Γραφικά και Εικονική Πραγματικότητα

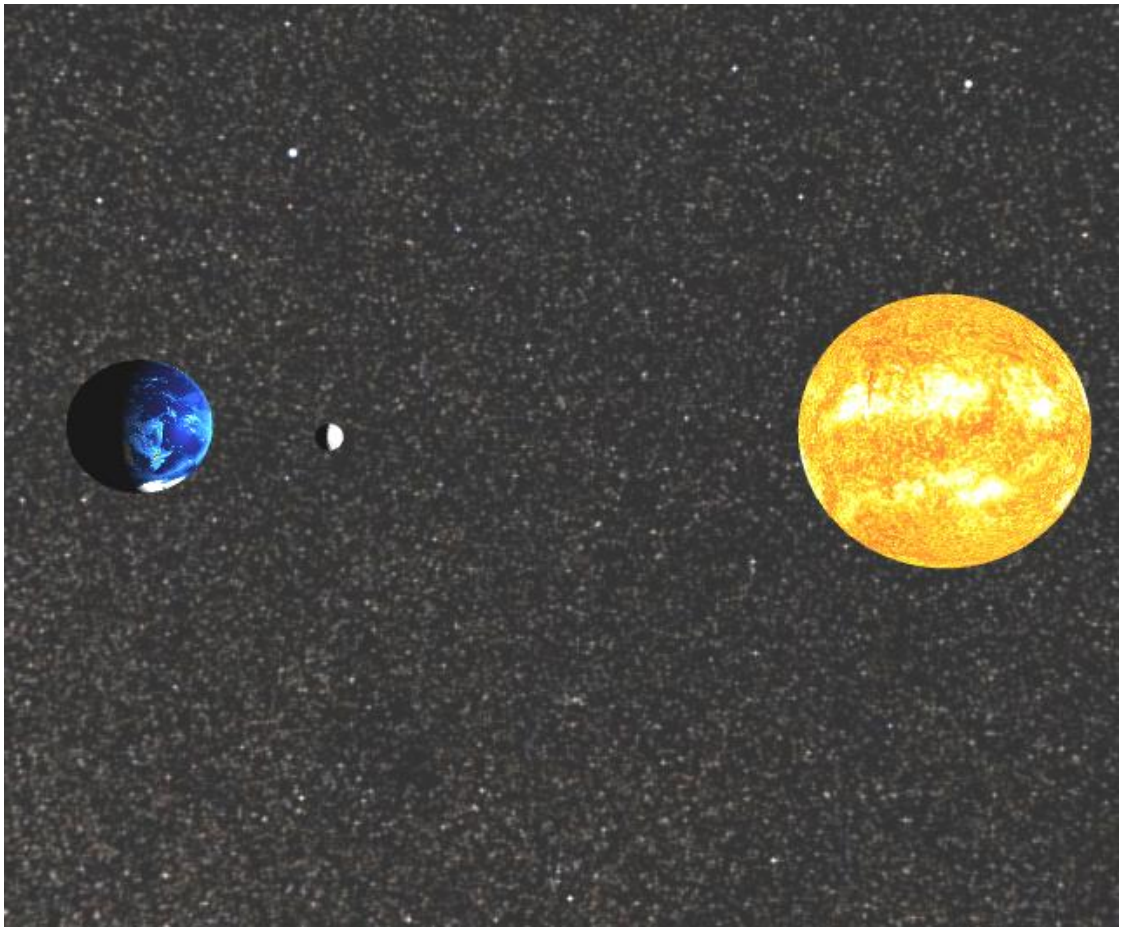
2017-18

Απαλλακτική εργασία

3 Body Problem

Αναστασόπουλος Νικόλαος

228268



Περιεχόμενα

1. Εισαγωγή

Σκοπός αυτής της εργασίας είναι η απεικόνιση πλανητικών σωμάτων στο γραφικό περιβάλλον που δημιουργείται με την OpenGL.

Αρχικά, πρέπει να γίνει animation τριών πλανητών (ήλιος - γη - σελήνη). Στο animation πρέπει να περιλαμβάνεται περιήγηση στο γραφικό περιβάλλον, ρεαλιστικότητα των πλανητών και σωστή αναπαράσταση των πλανητών σε σχέση με τις πραγματικές τροχιές τους στον χωροχρόνο.

Στο δεύτερο μέρος της εργασίας, αντικαθίσταται το animation με προσομοίωση της κίνησης των πλανητών σύμφωνα με τις νευτώνειες βαρυτικές δυνάμεις. Σε αυτή την προσομοίωση πρέπει να συμπεριληφθεί και η κρούση της σελήνης με έναν αστεροειδή και προσομοίωση του συστήματος μετά από αυτό το γεγονός.

Τέλος, αναπαρίστανται όλοι οι πλανήτες του ηλιακού συστήματος.

Η διαδοχή των παραγράφων ακολουθεί τις ενέργειες που υλοποιήθηκαν για να δημιουργηθεί το τελικό παραδοτέο και όχι ακριβώς την διαδοχή των ερωτημάτων. Παρ’ όλα αυτά δεν έχει παραληφθεί κανένα ερώτημα.

2. Μέρος Α’

2.1) Αρχικοποίηση του VS Project.

Για την επίλυση του Α μέρους δημιουργήθηκε Project στο Visual Studio με τα εξής εκτελέσιμα αρχεία (και headers όπου είναι απαραίτητα).

α) Δημιουργήθηκε ενιαίο εκτελέσιμο αρχείο και εκτελέσιμα αρχεία για τους shaders με παρόμοια μορφή και φιλοσοφία όπως στο εργαστήριο στον φάκελο /apallaktikh :

apallaktikh.cpp

StandardShading.fragmentshader

StandardShading.vertexshader

β) Χρησιμοποιήθηκαν εκτελέσιμα αρχεία που προστέθηκαν μέσω headers στο ενιαίο εκτελέσιμο τα οποία βρίσκονται στον φάκελο /common :

Αρχεία του εργαστηρίου :

camera.cpp	camera.h
ModelLoader.cpp	ModelLoader.h
shader.cpp	shader.h
texture.cpp	texture.h
util.cpp	util.h

Αρχεία που δημιουργήθηκαν :

planetPosition.cpp	planetPosition.h
drawPlanets.cpp	drawPlanets.h

γ) Προστέθηκαν όλες οι απαραίτητες βιβλιοθήκες στον φάκελο /external

δ) όλες οι εικόνες για απόδοση υφής καθώς και το αρχείο object που χρησιμοποιείται για πρότυπο κάθε πλανήτη βρίσκονται στον φάκελο /apallaktikh μιας και είναι το directory του Project.

