανίχνευση και τον χειρισμό συγκρούσεων βρίσκονται στο αρχείο “Collision.cpp” με αντίστοιχα ονόματα.

Συγκρούσεις με Axis Aligned Bounding Boxes (AABBs):

Χρησιμοποιούνται σε κάθε σύγκρουση που περιλαμβάνει τον παίχτη, την σφαίρα και τα Portals. Για τον λόγο αυτό έχει οριστεί ένα AABB struct στην κλάση Rigidbody που περιλαμβάνει min και max για τον κάθε άξονα αλλά και δύο τρισδιάστατα διανύσματα, εκ των οποίων το ένα είναι το κέντρο του

κουτιού και το δεύτερο τα θετικά extents σε κάθε άξονα. Τα διανύσματα αυτά μπορούν να υπολογιστούν από τα min και max.

Το μεγαλύτερο μέρος των collisions περιλαμβάνει AABBs. Οι αλγόριθμοι τους είναι τυποποιημένοι και απλοί. Συγκεκριμένα:

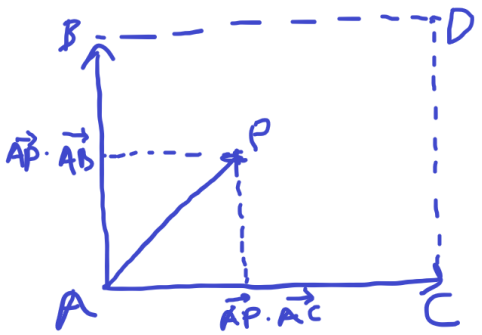
- **AABB – Sphere**: Για συγκρούσεις μεταξύ σφαίρας και Portal. Ο αλγόριθμος που χρησιμοποιούμε υπολογίζει την απόσταση του κέντρου της σφαίρας από το κοντινότερο σημείο του AABB. Αν αυτή είναι μικρότερη από την ακτίνα της σφαίρας τότε έχουμε σύγκρουση.
- **AABB – AABB**: Για τις συγκρούσεις του παίχτη με το Portal. Απλώς ελέγχουμε αν «επικαλύπτονται» τα min,max των δύο κουτιών.

Οι δύο παραπάνω έλεγχοι επηρεάζουν και πολλές από τις υπόλοιπες συγκρούσεις, επειδή για παράδειγμα θέλουμε τα αντικείμενα να αγνοούν συγκρούσεις με τους τοίχους της σκηνής αν συγκρούονται ταυτόχρονα και με το Portal

- **AABB – Plane Intersection**: Για τις «συγκρούσεις» του παίχτη με τους τοίχους και τα πατώματα της σκηνής. Εδώ χρησιμοποιούμε το διάνυσμα του κέντρου και των extents του bounding box. Υπολογίζουμε το διάνυσμα της κάθετης απόστασης κέντρου-plane με προβολή πάνω στο planeNormal. Έπειτα συγκρίνουμε με την προβολή των extents κατά το planeNormal.

Ένα σημαντικό πρόβλημα που εμφανίζεται σε κάθε σύγκρουση αντικειμένου με τους τοίχους της σκηνής είναι ότι στην πραγματικότητα οι συγκρούσεις δεν γίνονται με «άπειρα» planes, αλλά με rectangles. Για τον λόγο αυτό, μετά από την ανίχνευση της σύγκρουσης με το επίπεδο του τοίχου, πρέπει να γίνει ένας επιπλέον έλεγχος για να επιβεβαιώσουμε ότι το αντικείμενο βρίσκεται μέσα στα «όρια» του τοίχου. Για να εκτελέσουμε τον έλεγχο αυτό αρχικά πρέπει να προβάλουμε το κέντρο του αντικειμένου στο επίπεδο του τοίχου. Αυτό γίνεται αφαιρώντας την συνιστώσα του διανύσματος θέσης του κατά την διεύθυνση του normal του επιπέδου. Έπειτα πρέπει να ελέγξουμε αν το νέο σημείο που βρήκαμε βρίσκεται μέσα στο rectangle. Με σκοπό να δημιουργηθεί ένας κώδικας γενικής χρήσης που θα λειτουργούσε για τοίχο οποιουδήποτε προσανατολισμού δοκιμάστηκε η χρήση διανυσματικής άλγεβρας. Συγκεκριμένα, αν είχαμε έναν τοίχο με vertices ABCD και την προβολή P του κέντρου του αντικειμένου πάνω

στο επίπεδο του τοίχου, οι προβολές του διανύσματος AP σε δύο κάθετες πλευρές (dot(AP,AB) και dot(AP,AC)) πρέπει να είναι θετικές και μικρότερες από τα αντίστοιχα scalar products των πλευρών AB, AC του rectangle ώστε το σημείο P να βρίσκεται εντός του τοίχου.



Δυστυχώς, ο αλγόριθμος αυτός δεν είχε πάντα αξιόπιστα αποτελέσματα, αφού πολλές φορές ανιχνευόταν σύγκρουση εκτός των ορίων του τοίχου, και δεν κατάφερα να τον διορθώσω. Για τον λόγο αυτό, περιορίστηκα σε τοίχους που είναι προσανατολισμένοι στους άξονες x, y, z και χρησιμοποίησα απλές συγκρίσεις συντεταγμένων σημείων. Παρόλα αυτά, ο αρχικός κώδικας με τα διανύσματα παρατίθεται υπό μορφή σχολίων σε μερικές συναρτήσεις.

