Γραφικά και Εικονική Πραγματικότητα - Minecraft

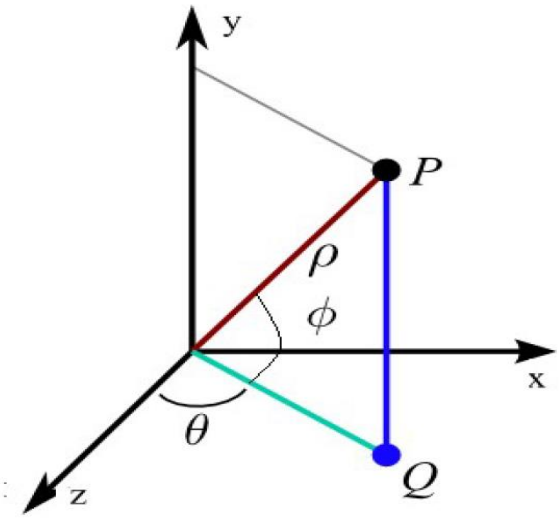
Ιωάννης Νικολάου - 1072681

4ο Έτος

Η εργασία αυτή χωρίζεται σε πέντε βασικά μέρη. Το player movement, world generation, ray-casting, lighting and shadows. Τα ερωτήματα 1-4 απαντώνται στην ενότητα world generation, τα ερωτήματα 5,7 στην ενότητα lighting and shadows και το ερώτημα 6 στην ενότητα ray-casting. Επίσης υλοποιήθηκε το bonus με το chunking, εν μέρει το biome generation κι επιπλέον ένας άλλος τύπος block τα λουλούδια και οι θάμνοι που δεν θεωρούνται blocky κι αναλύονται όλα αυτά πάλι στην ενότητα world generation.

Player Movement

Η κίνηση του παίκτη πραγματοποιείται θεωρώντας πως έχει την ίδια θέση με την κάμερα. Βρίσκοντας λοιπόν από τη θέση του ποντικιού στην οθόνη, την κατεύθυνση που κοιτάει ο παίκτης και τα άλλα κατάλληλα διανύσματα, όπως το right vector και το up vector, μπορούμε να υπολογίσουμε με βάση τις κινήσεις που κάνει με το ποντίκι ή το πληκτρολόγιο την καινούργια του θέση και να δημιουργήσουμε τα view και projection matrices τα οποία λαμβάνουν οι shaders για να μετακινήσουν σωστά τον κόσμο έτσι όπως τον βλέπει ο παίκτης. Οι κινήσεις που μπορεί να κάνει είναι να κουνήσει το ποντίκι κι έτσι να αλλάξει το camera orientation ή να πατήσει κάποιο από τα κουμπιά w, s, a, d, q και e για να μετακινηθεί αντίστοιχα μπροστά, πίσω, αριστερά, δεξιά, πάνω ή κάτω.



Σύμφωνα με αυτό το σχήμα η κατεύθυνση του ποντικιού είναι προς το P κι έτσι μπορούμε εύκολα να βρούμε την horizontal και vertical γωνία ώστε να υπολογίσουμε μετά τα διανύσματα που χρειαζόμαστε.

World Generation

Αρχικά θα μιλήσουμε για τα textures. Τα textures είναι σαν εικόνες όπου ο fragment shader μπορεί να χρησιμοποιήσει για να ζωγραφίσει τις πλευρές του κάθε voxel. Αυτό που συμβαίνει είναι ότι περνάμε έναν texture sampler στον fragment shader σαν uniform

μεταβλητή ώστε να μπορεί να δειγματοληπτήσει το χρώμα του texture σε μια συγκεκριμένη συντεταγμένη.

Στην έκδοση του πρώτου παραδοτέου δεν είχε υλοποιηθεί κάποια τεχνική για να υπάρχουν πολλά textures ταυτόχρονα κι απλώς είχαν γίνει bound λίγα textures τα οποία αντικατόπτριζαν μερικά είδη blocks του Minecraft αλλά σε καμία περίπτωση δεν ήταν αρκετά.

Ο fragment shader δεν μπορεί να έχει πρόσβαση σε όσα textures εμείς θέλουμε. Ο αριθμός των textures που μπορούμε να έχουμε καθορίζεται από τον αριθμό τον texture units που έχει η gpu και είναι συχνά ένας μικρός αριθμός. Η λύση σε αυτό το πρόβλημα έρχεται με το να συνδυάσουμε πολλά διαφορετικά textures σ' ένα μεγάλο texture.

Υπάρχουν δύο τρόποι για να γίνει αυτό. Ο πρώτος είναι το texture atlas όπου είναι ένα δισδιάστατο texture στο οποίο τοποθετούνται όλα τα textures το ένα δίπλα στο άλλο και χρησιμοποιούμε texture coordinates για να διαλέξουμε το συγκεκριμένο που θέλουμε. Ο δεύτερος είναι το texture array όπου είναι ένα τρισδιάστατο texture όπου στο οποίο τοποθετούνται όλα τα textures το ένα πάνω από το άλλο και χρησιμοποιούμε την z συντεταγμένη για να διαλέξουμε το συγκεκριμένο που θέλουμε. Στην παρούσα εργασία χρησιμοποιήθηκε ο δεύτερος τρόπος καθώς ο αριθμός των textures που μπορούμε να έχουμε είναι πολύ μεγαλύτερος από την πρώτη περίπτωση.