ε) Όλα τα προηγούμενα προστέθηκαν στο αρχείο CMakeLists.txt για να δημιουργηθεί το Project στο Visual Studio με το CMake.

Πολλά κομμάτια των εκτελέσιμων αρχείων βρίσκονται σε αντιστοιχία με το εργαστήριο και για αυτό δεν αναλύονται αρκετά.

2.2) Δημιουργία συστήματος - απόδοση υφής :

Η απεικόνιση των πλανητών γίνεται με την χρήση instances του ίδιου object. Για να μετασχηματιστεί αυτό το object στον κάθε πλανήτη εφαρμόζονται ομογενείς μετασχηματισμοί και αποδίδεται κατάλληλη υφή ανάλογα το instance. Οι μετασχηματισμοί υλοποιούνται στην συνάρτηση DrawingPlanets(…) που δημιουργήθηκε και η οποία αναλύεται στην παράγραφο 2.3, για αυτό δεν αναλύονται περεταίρω σε αυτή την παράγραφο.

Για την απόδοση υφής δημιουργήθηκαν pointers στις εικόνες που χρησιμοποιήθηκαν στο κύριο εκτελέσιμο αρχείο :

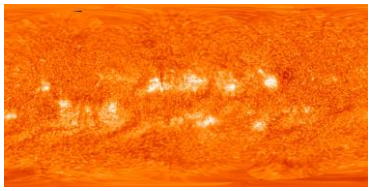
```
GLuint diffuseTextureEarth, specularTextureEarth, diffuseTextureSun, specularTextureSun,
       diffuseTextureBackground, specularTextureBackground, diffuseTextureMoon, specularTextureMoon;
```

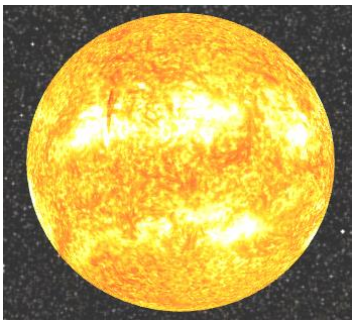
```
//load diffuse texture maps
diffuseTextureEarth = loadSOIL("earth_diffuse.jpg");
specularTextureEarth = loadSOIL("earth_diffuse.jpg");
diffuseTextureSun = loadSOIL("Map_of_the_full_sun.jpg");
specularTextureSun = loadSOIL("Map_of_the_full_sun.jpg");
diffuseTextureBackground = loadSOIL("Background.jpg");
specularTextureBackground = loadSOIL("Background.jpg");
diffuseTextureMoon = loadSOIL("moon.jpg");
specularTextureMoon = loadSOIL("moon.jpg");
```

κώδικας για φόρτωση εικόνων για απόδοση υφής

Οι εικόνες που χρησιμοποιήθηκαν για απόδοση υφής στο σύστημα γη - ήλιος - σελήνη καθώς και το αποτέλεσμα στο γραφικό περιβάλλον φαίνονται παρακάτω (ο κώδικας των shaders στο παράρτημα μιας και έχει υλοποιηθεί σε αντίστοιχο εργαστήριο!) :

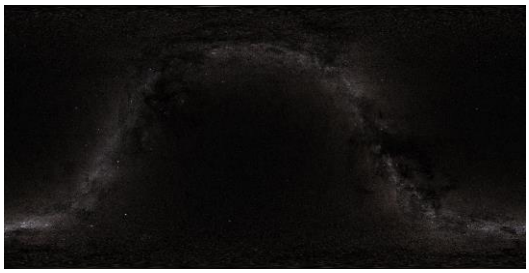
γη : ήλιος : φεγγάρι :


Για τον περιβάλλοντα χώρο χρησιμοποιήθηκε και πάλι το ίδιο object με ένα πολύ μεγάλο scaling έτσι ώστε να βρισκόμαστε πάντοτε στο εσωτερικό του. Αφού υλοποιήθηκε αυτό έγινε απόδοση υφής έτσι ώστε να καταλήγουμε στο επιθυμητό αποτέλεσμα. Η εικόνα για την απόδοση υφής καθώς και το αποτέλεσμα φαίνονται στην επόμενη σελίδα.

texture map : εφαρμογή :




Σε όλες τις παραπάνω εικόνες που αφορούν την απόδοση υφής παρατηρείται διαφορά στο επίπεδο φωτεινότητας μετά την απόδοση. Αυτό οφείλεται στις παραμέτρους του αλγόριθμου φωτισμού και επιλέχθηκαν με τέτοιο τρόπο έτσι ώστε το τελικό αποτέλεσμα να είναι εκτός από ρεαλιστικό, όσο το δυνατόν πιο ‘κομψό’ για το μάτι.

Μία σημαντική παρατήρηση ακόμα, όλες οι φωτογραφίες για απόδοση υφής έχουν ληφθεί με 360^o κάμερα έτσι ώστε να υπάρχει όσο το δυνατόν ελάχιστη παραμόρφωση με την χρήση τους πάνω στο object.

Για να μην ξεφεύγει η κάμερα εκτός των ορίων του background και να βλέπουμε κάτι ανούσιο έχει προστεθεί όριο στην περιήγηση μέσα στο εικονικό περιβάλλον, στο έτοιμο αρχείο camera.cpp από το εργαστήριο όπως φαίνεται παρακάτω :

```
position = clamp(position, vec3(-20.0, -20.0, -20.0), vec3(20.0, 20.0, 20.0));
```

Αυτή η εντολή προστέθηκε πριν τον υπολογισμό του view matrix και χρησιμοποιείται έτσι ώστε να περιορίζει τις δυνατές θέσεις που μπορεί να βρεθεί ο παρατηρητής στο world space εσωτερικά ενός κύβου με κέντρο το 0_{1x3} (θέση ήλιου στο project) και πλευράς 20.