<u>Λοιπές συγκρούσεις:</u>

- **Sphere-Plane**: Για τις συγκρούσεις της σφαίρας με τους τοίχους της σκηνής και με το Portal Plane. Ο αλγόριθμος είναι παρόμοιας λογικής με τον AABB-Plane collision, δηλαδή συγκρίνουμε απλώς την κάθετη απόσταση του κέντρου της σφαίρας από το επίπεδο με την ακτίνα της σφαίρας. Ο έλεγχος για τα όρια του τοίχου πραγματοποιείται και εδώ, ώστε η σφαίρα να μπορεί για παράδειγμα να πέσει «έξω» από την σκηνή αν ξεφύγει από τα όρια των τοίχων ή του πατώματος.
- **Line-Rectangle Intersection**: Για να ελέγξουμε αν η κάμερα του παίχτη περνάει το επίπεδο του Portal, ώστε να γίνει «τηλεμεταφορά» πριν τον σχεδιασμό του επόμενου καρέ. Αρχικά ελέγχουμε απλώς αν η ευθεία που ορίζεται από την τωρινή και την επόμενη θέση της κάμερας τέμνει το επίπεδο του Portal. Λύνοντας ένα σύστημα παραμέτρων που προκύπτει από τις διανυσματικές εξισώσεις του

επιπέδου και της ευθείας [2] μπορούμε να ελέγξουμε και αν η κάμερα βρίσκεται μέσα στα όρια του Portal.

- **Ray Casting σε Plane**: Για την τοποθέτηση των Portals με το ποντίκι και το πληκτρολόγιο (πλήκτρα Q και E) εφαρμόζεται Ray Casting, σύμφωνα με τον αλγόριθμο [3]. Ο αλγόριθμος αυτός λαμβάνει την θέση του ποντικιού στην οθόνη, την μετατρέπει σε συντεταγμένες κάμερας και έπειτα σε συντεταγμένες κόσμου με χρήση των αντίστροφων των πινάκων projectionMatrix και viewMatrix. Έπειτα το πρόβλημα εκφυλίζεται σε έναν απλό έλεγχο line-plane intersection. Η συνάρτηση που χρησιμοποιείται επιστρέφει το σημείο τομής, αν υπάρχει, το οποίο ελέγχεται έπειτα αν βρίσκεται μέσα στα όρια του τοίχου. Αν βρίσκεται, το portal τοποθετείται ελάχιστα μετατοπισμένο κατά το normal διάνυσμα του τοίχου, για να μην έχουμε επικάλυψη των fragments κατά το rendering, και με προσανατολισμό ίδιο με το normal του τοίχου.

Χειρισμός συγκρούσεων και Physics

Για την εφαρμογή φυσικών ιδιοτήτων και βαρύτητας στα αντικείμενα χρησιμοποιήθηκε η κλάση Rigidbody του 7^{ου} εργαστηρίου τόσο στον παίχτη όσο και στην σφαίρα. Όσον αφορά την σφαίρα, η λογική χειρισμού των συγκρούσεων είναι παρόμοια με εκείνη του εργαστηρίου. Τα αποτελέσματα στη σφαίρα είναι ιδιαίτερα ικανοποιητικά, οι κινήσεις και οι συγκρούσεις γίνονται ομαλά και δεν υπάρχουν ενδείξεις για ώθηση του συστήματος σε αστάθεια.

Παρόλα αυτά, δεν μπορούμε να εφαρμόσουμε την ίδια λογική και για τις συγκρούσεις του παίχτη. Αν εφαρμόσουμε διόρθωση θέσης στην σύγκρουση παίχτη-τοίχου/πατώματος παρατηρούμε ότι η κάμερα (η οποία κινείται συνεχώς σε συνάρτηση με τον παίχτη) εκτελεί απότομες κινήσεις κατά τη διόρθωση θέσης και το αποτέλεσμα δεν είναι καθόλου ικανοποιητικό. Για να διορθώσουμε το φαινόμενο αυτό, χρειάστηκε να εφαρμοστεί έλεγχος ίδιας λογικής με αυτόν της κάμερας-Portal, και επιπλέον μερικά άλλα τεχνάσματα για να φαίνεται ότι ο παίχτης κινείται σύμφωνα με τις αρχές της φυσικής. Η σύγκρουση του παίχτη με τον τοίχο αντιμετωπίζεται με αποθήκευση της παλιάς θέσης του και έλεγχο για σύγκρουση στην επόμενη. Αν ανιχνεύεται σύγκρουση στην επόμενη θέση τότε ο παίχτης ουσιαστικά ακινητοποιείται στην τωρινή του θέση.