Έχουμε λοιπόν έναν texture manager ο οποίος δέχεται ένα πλάτος, ύψος και το συνολικό αριθμό των textures που θέλουμε να έχουμε. Αυτό είναι ένα μικρό μειονέκτημα των texture arrays καθώς κάθε texture πρέπει να έχει το ίδιο πλάτος και ύψος. Ο texture manager κρατάει τη λίστα όλων των textures που έχουν ήδη προστεθεί. Έχει μια λειτουργία όπου προσθέτει ένα νέο texture εφόσον δεν υπάρχει ήδη και χρησιμοποιείται από το κάθε voxel ώστε να βρει το κατάλληλο z-index για το texture που πρέπει να βάλει στη συγκεκριμένη πλευρά του voxel.

Έπειτα δημιουργούμε τα διαφορετικά voxel types που θα έχει ο κόσμος μας, όπως grass, sand, dirt, stone, ακόμα και φυτά, κάκτους. Έτσι μπορούμε να έχουμε δικά μας custom voxel models με βάση τον φάκελο models, όπου εκεί αποθηκεύονται οι απαραίτητες πληροφορίες για κάθε μοντέλο, όπως vertex, texture, normal positions, shading values για την κάθε πλευρά του κάθε voxel και επιπλέον πληροφορίες για το αν το μοντέλο είναι κύβος και αν πρέπει να είναι transparent. Οπότε αναλόγως το όνομα του texture και την πλευρά στην οποία θέλουμε να το τοποθετήσουμε υπάρχει μια συνάρτηση με την οποία πάμε στην συγκεκριμένη πλευρά των texture coordinates και θέτουμε το z component ίσο με το index το οποίο αντιστοιχεί στο συγκεκριμένο texture.

Τώρα είναι το κύριο κομμάτι του world generation όπου θα φορτώνονται πολλά voxels ώστε να μπορούμε να τα δούμε την ίδια στιγμή όλα μαζί. Υπάρχουν πολλές προσεγγίσεις ώστε να επιτευχθεί αυτό. Ο πιο εύκολος τρόπος που ακολουθήθηκε και στο πρώτο ερώτημα είναι να κάνουμε translate το κάθε voxel ξεχωριστά στη σωστή θέση και να το ζωγραφίζουμε. Αυτός είναι πολύ αργός τρόπος καθώς οι draw calls στην opengl είναι γενικά πολύ χρονοβόρες κλήσεις καθώς είναι I/O operations με αποτέλεσμα τα FPS να είναι πάρα πολύ χαμηλά. Το N ήταν γενικά χαμηλό στην καλύτερη περίπτωση ήταν γύρω στο 16-22.

Η λύση που είχε επιλεγεί στην έκδοση του πρώτου παραδοτέου ήταν το instancing. Παρότι ήταν αποδοτική για το rendering πολλών voxels και το N έφτανε γύρω στο 300 υστερούσε στο να δημιουργηθεί ένα συνολικό mesh για όλα τα voxels καθώς δεν είχε αποθηκευτεί κάπου μέσα στο πρόγραμμα οι θέσεις των chunks παρά μόνο τα translation matrices τους.

Μια πολύ καλύτερη λύση είναι να ζωγραφίσουμε πολλά voxels με μία κλήση συστήματος ομαδοποιώντας τα σε chunks. Η γενική ιδέα είναι να δημιουργήσουμε ένα συνολικό mesh για ολόκληρο το chunk βασιζόμενο στα voxels που το αποτελούν σε αντίθεση να δημιουργούσαμε ένα για κάθε voxel. Τα chunks έχουν μέγεθος 16x16x16 που θεωρείται σχετικά αποδοτικό. Για ακόμη μεγαλύτερη βελτιστοποίηση του loading και deletion των voxels και των chunks χρησιμοποιήθηκαν και subchunks που βοήθησαν ιδιαίτερα στη συνέχεια στο placing and deletion of voxels. Η ιδέα είναι να χωρίσουμε τα chunks σε μικρότερα subchunk meshes τα οποία μπορούμε να ανανεώσουμε ξεχωριστά και γρήγορα και στη συνέχεια να ανανεώσουμε και το αρχικό chunk απλώς ενώνοντας όλα τα subchunk meshes σε ένα μεγάλο mesh το οποίο καταλήγουμε να στείλουμε στην gpu. Το μέγεθος των subchunk που επιλέχθηκε είναι 4x4x4.

Στη δημιουργία του κόσμου έχουμε τη θέση του κάθε chunk και την αποθηκεύουμε και αφού το δημιουργούμε θέτουμε αρχικά κάθε voxel σε ένα ειδικό τύπο voxel air ώστε απλώς να έχουμε καταλάβει τον χώρο που περικλείει το voxel και συνολικά το chunk ώστε στη συνέχεια να μπορούμε να το τροποποιήσουμε και να μη μπορεί να δεσμεύσει αυτό τον χώρο άλλο chunk. Έτσι για κάθε chunk δημιουργούμε subchunks μέσα του όπου στην αρχικοποίηση επειδή το voxel_number είναι 0 καθώς θέσαμε όλα τα voxels μέσα στο chunk σε αέρα δεν θα γίνουν οι έλεγχοι για να φτιαχτεί το mesh του chunk. Το συνολικό mesh του κάθε chunk φτιάχνεται στη συνέχεια αφού έχει δεσμευτεί ο χώρος από όλα τα chunks όπου όλα τους τα voxel είναι αέρας.