Τα όρια $[-20, 20]$ είναι αρκετά μιας και οι μονάδες που χρησιμοποιεί η OpenGL έχουν κανονικοποιηθεί σε astronomical units (εξηγούνται στο παράρτημα) έτσι ώστε να μπορούμε να χειριζόμαστε τις αποστάσεις με ‘λογικούς’ αριθμούς για το προγραμματιστικό περιβάλλον και τις έτοιμες εντολές της glm βιβλιοθήκης. Μόνο οι πλανήτες στο animation είναι υπο μεγέθυνση σε σχέση με το πραγματικό τους μέγεθος έτσι ώστε να φαίνονται καλύτερα.

Η απόδοση φωτισμού έχει γίνει με διαφορετικό τρόπο στους πλανήτες σε σχέση με τον ήλιο και το background κάτι που περιγράφεται στην παράγραφο 2.3 (β).

2.3) Animation :

Για το animation χρησιμοποιήθηκαν μαθηματικές προσεγγίσεις για την αναλυτική σχέση θέσης και χρόνου. Ο όρος προσεγγίσεις χρησιμοποιείται επειδή είναι αδύνατο να λυθούν αναλυτικά οι εξισώσεις κίνησης των πλανητικών σωμάτων λόγω της χαοτικής συμπεριφοράς που έχουν οι διαφορικές εξισώσεις που διέπουν το ηλιακό σύστημα σε σχέση με τις αρχικές τιμές και τις αστρονομικές μετρήσεις που μπορούν να υλοποιηθούν ανά πάσα στιγμή. Έτσι, χρησιμοποιήθηκαν προσεγγίσεις σύμφωνα με κάποια πρότυπα που έχουν αναπτυχθεί.

Η γενική μεταβλητή που προσομοιώνει τον χρόνο έχει αντιστοιχία σε μέρες δηλαδή, η αύξηση κατά μία μονάδα της μεταβλητής συνεπάγεται αύξηση μίας μέρας στην προσομοίωση.

Γενικά, στο project ο όρος ηλιοκεντρικές συντεταγμένες είναι ίδιος με τις συντεταγμένες στο world space μιας και ο ήλιος βρίσκεται πάντα ακίνητος στο σημείο 0_{1x3}.

α) Αλγόριθμοι κίνησης :

Για την θέση της γης χρησιμοποιήθηκε ο αλγόριθμος VSOP (παράρτημα για παραπάνω πληροφορίες) μέσω του οποίου μπορούν να υπολογιστούν με αρκετή ακρίβεια οι ηλιοκεντρικές καρτεσιανές συντεταγμένες της γης οποιαδήποτε στιγμή. Οι αρχικές τιμές του αλγόριθμου στην περίπτωση του project είναι μετρήσεις από την ημερομηνία J2000 (παράρτημα).

Ένα πρακτικό πρόβλημα που δημιουργείται είναι ότι αυτή η προσέγγιση χρησιμοποιεί ένα πολύ μεγάλο άθροισμα τριγωνομετρικών εξισώσεων από μηδενική έως πέμπτη τάξη ως προς τον χρόνο. Επειδή με την χρήση όλων των τριγωνομετρικών εξισώσεων η εκτέλεση άρχιζε να κινείται εκτός των νοητών ορίων του real time animation χρησιμοποιήθηκαν οι όροι μέχρι επτά σημαντικά ψηφία για τον υπολογισμό της θέσης.

Επίσης, αυτός ο αλγόριθμος χρησιμοποιεί το Ιουλιανό ημερολόγιο, για αυτό και η μεταβλητή που προσομοιώνει τον χρόνο στην mainLoop() μετατρέπεται πριν χρησιμοποιηθεί.

Ο αλγόριθμος, με όσες συνιστώσες του χρησιμοποιήθηκαν, υλοποιήθηκε στο εκτελέσιμο planetPosition.cpp. Η συνάρτηση που επιστρέφει την θέση της γης υπό μορφή διανύσματος είναι η vec3 EarthPos(float t){…} η οποία χρησιμοποιεί τις τιμές που υπολογίζονται στις συναρτήσεις float Earth_X0(float t){…} , . . . , float Earth_Z5(float t){…} που υπολογίζουν για κάθε συντεταγμένη και τάξη του χρόνου την θέση της γης. Οι συντεταγμένες που υπολογίζονται και επιστρέφονται είναι ήδη σε astronomical units.

Η περιστροφή της γης γύρω από τον εαυτό της καθώς και η γωνία του άξονα περιστροφής, υπολογίζονται στην mainLoop() και χρησιμοποιούνται ως ορίσματα στην void DrawingPlanets(…) του εκτελέσιμου drawPlanets.cpp. Αυτή η συνάρτηση υπολογίζει τον model matrix σύμφωνα με όλα τα προηγούμενα και στέλνει τα δεδομένα στους shaders αλλά αναλύεται στην παράγραφο (δ) γιατί υλοποιεί πολλές ενέργειες.

Σύμφωνα λοιπόν με τους παραπάνω υπολογισμούς το ολικό pose του αντικειμένου που υπολογίζεται είναι το εξής :

```
// compute model matrix
mat4 modelMatrix  = glm::translate(mat4(), position)*
    glm::rotate(mat4(), tilt, vec3(0, 0, 1))*
    glm::rotate(mat4(), rotation, vec3(0, 1, 0))*
    glm::scale(mat4(), vec3(size, size, size));
```

Position είναι η θέση του πλανήτη, tilt είναι η γωνία του άξονα περιστροφής, rotation είναι η γωνία περιστροφής γύρω από τον εαυτό του πλανήτη και scale είναι το μέγεθος του πλανήτη

Αυτός ο υπολογισμός γίνεται propagate στους shaders και χρησιμοποιείται για όλους τους υπολογισμούς.