Παρατηρήσεις και προτάσεις βελτίωσης

- Αξίζει να γίνει αναφορά στον τρόπο δημιουργίας του AABB του παίχτη. Ο «σωστός» τρόπος να δημιουργηθεί το AABB του παίχτη είναι με ένα βρόχο επανάληψης να διαπεραστεί ο πίνακας των vertices (σε συντεταγμένες κόσμου, δηλαδή σε κάθε καρέ πρέπει να εκτελούμε πολλαπλασιασμό με τον modelMatrix) και να σημειώσουμε τις ελάχιστες και μέγιστες συντεταγμένες. Αυτός ο τρόπος δοκιμάστηκε αρχικά, όμως, λόγω του μεγάλου αριθμού των vertices είχε σοβαρές επιπτώσεις στην επίδοση του προγράμματος και προκαλούσε frame drops. Για την διόρθωση του προβλήματος δημιουργήθηκε ένα κουτί γύρω από τον παίχτη, του οποίου τα vertices πολλαπλασιάζονται με τον ίδιο modelMatrix. Έτσι το AABB υπολογίζεται με έλεγχο σε αυτά τα vertices αντί των 5000 που περιέχει το αρχικό μοντέλο και η απόδοση βελτιώνεται σημαντικά. Για να «καλύπτει» σωστά τις διαστάσεις του παίχτη θα ήταν πιο λογικό το κουτί να είναι πιο «στενό» από ότι φαίνεται στην αρχική εικόνα του blender, ώστε να προσεγγίζει καλύτερα το σχήμα του. Αν επιχειρηθεί, όμως, να μεταβάλλουμε τις διαστάσεις του κύβου ανομοιόμορφα, λόγω του τρόπου που έχουμε ορίσει τις συγκρούσεις παίχτη-τοίχου, υπάρχει περίπτωση να ανιχνευθεί νέα σύγκρουση με τον τοίχο μόνο λόγω της περιστροφής της κάμερας (δηλαδή να είμαστε κοντά στον τοίχο και το χέρι του παίχτη να εισχωρήσει σε αυτόν όταν γυρνάμε στο πλάι). Αυτό θα προκαλούσε ακινητοποίηση του παίχτη, ή απότομη διόρθωση θέσης. Αυτός είναι ο λόγος που ο κύβος έχει υποστεί ομοιόμορφο scaling και δεν εξαρτάται δηλαδή ουσιαστικά από τις περιστροφές του παίχτη στον άξονα y.

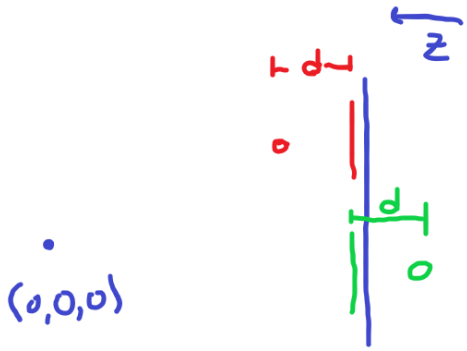
ΜΕΡΟΣ Β

4. Δημιουργία μοντέλου ενός Portal – Portal Image Rendering – Recursive Portals

Για να γίνει κατανοητή η λειτουργία που πρέπει να εκτελεί ένα Portal μπορούμε να το αντιμετωπίσουμε αρχικά απλώς σαν μία πόρτα σχήματος ορθογώνιου παραλληλόγραμμου. Αυτό ακριβώς είναι και το αρχικό μοντέλο που ορίστηκε στο αρχείο “Portal.h”, ένα απλό παραλληλόγραμμο με 4 vertices και προσανατολισμό στο +z. Όπως ακριβώς όταν κοιτάς μέσα από μία ανοιχτή πόρτα βλέπεις κανονικά τα αντικείμενα και τον χώρο πίσω από αυτήν, έτσι και όταν βρίσκεσαι πίσω από ένα Portal, πρέπει να βλέπεις το που θα μεταφερθείς μόλις το διασχίσεις. Μάλιστα, όταν κινείσαι στον χώρο μπροστά από την ανοιχτή πόρτα τα αντικείμενα που βρίσκονται πίσω από αυτήν μετακινούνται αναλόγως ή κάποια αντικείμενα που δεν ήταν πριν ορατά γίνονται τώρα εμφανή. Συμπεραίνουμε, λοιπόν, ότι το πρόβλημα παραγωγής της εικόνας που πρέπει να εμφανίζεται στο εσωτερικό ενός Portal μπορεί να αντιμετωπιστεί ως δύο ξεχωριστά υποπροβλήματα:

1. Ποιος είναι ο **μετασχηματισμός** (πίνακας) που συνδέει την κάμερα (προσανατολισμός και θέση) με ένα ζεύγος Portal και την υπόλοιπη σκηνή, ώστε να δημιουργείται το εφέ της «ανοιχτής πόρτας» που αναφέρθηκε παραπάνω;
2. Με ποιόν τρόπο θα σχεδιαστεί αυτή η εικόνα **μόνο στην επιφάνεια του Portal**, χωρίς να επηρεάζεται η υπόλοιπη σκηνή;