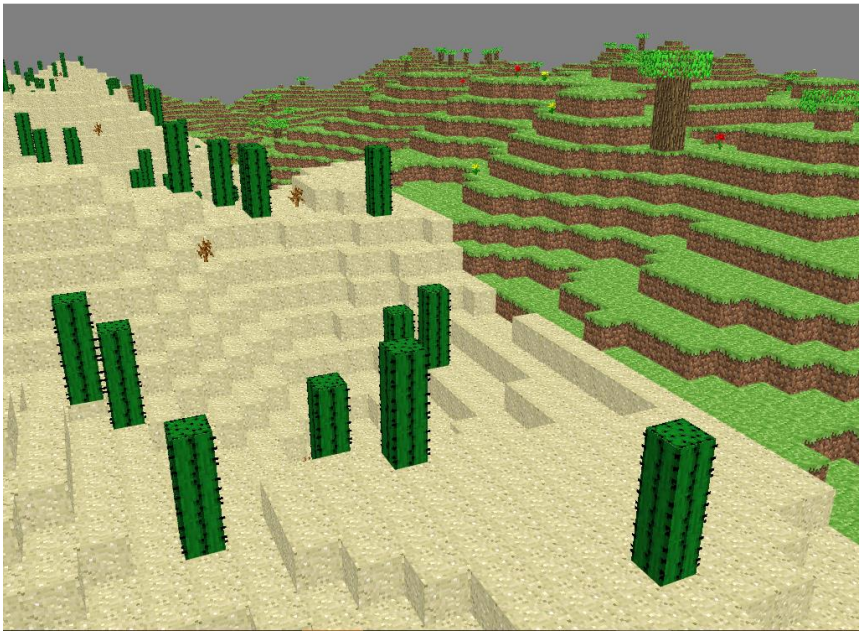
Το ύψος που θα έχει το κάθε column ουσιαστικά μέσα στο chunk καθορίζεται από ένα heightMap. Δημιουργείται ένα height map με βάση τον αλγόριθμο simplex που έχει η βιβλιοθήκη glm και χρησιμοποιούνται οι συντεταγμένες x, z του κάθε voxel μέσα στο chunk ώστε να βρεθεί μια τυχαία τιμή του ύψους. Χρησιμοποιούνται 3 values που γίνονται sample με διαφορετικό frequency και γίνεται έπειτα ένα scaling και μια κανονικοποίηση ώστε να βρεθεί το τελικό ύψος του συγκεκριμένου column. Στη συνέχεια έχοντας το heightmap για το συγκεκριμένο chunk χωρίζουμε τον κόσμο σε δύο biomes στην έρημο και στη ζούγκλα αφού γνωρίζουμε τη συνολική του έκταση. Μέχρι να φτάσουμε το ύψος που μας επέστρεψε το heightmap θέτουμε τα voxels grass ή sand ανάλογα με το biome. Χρησιμοποιούμε έναν τυχαίο number generator ώστε να κάνουμε sparse placing άλλων voxel types όπως λουλούδια, κάκτους, θάμνους και δέντρα.

Στη ζούγκλα ελέγχουμε αν βρισκόμαστε στα άκρα του chunk κι αν έχουμε χώρο να τοποθετήσουμε στο συγκεκριμένο column του chunk 3 oak wood voxels μετά το ύψος που μας επέστρεψε το heightmap. Αν αυτό είναι εφικτό θέτουμε το voxel type ίσο με το συγκεκριμένο index που έχει το wood και κοιτάμε αν έχουμε φτάσει στο όριο του chunk. Αν έχουμε φτάσει ή στη συγκεκριμένη θέση που βρισκόμαστε έχουμε ήδη συμπληρώσει το ύψος του δέντρου τότε αποθηκεύουμε τη θέση που βρισκόμαστε ώστε στη συνέχεια να βάλουμε και φύλλα. Σε άλλους generated αριθμούς και συγκεκριμένα ύψη βάζουμε τα υπόλοιπα voxel types, αλλιώς το voxel

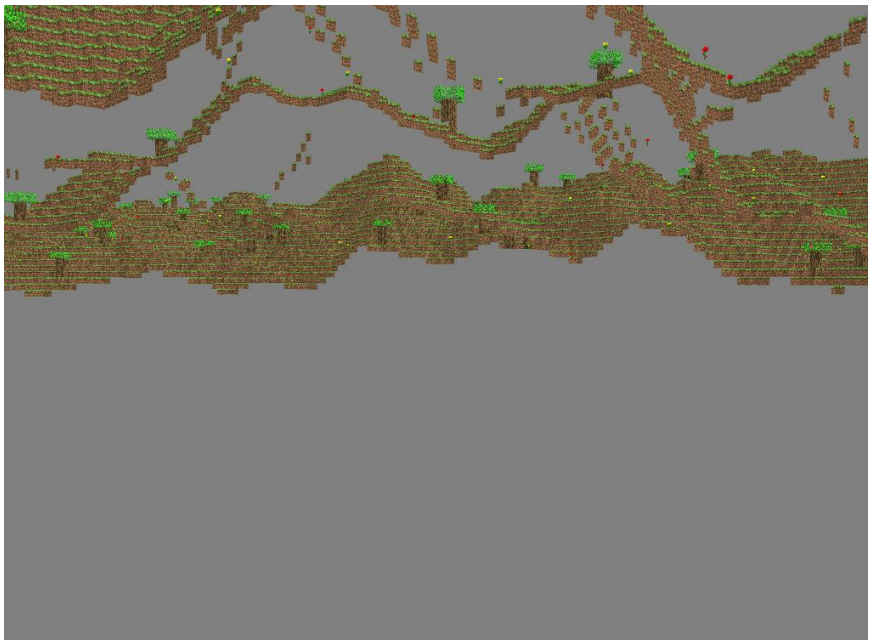
type air παραμένει. Αφού τελειώσουμε με όλα τα voxels επιστρέφουμε στις θέσεις που είχαμε αποθηκεύσει για την κορυφή των δέντρων και προσθέτουμε γύρω γύρω τα φύλλα που έχουν.