Η γωνία του άξονα περιστροφής (tilt) που χρησιμοποιείται στο project είναι σταθερή και ίση με 23.4^{o}. Γενικά, η γωνία αυτή δεν είναι σταθερή αλλά μεταβάλλεται με πολύ αργό ρυθμό και έχει πολύ μικρή διακύμανση. Έτσι μπορούμε πρακτικά να την πάρουμε ίση με την τιμή που έχει αυτή την στιγμή που είναι 23.4^{o}.

Η μεταβλητή position υπολογίζεται μέσω της vec3 EarthPos(float t){…}.

Η μεταβλητή rotation που υπολογίζεται στην mainLoop() ισούται με $\pi * time$ αφού η μεταβλητή που χρησιμοποιείται για τον χρόνο είναι σε αντιστοιχία με τις μέρες και κάθε μία μέρα πρέπει να εκτελείται μία περιστροφή.

Για το φεγγάρι χρησιμοποιήθηκε ένας πιο άμεσος αλγόριθμος υπολογισμού της θέσης ο οποίος λαμβάνει υπόψιν του στοιχεία μιας Keplerian ελλειπτικής τροχιάς όπως η εκκεντρότητα, η ανωμαλία κλπ. (παράρτημα). Αυτός ο αλγόριθμος επιστρέφει γεωκεντρικές καρτεσιανές συντεταγμένες και χρησιμοποιεί ως είσοδο του έναν αυξητικό τρόπο υπολογισμού του χρόνου σε μέρες μετρώντας από το J2000.

```
glm::vec3 MoonPos(float t)
{
    /*
    N = longitude of the ascending node
    i = inclination to the ecliptic (plane of the Earth's orbit)
    w = argument of perihelion
    a = semi-major axis, or mean distance from Sun
    e = eccentricity (0=circle, 0-1=ellipse, 1=parabola)
    M = mean anomaly (0 at perihelion; increases uniformly with time)
    w1 = N + w   = longitude of perihelion
    L  = M + w1  = mean longitude
    q  = a*(1-e) = perihelion distance
    Q  = a*(1+e) = aphelion distance
    P  = a ^ 1.5 = orbital period (years if a is in AU, astronomical units)
    T  = Epoch_of_M - (M(deg)/360_deg) / P  = time of perihelion
    v  = true anomaly (angle between position and perihelion)
    E  = eccentric anomaly
    */
    float N = radians(125.1228 - 0.0529538083 * t);
    float i = radians(5.1454);
    float w = radians(318.0634 + 0.1643573223 * t);
    float a = 0.25;  //(Earth - moon radii)
    float e = 0.054900;
    float M = radians(115.3654 + 13.0649929509 * t);
    float E = radians(M + e *sin(M)*(1.0 + e*cos(M)));
    float xv = a*(cos(E) - e);
    float yv = a*(sqrt(1.0 - pow(e, 2))*sin(E));
    float v = degrees(atan2(yv, xv));
    float r = sqrt(pow(xv, 2) + pow(yv, 2));
    return vec3(r*(cos(N)*cos(v + w) - sin(N)*sin(v + w)*cos(i)),
        r * (sin(v + w) * sin(i)),
        r*(sin(N)*cos(v + w) + cos(N)*sin(v + w)*cos(i)));
}
```

Ο αλγόριθμος υλοποιήθηκε στην συνάρτηση vec3 MoonPos(float t){…} στο εκτελέσιμο planetPosition.cpp.

Ο αλγόριθμος που προγραμματίσθηκε φαίνεται δίπλα ενώ η αναλυτική του μορφή περιγράφεται στο παράρτημα.

Για την περιστροφή του φεγγαριού γύρω από τον εαυτό του χρησιμοποιήθηκε το γεγονός ότι υπάρχει το dark side of the moon. Δηλαδή, πάντα είναι μόνο μία πλευρά ορατή από την γη. Έτσι, βρέθηκε η συχνότητα περιστροφής του φεγγαριού γύρω από τον εαυτό του.

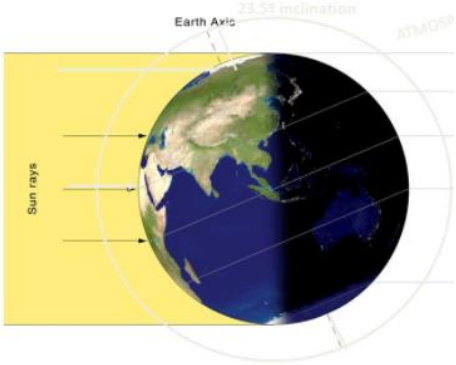
Η τελευταία μεταβλητή που χρειαζόμαστε για το pose του φεγγαριού είναι η γωνία του άξονα περιστροφής έτσι ώστε να κληθεί η συνάρτηση void DrawingPlanets(…). Όπως και με την γη, η γωνία που έχει ως άξονα περιστροφής γύρω από τον εαυτό του το φεγγάρι είναι πρακτικά σταθερή και ίση -5^{o}.

Οι γεωκεντρικές συντεταγμένες του φεγγαριού μετατρέπονται σε ηλιοκεντρικές προσθέτοντας το διάνυσμα θέσης της γης στις συντεταγμένες αυτού στην mainLoop() πριν γίνουν υπολογισμοί.

```
vec3 moonP = MoonPos(time) + earthP;
```

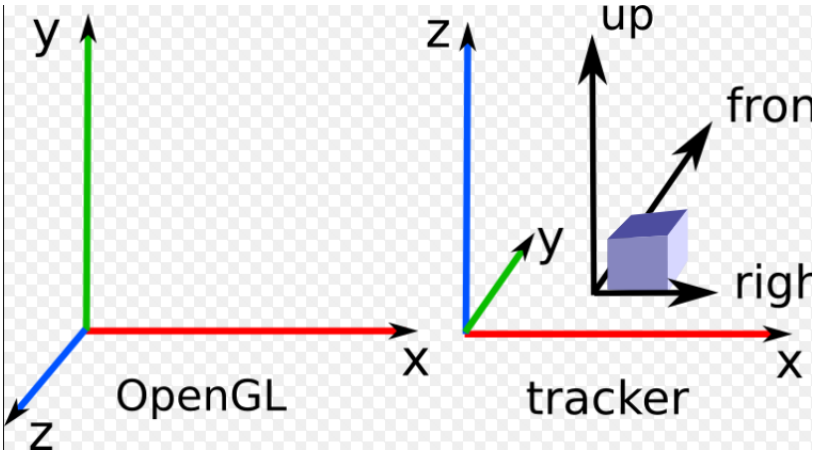
Οι παραπάνω τιμές χρησιμοποιούνται στην void DrawingPlanets(…) για να υπολογισθεί ο model matrix του φεγγαριού όπως και στην περίπτωση της γης.