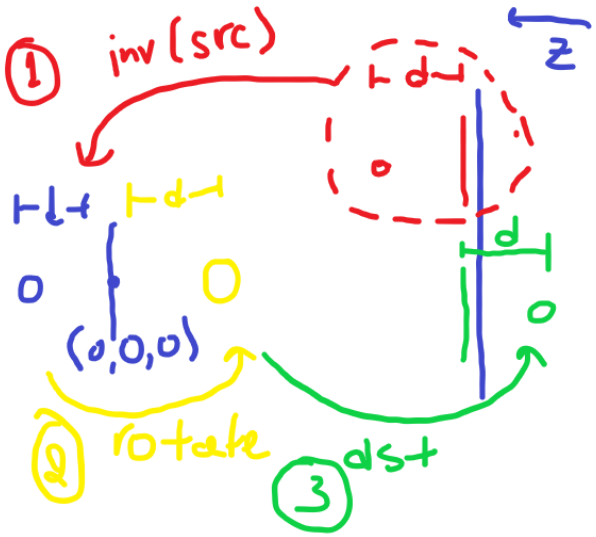
Για τον καθορισμό του μετασχηματισμού αρκεί να πάρουμε ένα απλό παράδειγμα. Έστω ότι τοποθετούμε τα 2 Portals το ένα δίπλα στο άλλο σε έναν τοίχο που κοιτάει προς τον άξονα z. Αυτό σημαίνει ουσιαστικά ότι ο modelMatrix των 2 portals θα αποτελείται απλώς από έναν πίνακα translate (εφόσον το αρχικό μοντέλο του portal δείχνει ήδη προς το +z). Η εικόνα που πρέπει να φαίνεται μέσα στο portal το οποίο κοιτά η κάμερα πρέπει να είναι η ίδια εικόνα που θα έβλεπε αν ήταν στην ίδια απόσταση **πίσω** από το Portal «εξόδου». Δηλαδή:



Άρα πρέπει να βρούμε τον μετασχηματισμό που μεταφέρει την κόκκινη κάμερα στην πράσινη (και να ζωγραφίσουμε αυτό που βλέπει η πράσινη κάμερα στην επιφάνεια του κόκκινου Portal). Στην περίπτωση που εξετάζεται ο μετασχηματισμός είναι πολύ απλό να βρεθεί. Αρχικά θέλουμε να περιστρέψουμε την κόκκινη κάμερα γύρω από το κόκκινο Portal, ώστε να βρεθεί πίσω από αυτό. Για να το πετύχουμε αυτό μετατοπίζουμε το κόκκινο σύστημα τοποθετώντας το portal στο σημείο (0,0,0) του κόσμου πολλαπλασιάζοντας με τον αντίστροφο του modelMatrix του, ο οποίος είναι απλώς ένα translation. Το ίδιο translate δέχεται και η κόκκινη κάμερα με αποτέλεσμα να **διατηρήσει την σχετική της θέση με το κόκκινο Portal**. Έπειτα απλώς περιστρέφουμε την κάμερα γύρω από τον άξονα του κόκκινου Portal κατά 180 μοίρες. Τέλος, θέλουμε να μεταφέρουμε το νέο σύστημα στο σύστημα του portal «προορισμού», το οποίο επιτυγχάνεται με πολλαπλασιασμό με τον modelMatrix του 2ου portal. Ο συνολικός μετασχηματισμός (πολλαπλασιάζει από αριστερά τον viewMatrix της κάμερας) είναι:

portalTransformation= destinationModel* rotate(mat4(),3.14f,vec3(0,1,0))*inverse(sourceModel)

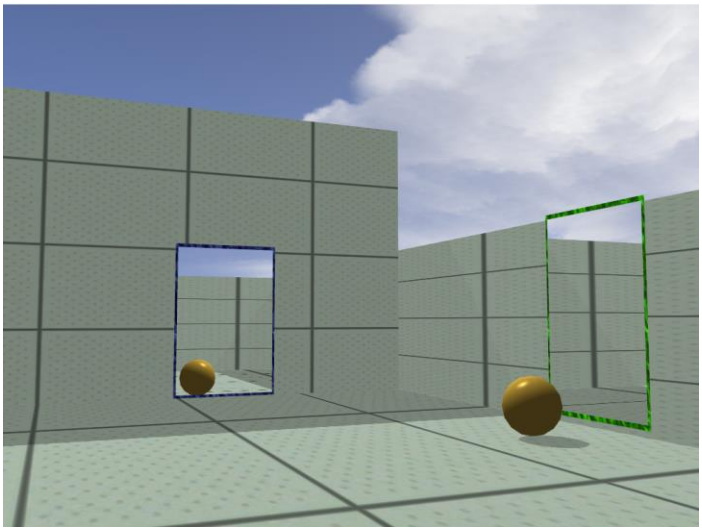
Και η διαδικασία φαίνεται παρακάτω



Ο μετασχηματισμός αυτός είναι το σημαντικότερο βήμα για την σωστή λειτουργία όλων των αρχών ενός ζεύγους Portal, εφόσον εφαρμόζεται ακριβώς με την ίδια λογική και για την «τηλεμεταφορά» των αντικειμένων. Για την δημιουργία του νέου viewMatrix (της πράσινης «κάμερας»), θα πολλαπλασιάσουμε τον αρχικό viewMatrix με τον αντίστροφο αυτού του μετασχηματισμού από δεξιά.

Το επόμενο βήμα, είναι να βρούμε πώς θα σχεδιάσουμε εντός των πλαισίων του Portal που κοιτά η κάμερα, την εικόνα που βλέπει μια δεύτερη **εικονική** κάμερα που βρίσκεται στη θέση που αναφέραμε παραπάνω. Μετά από έρευνα [4][5][6], συμπεράθηκε ότι ο καταλληλότερος τρόπος πραγματοποιείται με την χρήση του stencil buffer.