Έτσι λοιπόν αφού ολοκληρωθεί το terrain generation πρέπει να ανανεώσουμε τις πληροφορίες για το κάθε voxel αφού πλέον δεν είναι όλα τα voxels type air. Αυτό γίνεται διατρέχοντας όλα τα chunks που έχουν δημιουργηθεί στον κόσμο κάνοντας πρώτα update τα subchunk meshes κι έπειτα update το mesh του chunk. Όταν γίνεται update το subchunk mesh κάθε subchunk του chunk αυτό που συμβαίνει είναι να βρούμε το index του συγκεκριμένου voxel που διατρέχουμε μέσα στο chunk και να ελέγξουμε αν κάθε πλευρά γύρω του είναι transparent ή όχι. Αν δεν είναι transparent τότε μπορούμε να προσθέσουμε τη συγκεκριμένη πλευρά στο mesh του συγκεκριμένου subchunk. Όταν ολοκληρωθεί αυτό για κάθε subchunk του chunk τότε προσθέτουμε όλες τις πληροφορίες για vertex, texture, normal positions και shading values όλων των subchunks στα vectors που έχει το chunk γι’ αυτό το σκοπό κι έπειτα στέλνουμε αυτά τα δεδομένα στην gpu έχοντας bound τους κατάλληλους buffers.

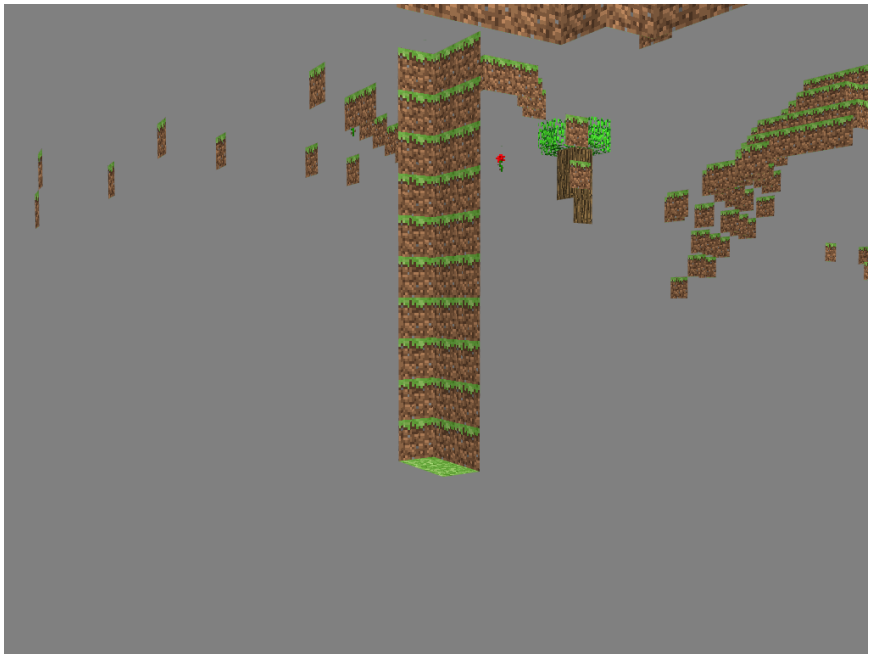
Με την τεχνική του να δημιουργούμε ένα συνολικό mesh για κάθε chunk γλιτώνουμε πάρα πολύ υπολογιστική ισχύ έχοντας enabled επίσης και το back face culling. Έτσι μόνο τα voxels που είναι ορατά στον χρήστη είναι κάθε φορά ορατά, όπως μπορεί να φανεί και από την παρακάτω φωτογραφία.



Εδώ είναι ο κόσμος από τη μεριά του χρήστη όπου φαίνονται μόνο οι πλευρές των voxels που δεν έχουν κοπεί από το back face culling καθώς δείχνουν προς την κάμερα.



Στην παραπάνω φωτογραφία είμαστε κάτω από την επιφάνεια και βλέπουμε ότι λόγω του back face culling και του chunk-subchunk system έχουν γίνει load πολύ λιγότερα voxel faces σε σχέση με αυτά που υπάρχουν στην πραγματικότητα.



Τέλος σ’ αυτή την φωτογραφία φαίνεται πόσο χρήσιμο είναι το subchunking system αφού έχει γίνει το deletion μερικών voxels έχοντας σκάψει ο χρήστης κατακόρυφα προς τα κάτω. Έχουν γίνει load πολύ λιγότερα voxel faces σε σχέση με το να γινόταν αυτό για όλο το chunk στο οποίο ανήκουν τα voxels κάνοντας το user experience πιο ευχάριστο και τα FPS υψηλότερα.

Ray Casting

Όταν ο παίκτης κάνει κλικ με το ποντίκι θέλει να προσθέτει ή να καταστρέφει τα voxels. Αυτό που θέλουμε είναι με βάση το direction της κάμερας να μπορούμε να αντιλαμβανόμαστε ποιο voxel διάλεξε. Έχοντας πάλι το unit vector που αντικατοπτρίζει την κατεύθυνση που κοιτάει ο παίκτης όπως είχαμε και στο player movement, θέλουμε πάνω στη γραμμή που ορίζει αυτό το διάνυσμα να έχουμε ένα σημείο P το οποίο αυξάνεται σταδιακά και τελικά προσπίπτει στην τομή της γραμμής αυτής και του επιπέδου του τρισδιάστατου grid στο οποίο υπάρχει το voxel. Οπότε πρέπει να έχουμε υπόψιν την αρχική θέση του σημείου αυτού και την σταδιακή θέση που αυτό λαμβάνει μετά από κάθε μετατόπισή του πάνω στη γραμμή του unit vector (η αρχική θέση πάντα είναι η θέση της κάμερας).