Οι γωνίες που χρησιμοποιούνται έχουν ως επίπεδο αναφοράς το επίπεδο που δημιουργείται μεταξύ των κέντρων των τροχιών της γης και του φεγγαριού και της θέσης του κέντρου του ήλιου. Το normal διάνυσμα πάνω σε αυτή την επιφάνεια μας δείχνει τον άξονα περιστροφής από όπου υπολογίζεται η γωνία του άξονα περιστροφής κάθε ηλιακού σώματος γύρω από τον εαυτό του όπως φαίνεται και στην εικόνα δίπλα. Το normal διάνυσμα για να έχει σωστή φορά επίσης, πρέπει να έχει την ίδια φορά με την φορά της στροφορμής εάν το σύστημα περιστρεφόταν δεξιόστροφα.



Για τον ήλιο και το background δεν χρειάζεται να γίνει κάποιο animation έτσι χρησιμοποιούμε σταθερές τιμές για τον υπολογισμό του model matrix αντίστοιχα.

Τέλος χρησιμοποιήθηκαν κάποια αρνητικά πρόσημα στις συναρτήσεις του planetPosition.cpp. Αυτό γίνεται διότι χρησιμοποιούμε τον z άξονα της OpenGL αντί του y άξονα του καρτεσιανού συστήματος για απεικόνιση των αποτελεσμάτων και αφού αυτοί οι άξονες έχουν αντίθετη φορά προστίθεται ένα αρνητικό πρόσημο.



β) Αλγόριθμοι φωτισμού – σκίασης

Η απόδοση χρώματος γίνεται στον fragmentShader. Ο κύριος αλγόριθμος φωτισμού που υλοποιείται είναι ο αλγόριθμος του Phong. Δεν θα γίνει ανάλυση του αλγόριθμου εδώ μιας και έχει χρησιμοποιηθεί στο αντίστοιχο εργαστήριο και έχει υλοποιηθεί με τον ίδιο ακριβώς τρόπο (παράρτημα).

Με τον αλγόριθμο του Phong αποδίδεται χρώμα σε όλα τα πλανητικά σώματα εκτός από τον ήλιο και το background.

Για τον ήλιο και το background έχει χρησιμοποιηθεί φως σταθερής έντασης μιας και είναι πηγές φωτός. Ο ήλιος περικλείει την κύρια πηγή φωτός που δημιουργεί όλα τα χαρακτηριστικά του φωτός στους πλανήτες.

Το background στην πραγματικότητα είναι πολλές και αμυδρές πηγές φωτός. Κάθε μακρινό άστρο θα έπρεπε να συμπεριφέρεται ως πηγή φωτός στο εικονικό περιβάλλον. Για αυτό το λόγο αντιμετωπίζεται σαν μία ενιαία πηγή φωτός (όπως δηλαδή και ο ήλιος).

Για να γίνει ο διαχωρισμός των περιπτώσεων δημιουργείται μία uniform μεταβλητή στον fragmentshader η οποία ξεχωρίζει αυτές τις περιπτώσεις.

```
uniform float textureCase;
```

Αυτή η μεταβλητή λαμβάνει τιμή μέσω της συνάρτησης DrawingPlanets(…), που καλείται στην mainLoop(), και έχει γνώση της τοποθεσίας της uniform μεταβλητής, μιας και στο apallaktikh.cpp κώδικα αρχικοποιείται ως global μεταβλητή.

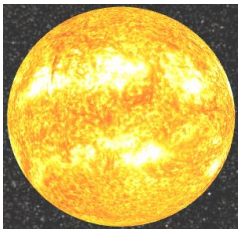
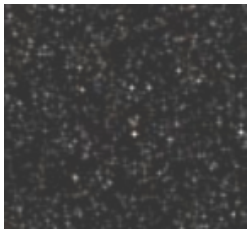
Με χρήση αυτής της μεταβλητής λοιπόν διαχωρίζονται οι περιπτώσεις με το εξής μπλοκ κώδικα στον fragmentshader :

```
// sun or background
if(textureCase>0){
    Is = Ls * Ks;
    Id = Ld * Kd;
    Ia = vec4(0.2, 0.2, 0.2, 1.0);
    l_power = 0.7;
};
```

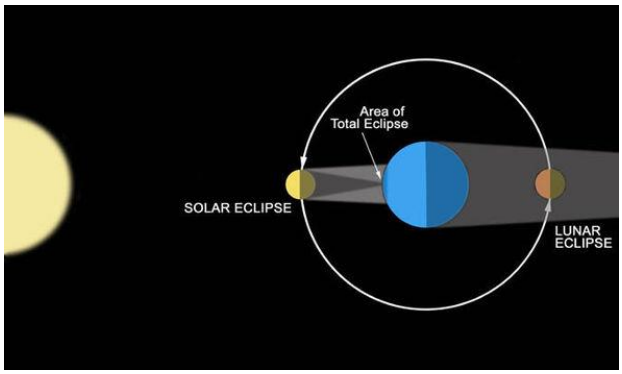
Όταν εκτελείται αυτό το κομμάτι κώδικα ο αλγόριθμος του Phong έχει ήδη εκτελεστεί και οι μεταβλητές Is, Id, Ia και l_power έχουν ήδη πάρει τιμή. Αν όμως, πρόκειται για σχεδιασμό του ήλιου ή του background αυτές οι τιμές αντικαθίστανται με τις τιμές που φαίνονται στην εικόνα. Οι τιμές αυτές επιλέχθηκαν με την λογική trial and error έως ότου το αποτέλεσμα στο εικονικό περιβάλλον να είναι όσο το δυνατόν πιο θελκτικό.