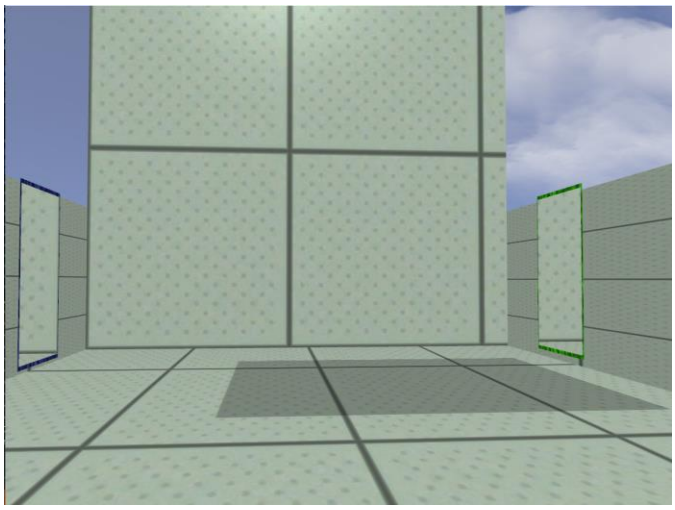
Ο stencil buffer βρίσκεται στην GPU και έχει αποθηκευμένο έναν αριθμό για κάθε pixel της οθόνης. Αν κάνουμε render στον stencil buffer τα Portals και αλλάξουμε την τιμή του stencil buffer στα pixel που περιέχουν, έπειτα μπορούμε να χρησιμοποιήσουμε τα stencil test που διαθέτει η OpenGL για να κάνουμε render τη σκηνή σε δύο φάσεις. Στην πρώτη θα σχεδιάσουμε στο εσωτερικό των Portals (των οποίων τα fragments στην οθόνη θα έχουν πχ στον stencil buffer τιμή 1) την εικόνα που βλέπει η εικονική κάμερα, και στην δεύτερη για να σχεδιάσουμε την υπόλοιπη οθόνη (πχ stencil buffer τιμή 0) όπως φαίνεται από την αρχική κάμερα. Την λειτουργία αυτή εκτελεί η συνάρτηση renderPortalImage() στο αρχείο “lab.cpp”. Τα βήματα που ακολουθούνται περιγράφονται αναλυτικά με σχόλια στον κώδικα της συνάρτησης.

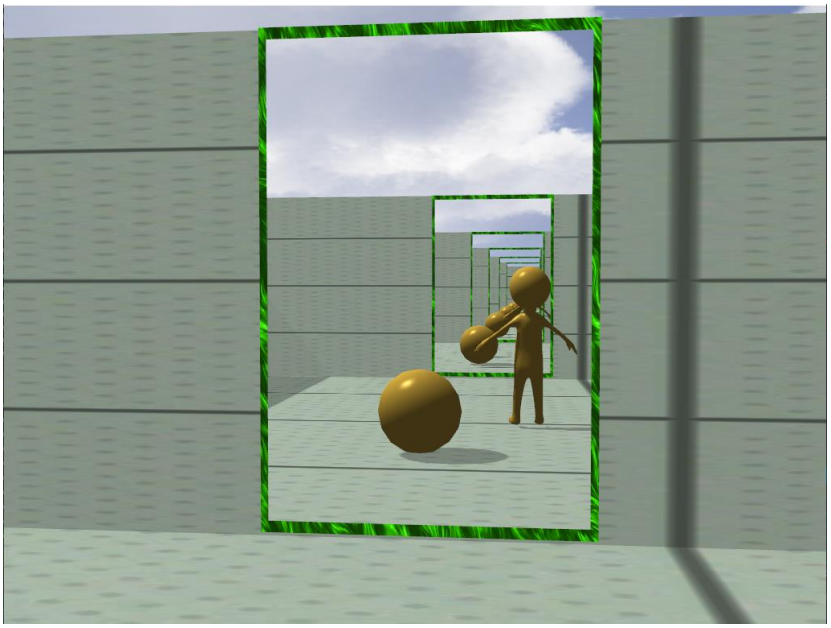


Παρατήρηση: Είναι προφανές ότι θα δημιουργούσε μεγάλο πρόβλημα στην εικόνα των Portals αν μεταξύ των «εικονικών» καμερών και του Portal εξόδου βρισκόταν ένα αντικείμενο της σκηνής που σχεδιάζουμε. Για αυτόν τον λόγο είναι τόσο σημαντική η αποφυγή χρήσης «συμπαγών» τοίχων. Αν τα τοιχώματα ήταν συμπαγή, η οπτική της εικονικής κάμερας θα εμποδιζόταν από την επιφάνεια του τοίχου που θα είχε normal προς την κάμερα. Αντίθετα, στην προκειμένη περίπτωση επειδή οι τοίχοι στους οποίους τοποθετούνται τα Portals είναι πάντοτε στραμμένοι αντίθετα από τις εικονικές κάμερες, με απλή χρήση του backface culling της OpenGL δεν επηρεάζουν καθόλου την παραγόμενη εικόνα. Για να επιτευχθεί το portal image effect στην περίπτωση που έχουμε συμπαγή τοιχώματα, είναι απαραίτητη η τροποποίηση του Near Plane του projection matrix της εικονικής κάμερας, ώστε εκείνο να ταυτίζεται με το επίπεδο του Portal. Τέτοια τροποποίηση μπορεί να γίνει με την τεχνική “Oblique View Frustum Clipping” [7]