Αρχικά θέλουμε να βρούμε την πιο κοντινή τομή του σημείου μας μεταξύ της γραμμής και του επιπέδου του grid. Αντί να σκεφτούμε όλες τις πιθανές τομές μπορούμε να σκεφτούμε τις 6 πιθανές πλευρές οι οποίες περικλείουν το P στην οποία περίπτωση μπορούμε να κεντράρουμε αυτές τις πλευρές σε σχέση με το origin αφαιρώντας το voxel position στο οποίο βρίσκεται το σημείο μας. Αυτό το voxel position θα είναι ένα integer vector b και μπορούμε να το θεωρήσουμε ως το στρογγυλοποιημένο point P position. Αυτό το κάνουμε για να βρούμε το voxel position στο οποίο βρίσκεται το σημείο μας επειδή τα voxels είναι 1 unit σε πλάτος και είναι κεντραρισμένα σε ακέραιες θέσεις.

Χρησιμοποιούμε μια step συνάρτηση η οποία παίρνει τη θέση του σημείου μας και αφαιρεί τη θέση του block και καταλήγουμε με μια νέα τοπική θέση. Το νέο σημείο αλλά και η μετατοπισμένη γραμμή ορίζονται ως O και r αντίστοιχα. Μια ακόμα τεχνική που μπορούμε να ακολουθήσουμε είναι να μη λάβουμε υπόψιν τις αρνητικές πλευρές αφού μπορούμε να αντιστρέψουμε την θέση μας γύρω από το origin σε μια συντεταγμένη αν η συντεταγμένη του unit vector είναι αρνητική για να πάρουμε το ίδιο αποτέλεσμα όπως οι θετικές πλευρές. Αφού πλέον δεν χρειάζεται να ανησυχούμε για το πρόσημο του unit vector όταν θα υπολογίζουμε τις τομές μπορούμε να δημιουργήσουμε δύο absolute vectors το u και το L για το unit vector και το O αντίστοιχα.

Τώρα πρέπει να βρούμε ποια είναι η πιο κοντινή τομή στο L μεταξύ του r και των 3 πλευρών. Χρειαζόμαστε εξισώσεις για να ορίσουμε το r και τις πλευρές μας. Το r είναι εύκολο να το ορίσουμε καθώς έχουμε το v διάνυσμα που δείχνει την κατεύθυνση κι ένα σημείο από το οποίο να περάσει που είναι το L. Μπορούμε να δημιουργήσουμε μια παραμετρική εξίσωση όπως αυτή.

$$\mathrm{r} \equiv \frac{x - L_x}{\overrightarrow{v_x}} = \frac{y - L_y}{\overrightarrow{v_y}} = \frac{z - L_z}{\overrightarrow{v_z}}$$

Θα δούμε τώρα για την πλευρά που βρίσκεται στον χ άξονα F_x αλλά η ίδια μεθοδολογία λειτουργεί και για τις άλλες 2 πλευρές. Για να βρούμε την τομή με μία πλευρά πρώτα πρέπει να

βρούμε την τομή με το επίπεδο που ορίζει αυτή η πλευρά που ονομάζουμε k, αφού οι πλευρές μας είναι ευθυγραμμισμένες με τους άξονες μπορούμε να χρησιμοποιήσουμε την εξίσωση των επιπέδων τους.

$$k \equiv x = \frac{1}{2}$$

Βάζουμε τώρα τις δύο αυτές εξισώσεις μαζί στο ίδιο σύστημα και λύνουμε ως προς x, y και z.

$$k \cap r \equiv \begin{cases} x = \frac{1}{2} \\ \frac{x - L_x}{\overrightarrow{v_x}} = \frac{y - L_y}{\overrightarrow{v_y}} = \frac{z - L_z}{\overrightarrow{v_z}} \end{cases} \Leftrightarrow \begin{cases} x = \frac{1}{2} \\ y = \frac{\frac{1}{2} - L_x}{\overrightarrow{v_x}} \overrightarrow{v_y} + L_y \\ y = \frac{\frac{1}{2} - L_x}{\overrightarrow{v_x}} \overrightarrow{v_z} + L_z \end{cases}$$

Μπορούμε να μην τα κάνουμε όλα αυτά αν το $\overrightarrow{v_x}$ είναι μηδέν καθώς αυτό σημαίνει ότι το r και το k είναι παράλληλα και δεν θα μπορούσαν ποτέ να έχουν τομή.

Τώρα γνωρίζουμε που στον χώρο βρίσκεται η τομή με το επίπεδο αλλά μένει να ελέγξουμε αν η τομή βρίσκεται μέσα στην πλευρά μας. Αυτό είναι σχετικά εύκολο καθώς οι πλευρές μας είναι ευθυγραμμισμένες με τους άξονες. Πρέπει να πάρουμε την απόσταση μεταξύ του σημείου της τομής και του L και να μετακινήσουμε το P κατά μήκος της γραμμής του αρχικού μοναδιαίου διανύσματος χρησιμοποιώντας το πυθαγόρειο θεώρημα.

$$d = \sqrt{(x - L_x)^2 + \left(y - L_y\right)^2 + (z - L_z)^2}$$

Έχουμε μια συνάρτηση check η οποία λαμβάνει αυτή την απόσταση, τη θέση του τρέχοντος voxel, τη θέση του επόμενου voxel που είναι η θέση του voxel που είναι γειτονικό στην πλευρά μας και την ενέργεια που θέλει να εκτελέσει ο χρήστης. Αυτή η ενέργεια είναι να σπάσει ένα voxel με το αριστερό κλικ, να τοποθετήσει ένα voxel με το δεξί κλικ και να κάνει sample ένα voxel που βρίσκεται στον κόσμο με το μεσαίο κλικ ώστε να το τοποθετήσει στη συνέχεια. Κάνουμε την ίδια διαδικασία και για τις άλλες δύο πλευρές. Όσο το επόμενο block είναι αέρας πρέπει να μετακινήσουμε το P πάνω στην ευθεία σύμφωνα με την εξίσωση