Ο αλγόριθμος του Phong δεν προσδίδει φως σε vertices όπου το normal διάνυσμα δείχνει προς αντίθετη μεριά σε αντίθεση με την υλοποίηση για τον ήλιο και το background.

Phong algorithm : Light source : Background :

Πριν όμως αποδοθεί χρώμα στον fragmentshader γίνεται ακόμα ένας έλεγχος. Όταν τα τρία πλανητικά σώματα ευθυγραμμίζονται τότε υπάρχει η περίπτωση η γη να σκιάζει την σελήνη ή και το αντίθετο.

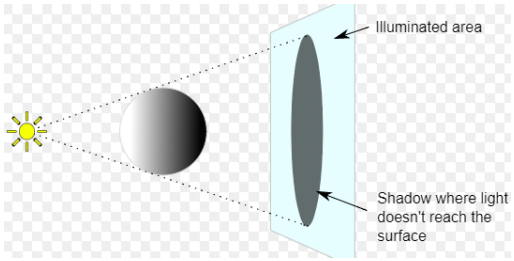


Στην εικόνα δίπλα φαίνονται και οι δύο περιπτώσεις. Όταν η σελήνη βρίσκεται ανάμεσα στην γη και τον ήλιο σκιάζεται η γη, ενώ όταν βρίσκεται η γη ανάμεσα στον ήλιο και την σελήνη σκιάζεται η σελήνη.

Στην περίπτωση του project ο ήλιος χειρίζεται ως σημειακή πηγή φωτός και οι πράξεις απλοποιούνται εν μέρει.

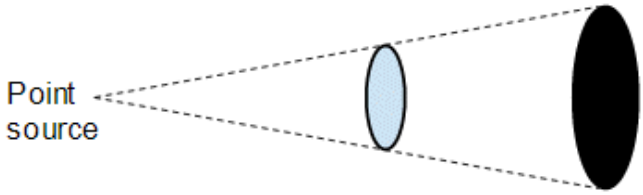
Αν λοιπόν αντιμετωπίσουμε τον ήλιο σαν μία σημειακή φωτός μπορούμε να ορίσουμε το πολύγωνο σκιάς με μία μόνο μεταβλητή λόγω της σφαιρικής φύσης των σωμάτων που την δημιουργούν.

Αρχικά, πρέπει να γίνει κατανοητό το πολύγωνο σκιάς που δημιουργεί μία σφαίρα :



Στην πραγματικότητα δεν πρόκειται για πολύγωνο αλλά για έναν κώνο στον χώρο.

Η ίδια συμπεριφορά μπορεί να μοντελοποιηθεί και με το εξής σχήμα που φαίνεται δεξιά :



Τα σημεία εσωτερικά του κώνου έχουν μία κοινή ιδιότητα που μπορεί να τα περιγράψει ανεξαρτήτως αν ο κώνος δημιουργείται από σφαίρα ή δίσκο μόνο που ο δίσκος χρησιμοποιήθηκε γιατί φαίνεται πιο εύκολα αυτή η ιδιότητα.

Έστω λοιπόν Ο το σημείο της πηγής φωτός, Α το κέντρο του δίσκου (ή της σφαίρας), Β ένα σημείο της περιφέρειας του δίσκου (στην σφαίρα αυτά τα σημεία είναι αυτά που η ακτίνα φωτός είναι εφαπτόμενη στην επιφάνεια) και Γ οποιοδήποτε σημείο εντός του κώνου. Τότε για όλα τα σημεία Γ εσωτερικά του κώνου ισχύει ότι :

$$|\widehat{\overrightarrow{OA},\overrightarrow{OB}}| \geq |\widehat{\overrightarrow{OA},\overrightarrow{O\Gamma}}| \qquad \textit{(σχέση 1)}$$

Δηλαδή, η στερεά γωνία που σχηματίζει κάθε σημείο εσωτερικό του κώνου με το διάνυσμα θέσης της σφαίρας, πρέπει να είναι μικρότερο κατά απόλυτη τιμή από την

στερεά γωνία που δημιουργεί το διάνυσμα θέσης της σφαίρας με ένα σημείο της περιφέρειας αυτής όπου η ακτίνα φωτός είναι εφαπτόμενη.

Στην πράξη ο αλγόριθμος υλοποιήθηκε σε δύο μέρη. Αρχικά, οι πρώτοι υπολογισμοί έγιναν στην συνάρτηση void ComputeShadowPrism(vec3 EarthPos, vec3 MoonPos,int planet){…} στο εκτελέσιμο DrawPlanets.cpp, και ότι χρειάζεται γίνεται propagate στον fragmentshader μέσω uniform μεταβλητών για να γίνουν οι υπόλοιπες πράξεις του αλγορίθμου.

Πιο αναλυτικά, η συνάρτηση :

```
void ComputeShadowPrism(vec3 EarthPos, vec3 MoonPos,int planet) {
    float earthShadowPrism = 0.0;
    float moonShadowPrism = 0.0;
    // Euler angle for earth
    if (length(MoonPos) <= length(EarthPos)) {
        earthShadowPrism = length(MoonPos) /sqrt( pow(length(MoonPos),2)+pow(0.01,2));
    };
    // Euler angle for moon
    if (length(MoonPos) >= length(EarthPos)) {
        moonShadowPrism = length(EarthPos) / sqrt(pow(length(EarthPos), 2) + pow(0.05, 2));
    };

    switch (planet) {
        case earth:
            glUniform1f(ShadowL, earthShadowPrism);
            glUniform3f(centerL, MoonPos.x, MoonPos.y, MoonPos.z);
            glUniform3f(center2L, EarthPos.x, EarthPos.y, EarthPos.z);
            break;
        case moon:
            glUniform1f(ShadowL, moonShadowPrism);
            glUniform3f(centerL, EarthPos.x, EarthPos.y, EarthPos.z);
            glUniform3f(center2L, MoonPos.x, MoonPos.y, MoonPos.z);
            break;
        default:
            glUniform1f(ShadowL, 0.0f);
            break;
    };
}
```

Η συνάρτηση δέχεται ως ορίσματα την θέση της γης, την θέση του ήλιου και για ποιον πλανήτη θέλουμε να υπολογίσουμε την ύπαρξη ή όχι σκιάς. Αρχικά, η γωνία του κώνου που δημιουργεί η σκιά ενός πλανήτη είναι μηδενική.