RECURSIVE PORTALS

Η συνάρτηση αυτή λειτουργεί αποτελεσματικά όταν κοιτώντας μέσα από ένα Portal, δεν βρίσκεται στο οπτικό μας πεδίο το ίδιο το Portal μέσα από το οποίο κοιτάζουμε, όπως στην παραπάνω εικόνα. Τι γίνεται όμως όταν τοποθετείς δύο Portal το ένα απέναντι από το άλλο, δημιουργώντας έτσι ουσιαστικά έναν «άπειρο διάδρομο»;



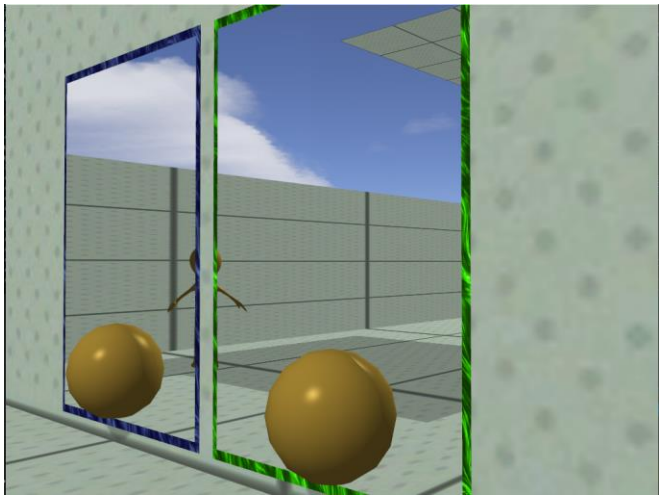


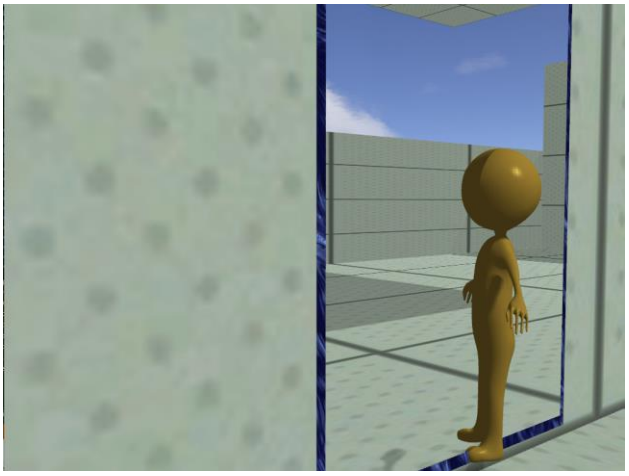
Το επιθυμητό αποτέλεσμα φαίνεται στην εικόνα παραπάνω, που θυμίζει το “infinite loop” που δημιουργείται όταν ένα αντικείμενο τοποθετείται μεταξύ δύο καθρεφτών. Προφανώς, είναι αδύνατον να σχεδιάσουμε άπειρες εκδοχές της αρχικής σκηνής όπως φαίνονται μέσα από άπειρα portals. Αρκεί να σχεδιάσουμε αρκετά «επίπεδα βάθους» ώστε εκ πρώτης όψεως ένας χρήστης του προγράμματος να μην αποπειραθεί καν να μετρήσει τον αριθμό τους. Μάλιστα, αν κανείς παρατηρήσει την παραπάνω εικόνα, βλέπει ότι η σκηνή έχει επανασχεδιαστεί 5 φορές στην επιφάνεια 5 portals (μετρώντας πχ τα πράσινα πλαίσια των portals).

Για να επιτευχθεί αυτό το «τέχνασμα», αρκεί να τροποποιήσουμε την αρχική συνάρτηση ώστε να λειτουργεί **αναδρομικά** μέχρι ένα συγκεκριμένο «βάθος». Η συνάρτηση που υλοποιεί το παραπάνω αποτέλεσμα είναι η renderRecursivePortals() στο αρχείο “lab.cpp” και παίρνει ως ορίσματα το μέγιστο βάθος αναδρομής , το τωρινό βάθος αναδρομής και έναν viewMatrix. Η συνάρτηση καλείται αρχικά στο επίπεδο αναδρομής 0 στην mainloop() (δηλαδή στο «σύμπαν» της αρχικής κάμερας) με τον viewMatrix της αρχικής κάμερας και χρησιμοποιώντας τον γνωστό μετασχηματισμό για να υπολογίσει τον viewMatrix της εικονικής κάμερας, επανακαλείται στο επόμενο «επίπεδο ανάδρασης» με τον νέο viewMatrix ώσπου να φτάσει στο μέγιστο «βάθος». Μέχρι να φτάσει στο μέγιστο βάθος αναδρομής ανανεώνει τις τιμές του stencil buffer σύμφωνα με την «εικόνα» που βλέπουν οι εικονικές κάμερες. Όταν φτάσει στο μέγιστο βάθος, αρχίζει επαναληπτικά να κάνει render την σκηνή, αναλόγως με την τιμή του stencil buffer.