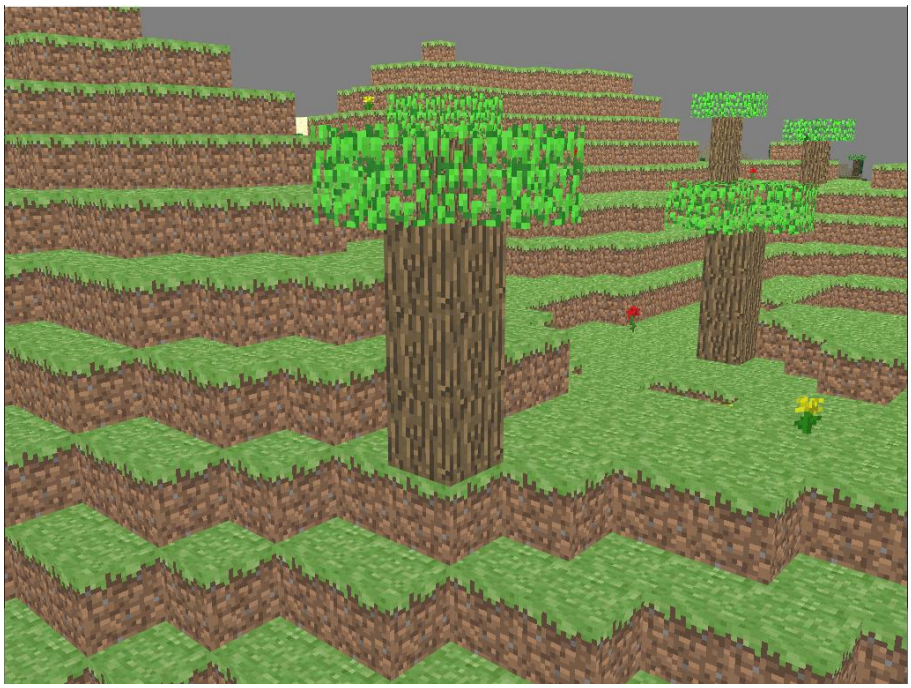
$$P + \vec{v} \cdot d$$

Να θέσουμε τη θέση του τωρινού voxel ως τη θέση του επόμενου voxel και να αυξήσουμε τη συνολική απόσταση με την απόσταση που υπολογίσαμε προηγουμένως ώστε να μην έχει ο παίκτης απεριόριστο hit range.

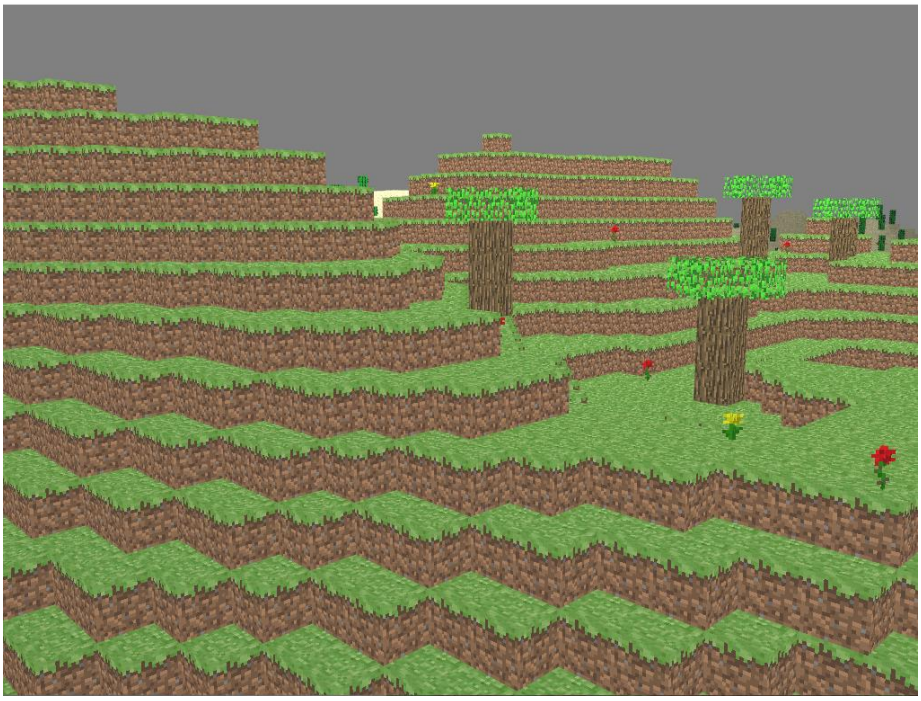
Τώρα όσον αφορά τα 3 inputs που μπορεί να έχει ο παίκτης με το ποντίκι. Όταν θέλει να προσθέσει ένα voxel πρέπει πρώτα να βρούμε σε ποιο chunk του κόσμου ανήκει αυτό το voxel κι αν δεν ανήκει να δημιουργήσουμε καινούργιο chunk. Στη συνέχεια ελέγχουμε αν ο τύπος του voxel είναι διαφορετικός από αυτόν που υπάρχει ήδη σε αυτή τη θέση κι έπειτα κάνουμε update τα απαραίτητα subchunks και chunks των γειτονικών voxels. Όταν ο χρήστης

θέλει να αφαιρέσει ένα voxel εκτελούμε την ίδια διαδικασία αλλά αντί για το holding voxel type θέτουμε το voxel type του συγκεκριμένου voxel με αέρα. Στην τελευταία ενέργεια του χρήστη που είναι να αλλάξει τον τύπο του voxel που έχει εκείνη τη στιγμή στα χέρια του πρέπει να βρούμε σε ποιο chunk του κόσμου ανήκει αυτό το voxel κι έπειτα να βρούμε την τοπική θέση του voxel μέσα στο chunk ώστε να μπορούμε να έχουμε πρόσβαση στον τύπο του.