Οι έλεγχοι if υλοποιούνται για να δούμε ποιος πλανήτης βρίσκεται ενδιάμεσα από τον άλλον και τον ήλιο. Αν ο έλεγχος ισχύει, τότε υπολογίζεται το συνημίτονο της στερεάς γωνίας που υπολογίζει τον κώνο. Δεν χρειάζεται να βρούμε το εφαπτόμενο σημείο στην ακτίνα του ηλίου για δύο λόγους. Αρχικά, επειδή ο ήλιος έχει μεγαλύτερη ακτίνα από την γη, το εφαπτόμενο σημείο βρίσκεται κάπου στην περιφέρεια αλλά πρακτικά μπορούμε να πάρουμε το σημείο πάνω (ή κάτω) από το κέντρο της σφαίρας μιας και τα λάθη αυτά είναι αμελητέα σε σχέση με το μέγεθος των πλανητών και τις αποστάσεις. Επίσης, αφού χρησιμοποιούμε το συνημίτονο, ο τρόπος υπολογισμού της υποτείνουσας εμπεριέχει την χρήση του εφαπτόμενου σημείου στην ακτίνα του ηλίου.

Αφού λοιπόν υπολογιστεί η γωνία γίνεται propagate αυτή η γωνία μαζί με την θέση των κέντρων των δύο πλανητών όπου δημιουργείται και αποτυπώνεται η σκιά αντίστοιχα στον fragmentshader μέσω uniform μεταβλητών.

Όταν δεν υπάρχει καμία αλληλεπίδραση των σωμάτων η στερεά γωνία που γίνεται propagate είναι μηδενική (χρησιμοποιείται σαν έλεγχος στον shader!).

Το δεύτερο κομμάτι του αλγόριθμου, *(σχέση 1)* υπολογίζεται στον fragmentshader. Όλα τα απαραίτητα στοιχεία για τον υπολογισμό του τελευταίου κομματιού του αλγορίθμου έχουν γίνει ήδη propagate :

```
uniform float Shadow;
uniform vec3 center;
uniform vec3 center2;
```

Shadow είναι η στερεά γωνία του κώνου, center / center2 είναι τα κέντρα των πλανητών που δημιουργείται / αποτυπώνεται η σκιά.

Αφού πάρουν τιμή αυτές οι μεταβλητές και έχουν ήδη υλοποιηθεί οι αλγόριθμοι φωτισμού, τελευταίος έλεγχος πριν την απόδοση τιμής στο χρώμα είναι ο εξής :

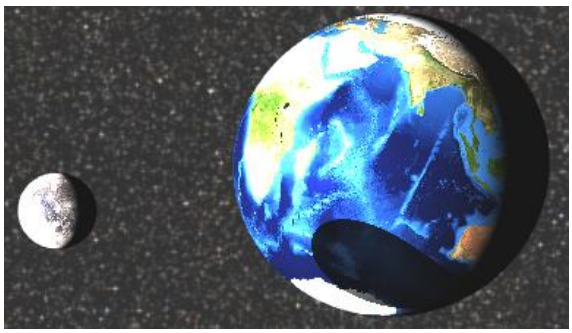
```
if( (abs( dot (normalize( center ) , normalize( center2 + (M*vec4(vertex_position_modelspace,1)).xyz ) ) ) > Shadow) && (Shadow>0) ){
    l_power = l_power/10;
};
```

Επειδή το εσωτερικό γινόμενο υπολογίζει το ημίτονο της γωνίας μεταξύ διανυσμάτων, η ανισότητα στην *(σχέση 1)* αλλάζει φορά. Όμως, για να υπολογίζεται σκιά μόνο όταν είναι αυτό απαραίτητο, χρησιμοποιείται και έλεγχος για το αν υπάρχει στερεά γωνία σκιάς (στην περίπτωση του ήλιου δεν υπάρχει).

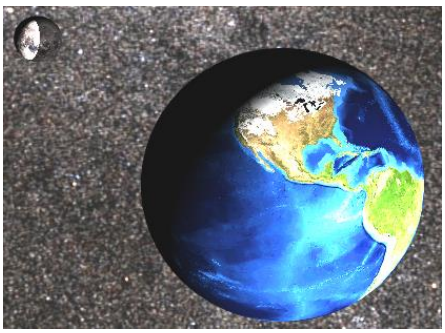
Αν ο έλεγχος για σκιά επαληθευτεί, τότε μειώνουμε την ένταση του φωτός, χωρίς όμως να την μηδενίζουμε μιας και πρακτικά λόγω του μεγαλύτερου μεγέθους του ήλιου από την γη και την σελήνη πάντα υπάρχει δρόμος για το φως σχεδόν σε όλα τα σημεία του πλανήτη που είναι στραμμένα προς τον ήλιο (όπως είναι σπάνια η πλήρη έκλειψη ηλίου).

Τα αποτελέσματα της επίδρασης αυτού του αλγορίθμου στο εικονικό περιβάλλον φαίνονται παρακάτω για την γη και την σελήνη :

Solar eclipse :



Lunar eclipse :



γ) Περιήγηση στο εικονικό περιβάλλον

Για την περιήγηση στο εικονικό περιβάλλον χρησιμοποιήθηκε ο κώδικας camera.cpp με την έτοιμη κλάση camera από το εργαστήριο. Αυτό το αρχείο υπολογίζει τον view matrix και τον projection matrix για χρήση τους στον vertexshader. Επίσης, σε αυτό το αρχείο γίνεται poll για χρήση οποιουδήποτε πλήκτρου για περιήγηση στο εικονικό περιβάλλον. (περισσότερες λεπτομέρειες στο παράρτημα)

Μόνη αλλαγή στον κώδικα είναι η χρήση ορίων για την κάμερα που έχει ήδη αναλυθεί στην παράγραφο 2.2 έτσι ώστε να βρισκόμαστε πάντα εντός των ορίων του φόντου.