5. Μέθοδος δημιουργίας Smooth Portal

Για να έχουμε ένα ρεαλιστικότερο εφέ πρέπει αν ένα μέρος ενός αντικειμένου (ή του παίχτη) εισέρχεται στο portal τότε να εμφανίζεται ομαλά στην έξοδο, όπως φαίνεται στις φωτογραφίες παρακάτω:



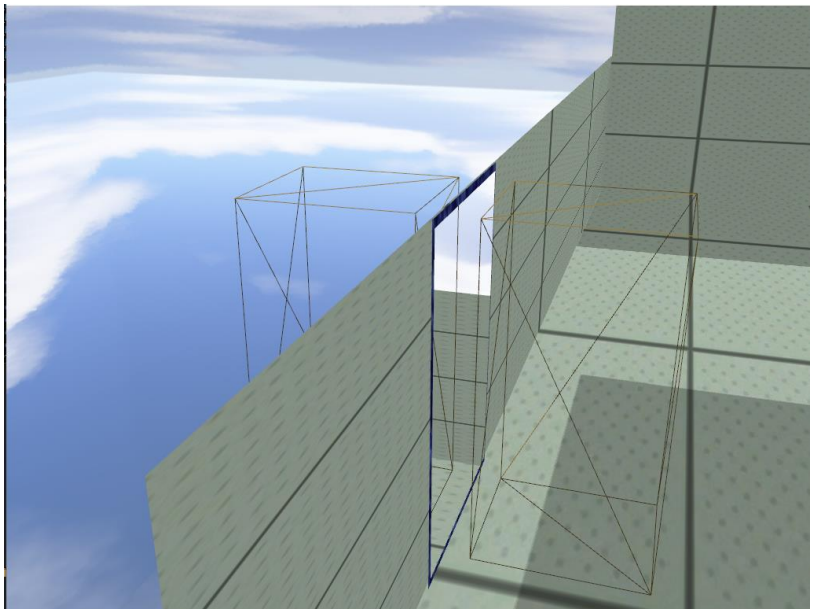


Όπως φαίνεται από την εικόνα η μισή σφαίρα έχει εισέλθει στο πράσινο portal και έχει βγει ήδη από το μπλε portal. Το ίδιο συμβαίνει και με το μοντέλο του παίχτη.

Επειδή δεν είναι δυνατόν να «τηλεμεταφέρουμε» κάθε vertex των αντικειμένων ξεχωριστά, χρησιμοποιούμε ακόμη μία «ψευδαίσθηση». Αρχικά, τοποθετούμε ένα bounding box μπροστά και ένα πίσω από το μοντέλο του portal. Τα κουτιά αυτά πολλαπλασιάζονται με τον modelmatrix του portal και χρησιμεύουν στην ανίχνευση της θέσης των αντικειμένων σχετικά με το portal. Έτσι, «χωρίζουμε» την τηλεμεταφορά του αντικειμένου σε 2 μέρη. Στο πρώτο μέρος βρίσκεται μπροστά από το portal, και σχεδιάζεται τόσο αυτό όσο και ένα **αντίγραφο** του το οποίο έχει ήδη τηλεμεταφερθεί (πολλαπλασιάζουμε με μετασχηματισμό τον modelmatrix και ξανασχεδιάζουμε το αντικείμενο). Το δεύτερο μέρος της «τηλεμεταφοράς» συμβαίνει αν το αντικείμενο τέμνει το πίσω κουτί του portal αλλά **δεν** τέμνει το επίπεδο του portal. Όταν συμβεί αυτό, τη θέση του αντίγραφου λαμβάνει πια το αρχικό αντικείμενο που τηλεμεταφέρεται κανονικά. Η τηλεμεταφορά του αρχικού αντικειμένου είναι σημαντική διότι το αντίγραφο δεν έχει δικές του φυσικές ιδιότητες, παρά μόνο αντιγράφει

εκείνες του αρχικού αντικειμένου, οπότε δεν μπορεί να χρησιμοποιηθεί και για ανίχνευση συγκρούσεων.

Στην παρακάτω εικόνα φαίνονται και τα κουτιά που χρησιμοποιήθηκαν για να δημιουργηθεί αυτό το εφέ.



6. Portal physics – Infinite Loops

Πάνω στην χρήση των portal physics βασίζεται και ο στόχος του παιχνιδιού. Χρησιμοποιώντας τον ψηλό τοίχο και το ταβάνι (ή και με άλλον τρόπο) ο παίχτης μπορεί να «εκτιναχθεί» από το πρώτο πάτωμα στο δεύτερο όπου βρίσκεται το τρόπαιο. Παρόλο που η δημιουργία ενός infinite loop με την σφαίρα είναι ουσιαστικά ήδη έτοιμη, η εφαρμογή στον παίχτη είναι κάπως δυσκολότερη. Για να επιτύχουμε αυτό το εφέ πρέπει αρχικά να εφαρμόσουμε αποτελεσματικά βαρύτητα στον παίχτη και επίσης να εφαρμόσουμε και «τριβές αέρα» στα αντικείμενα, ώστε να αποφύγουμε την συνεχή επιτάχυνση τους κατά το infinite loop. Για να επιτευχθεί η εφαρμογή της βαρύτητας χρειάστηκε να χειριστούμε τις συγκρούσεις του παίχτη με το πάτωμα ξεχωριστά από ότι με τους υπόλοιπους τοίχους, σε αντίθεση με την σφαίρα. Όταν ο παίχτης βρίσκεται πάνω στο πάτωμα απενεργοποιούμε την δύναμη της βαρύτητας και μηδενίζουμε την συνιστώσα της ταχύτητάς του στον άξονα y. Όταν ο παίχτης βρίσκεται πάνω από ένα Portal, αγνοούμε την σύγκρουση με το πάτωμα και

επιτρέπουμε την εφαρμογή της βαρύτητας ώστε ο παίχτης να μπορεί να περάσει μέσα από το Portal και να μην αιωρείται από πάνω του, όπως ουσιαστικά κάνει όταν έχουμε σύγκρουση με το πάτωμα. Ο λόγος που δεν κινείται ο παίχτης στον άξονα y όταν πατάει το πλήκτρο W και κοιτάει προς τα πάνω είναι επειδή μετακινούμε την θέση του κατά την διεύθυνση της προβολής του διανύσματος direction της κάμερας στο επίπεδο XZ αντί του αρχικού διανύσματος. Δεν οφείλεται δηλαδή η κίνηση αυτή στην ύπαρξη βαρύτητας.