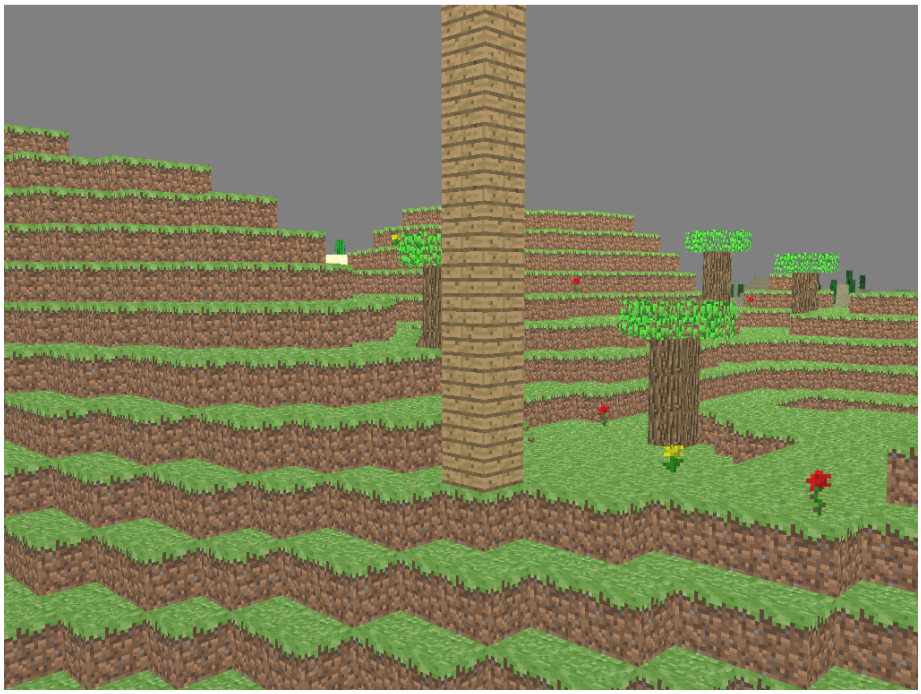
Τώρα θα δείξουμε με εικόνες τις κινήσεις που έχει ο χρήστης ξεκινώντας με την παρακάτω αρχική εικόνα.



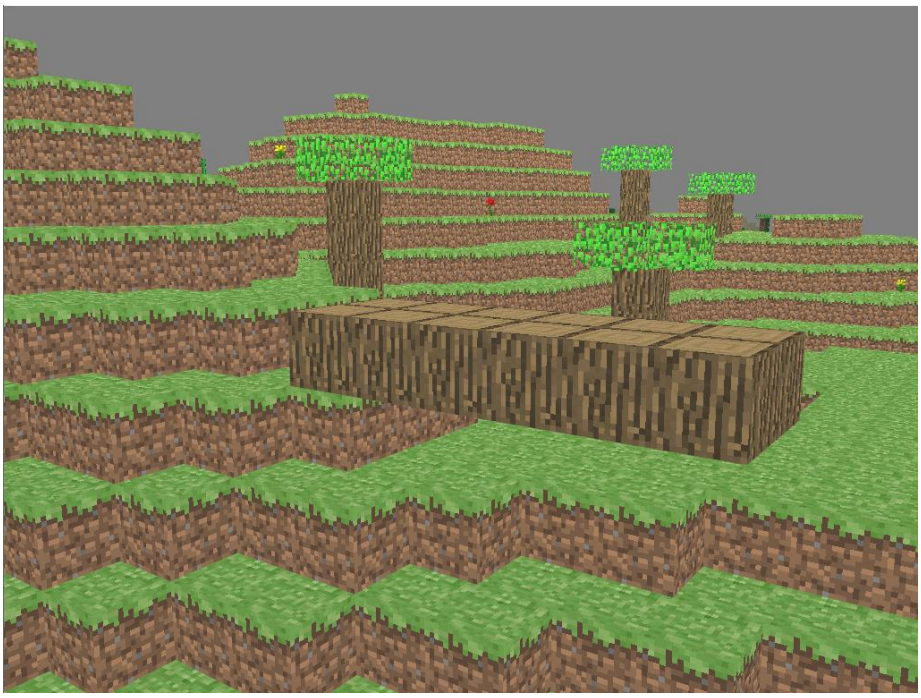
Έστω ότι ο χρήστης θέλει να καταστρέψει το πιο κοντινό δέντρο τότε μετά από διαδοχικά κλικ του αριστερού κλικ του ποντικιού το αποτέλεσμα που θα προκύψει είναι να εξαφανιστεί το δέντρο όπως φαίνεται στην παρακάτω φωτογραφία.



Έστω ότι ο χρήστης θέλει να τοποθετήσει μερικά voxels τότε πατώντας το δεξί κλικ του ποντικιού μπορεί με τον προεπιλεγμένο τύπο voxel plank να δημιουργήσει τον παρακάτω πύργο.