Επιπλέον, για να δώσουμε την ψευδαίσθηση της τριβής αέρα, εκτελούμε συνεχώς έναν έλεγχο στην συνιστώσα της ταχύτητας των αντικειμένων προς το -y. Αν η συνιστώσα αυτή έχει ξεπεράσει ένα όριο, απενεργοποιούμε την βαρύτητα στο αντικείμενο και ουσιαστικά έχουμε άθροισμα δυνάμεων ίσο με το 0, δηλαδή δεν επιταχύνεται άλλο το σώμα. Η βαρύτητα θα επανέλθει όταν το σώμα λόγω σύγκρουσης με κάποιον τοίχο μειώσει την κατακόρυφη ταχύτητα του.

Τα δύο αυτά τεχνάσματα δίνουν ένα σχετικά ρεαλιστικό αποτέλεσμα. Τόσο ο παίχτης όσο και η σφαίρα μπορούν να εκτελέσουν οριζόντια βολή με τον τρόπο που περιγράφεται στην εκφώνηση. Είναι απαραίτητο να μετασχηματίσουμε και την ταχύτητα κάθε αντικειμένου κατά την τηλεμεταφορά, εκτός από την θέση του. Ο μετασχηματισμός φυσικά είναι ο ίδιος με εκείνον της θέσης. Λόγω της πολυπλοκότητας των συγκρούσεων του παίχτη, όμως, υπάρχουν μερικές περιπτώσεις όπου το πρόγραμμα λειτουργεί κάπως απρόβλεπτα. Επιπλέον, η χρήση της βαρύτητας προκαλεί μερικές φορές (ανάλογα με την θέση των portals) προβλήματα στην τηλεμεταφορά του παίχτη αν ο παίχτης κινείται με χαμηλή ταχύτητα όταν πάει να διασχίσει ένα portal.

Για το «bonus» ερώτημα της εργασίας, δοκίμασα με τα textures που χρησιμοποίησα και την “puzzle-solving” χρήση των portals να δώσω στο πρόγραμμα αντίστοιχο ύφος με την ομώνυμη τριλογία video games “Portal 1, 2, 3” της εταιρείας “Valve”. Τα textures που χρησιμοποιούνται στους τοίχους είναι από το 1ο παιχνίδι της σειράς.

ΠΑΡΑΡΤΗΜΑ

Οδηγίες χρήσης προγράμματος

Ο παίχτης μετακινείται με τα πλήκτρα WASD και εκτελεί άλμα με το πλήκτρο space. Για να τοποθετήσουμε τα portals χρησιμοποιούμε τα κουμπιά Q και E. Για να εκτοξεύσουμε την μπάλα χρησιμοποιούμε το κουμπί T και το κουμπί R για να την «μαζέψουμε». Αν ο παίχτης πέσει κάτω από την πίστα γίνεται αυτόματα respawn.

Σημείωση: Όταν έχουμε μόνο ένα portal στην σκηνή, χρησιμοποιείται ένα κινούμενο texture αντί για κάποιο portal image, εφόσον δεν έχουμε ζεύγος (δηλαδή έξοδο στο portal). Η επίτευξη του κυκλικού εφέ κίνησης πραγματοποιείται στον shader με πολλαπλασιασμό των «vertex_UVs» των portal vertices με έναν 3χ3 πίνακα μετασχηματισμού που εξαρτάται από τον χρόνο (τον οποίο λαμβάνουμε από την CPU με μία uniform variable). (Fun Fact: Τα textures που χρησιμοποιήθηκαν είναι εικόνες μαλλιών σε μπλε και πράσινο χρώμα που όταν κινούνται κυκλικά δημιουργούν ένα περίεργο εφέ).

Επίσης, το πρόγραμμα είναι φτιαγμένο πάνω στο project του 7ου εργαστηρίου, οπότε για να προστεθούν τα επιπλέον αρχεία που χρειάστηκαν τροποποιήθηκε το αρχείο “CMakeLists.txt”. Το νέο αρχείο περιλαμβάνεται στο παραδοτέο. Τέλος, η κλάση Camera δεν χρησιμοποιείται, και ό,τι χρειάστηκε για την κάμερα του παίχτη μεταφέρθηκε στην νέα κλάση Player.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] LearnOpenGL - Shadow Mapping

[2] Line–plane intersection - Wikipedia

[3] Mouse Picking with Ray Casting - Anton's OpenGL 4 Tutorials (antongerdelan.net)

[4] OpenGL Programming/Mini-Portal - Wikibooks, open books for an open world

[5] Rendering recursive portals with OpenGL - thomas.nl

[6] Rendering "Portal" by torinmr

[7] Oblique View Frustum Depth Projection and Clipping (terathon.com)