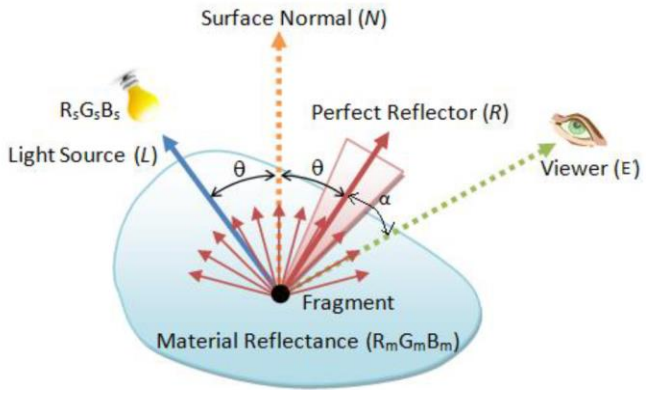
Τέλος συνδυάζοντας και την επιλογή του sampling με το μεσαίο κλικ του ποντικιού μπορεί να διαλέξει πάλι τον τύπο voxel oak wood και να φτιάξει μια αυτοσχέδια κατασκευή.



Lighting - Shadow Mapping

Το σύστημα φωτισμού που ακολουθήθηκε είναι σύμφωνο με το συνδυασμό της θεωρίας φωτισμού του Phong και μερικών αρχικών shading values που δίνονται στο κάθε μπλοκ καθώς είναι γνωστό ότι αρχικά η πάνω πλευρά θα είναι πιο φωτεινή η πλάγιες πλευρές λιγότερο και η κάτω πλευρά το λιγότερο από όλες. Η θεωρία του Phong ξεκινάει με μία συνιστώσα φωτός για το ambient light που υπάρχει πάντα σε όλα τα fragments που ζωγραφίζονται στη σκηνή. Την diffuse συνιστώσα όπου έχει σχέση με το normal της επιφάνειας κάθε primitive τριγώνου που βρίσκεται στη σκηνή και τη γωνία που σχηματίζει με το vector της φωτεινής πηγής κι έτσι λαμβάνοντας το συνημίτονο αυτής της γωνίας μπορούμε να υπολογίσουμε τη diffuse συνιστώσα. Τέλος η specular συνιστώσα έχει σχέση και με τη γωνία παρατήρησης, πόσο απέχει η τέλεια ανάκλαση του φωτός πάνω στην επιφάνεια από τη γωνία παρατήρησης της κάμερας κι έτσι λαμβάνοντας το συνημίτονο αυτής της γωνίας κι έχοντας και το specular factor μπορούμε να μοντελοποιήσουμε τη specular συνιστώσα. Ακόμη μπορούμε να προσθέσουμε στην εξίσωση του phong και την απόσταση της φωτεινής πηγής και την ισχύ της φωτεινής πηγής ώστε να έχουμε πιο ρεαλιστικά αποτελέσματα με σκοπό ένα αντικείμενο πολύ κοντά στο φως να είναι περισσότερο φωτισμένο από ένα αντικείμενο που βρίσκεται πιο μακριά παρόλο που μπορεί η diffuse και specular συνιστώσες τους να είναι ίδιες. Όλες αυτές οι πράξεις πραγματοποιούνται στον fragment shader για κάθε fragment και απλώς χρειάζεται να αρχικοποιηθεί μια φωτεινή πηγή με σταθερές ambient, diffuse, specular, ισχύος και την αρχική θέση της φωτεινής πηγής. Η φωτεινή πηγή μπορεί με τα κουμπιά u, j, h, k, i, y να μετακινηθεί μπροστά, πίσω, αριστερά, δεξιά, πάνω και κάτω αντίστοιχα. Οι εξισώσεις για το μοντέλο του Phong είναι οι εξής.

Fragment Shader



Color Model

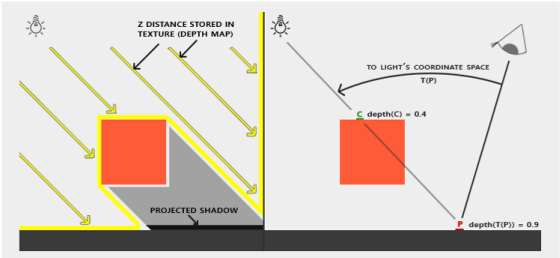
- $Color = Ambient + Diffuse + Specular$
- $Diffuse = \frac{Material\ Diffuse\ Color * Light\ Color * Light\ Power * \cos(\theta)}{distance^2}$
- $Specular = \frac{Material\ Specular\ Color * Light\ Color * Light\ Power * \cos(\alpha)^{Specular\ Explonet}}{distance^2}$

Το shadow mapping έχει υλοποιηθεί κρατώντας σε ένα texture τα depth values για όλα τα fragments από την πλευρά που τα βλέπει το φως με τον depth fragment shader. Έπειτα αφού έχουμε αυτό το depth map το δίνουμε στον basic fragment shader κι αφού έχουμε τη συντεταγμένη κάθε fragment όπως τη βλέπει το φως κι όπως την βλέπει η κάμερα μπορούμε να συγκρίνουμε το depth value έτσι όπως το βλέπει η κάμερα με το αποθηκευμένο depth value από το depth pass με σκοπό να δούμε αν το συγκεκριμένο fragment θα αποτελεί μέρος της σκιάς ή όχι.

Θεωρία Σκίασης

Βήματα:

1. Σχεδίαση της σκηνής από την οπτική του φωτός
2. Δημιουργία ενός depth map
3. Σχεδίαση της σκηνής
   -> Μεταφορά κάθε fragment στο lightspace και ελέγχουμε αν η απόσταση του είναι μικρότερη/ίση από την τιμή που είναι αποθηκευμένη στον depth buffer για να δούμε αν σκιάζεται.



Τα αποτελέσματα που προκύπτουν εφαρμόζοντας τη θεωρία της σκίασης και του φωτισμού στο πρόγραμμα φαίνονται παρακάτω. Καθώς έχει γίνει placing καινούργιων voxels η σκίαση και ο φωτισμός προσαρμόζονται αναλόγως.