δ) Κλήση των αλγορίθμων - ροή προγράμματος

Σε αυτή την παράγραφο αναλύεται η κλήση όλων των μπλοκ κώδικα που συνεισφέρουν στο animation κομμάτι (μόνο, όχι στην προσομοίωση!) της απαλλακτικής εργασίας. Κύρια λογική όλου του project είναι η δομημένη και κατανοητή ροή του κώδικα και κλήση των αλγορίθμων που υπάρχουν εκτός του κυρίως κώδικα ώστε να αποφευχθεί ο χαοτικός προγραμματισμός και να γίνει δομημένο το project.

Ο έλεγχος ροής της εργασίας βρίσκεται στο κύριο εκτελέσιμο \apallaktikh\apallaktikh.cpp. Αυτός ο κώδικας αποτελείται από έξι βασικά μέρη τα οποία αναλύονται με την σειρά που συναντώνται στον κώδικα.

1) Στις πρώτες γραμμές του κώδικα ορίζονται όλα τα header αρχεία για βιβλιοθήκες (/external) και εκτελέσιμα αρχεία (/common), τα πρότυπα των τεσσάρων συναρτήσεων που καλούνται στην main(), οι global μεταβλητές που χρησιμοποιούνται σε όλο το project καθώς και κάποια macro ονόματα (planet, TITLE κλπ.).

2) Η συνάρτηση void createContext() κάνει τρεις σημαντικές εργασίες. Αρχικά, φορτώνει τα προγράμματα των shaders και το object που χρησιμοποιείται στην συνέχεια του project. Ύστερα, αποδίδει τιμή σε όλους τους global pointers που ‘δείχνουν’ είτε σε κάποια εικόνα στο directory του project, είτε σε κάποια uniform μεταβλητή που υπάρχει στους shaders. Τέλος, δημιουργούνται οι κατάλληλοι buffers για να διοχετευτεί το αντικείμενο μέσω layout στους shaders.

3) Η συνάρτηση void free() διαγράφει από την μνήμη όλους τους buffers και pointers που δημιουργήθηκαν από την void createContext().

4) Η συνάρτηση void mainLoop() είναι η λούπα η οποία καλεί όλους τους αλγορίθμους που αναλύθηκαν παραπάνω, καλεί την συνάρτηση DrawingPlanets(…) η οποία αναλύεται παρακάτω, καλεί την ComputeShadowPrism(…) και ανανεώνει την μεταβλητή που προσομοιώνει τον χρόνο σε κάθε iteration.

5) Η συνάρτηση void initialize() δημιουργεί το γραφικό παράθυρο (περιβάλλον) και την διεπαφή με αυτό και αρχικοποιεί την κλάση camera με τις ιδιότητες αυτού του παραθύρου.

6) Τέλος, στην main() συνάρτηση καλούνται όλες οι προηγούμενες συναρτήσεις με την σειρά initialize(), createContext(), mainLoop(), free() και αν αποτύχει η κλήση κάποιας συνάρτησης απλά καλεί την free().

Πρακτικά, όλες οι ενέργειες, υπολογισμοί και η σχεδίαση ξεκινούν από την mainLoop(). Σε αυτή την συνάρτηση γίνεται κλήση όλων των συναρτήσεων που έχουν αναπτυχθεί και για αυτό πρέπει να αναλυθεί λίγο η σειρά των υπολογισμών.

Για τις συναρτήσεις που υπολογίζουν την θέση των πλανητών (planetPosition.cpp) έχει προστεθεί το πρότυπό τους στο αντίστοιχο header αρχείο ενώ για τις συναρτήσεις DrawingPlanets() και computeShadowPrism() (drawPlanets.cpp) στο αντίστοιχο header έχουν προστεθεί και οι global μεταβλητές (extern στο header) μιας και αυτές οι δύο συναρτήσεις είναι ο τρόπος επικοινωνίας με τους shaders καθώς όλα τα δεδομένα από την mainLoop() ή ακόμα και το object στέλνονται μέσω αυτών των συναρτήσεων στους shaders.

Η ComputeShadowPrism(…) έχει ήδη αναλυθεί.

Η DrawingPlanets(…) :

```
void DrawingPlanets(mat4 projectionMatrix, mat4 viewMatrix, vec3 position, float rotation, float tilt,
    float size, vec3 lightPos, int caseTexture, GLuint diffuseTexture, GLuint specularTexture)
{

    glBindVertexArray(objVAO);
    // compute model matrix
    mat4 modelMatrix  = glm::translate(mat4(), position)*
        glm::rotate(mat4(), tilt, vec3(0, 0, 1))*
        glm::rotate(mat4(), rotation, vec3(0, 1, 0))*
        glm::scale(mat4(), vec3(size, size, size));

    // transfer uniforms to GPU
    glUniformMatrix4fv(projectionMatrixLocation, 1, GL_FALSE, &projectionMatrix[0][0]);
    glUniformMatrix4fv(viewMatrixLocation, 1, GL_FALSE, &viewMatrix[0][0]);
    glUniformMatrix4fv(modelMatrixLocation, 1, GL_FALSE, &modelMatrix[0][0]);
    glUniform3f(lightLocation, lightPos.x, lightPos.y, lightPos.z); // light
    glUniform1f(TextureCaseLocation, caseTexture);

    // bind textures and transmit diffuse and specular maps to the GPU
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, diffuseTexture);
    glUniform1i(diffuceColorSampler, 0);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, specularTexture);
    glUniform1i(specularColorSampler, 1);

    // draw
    glDrawArrays(GL_TRIANGLES, 0, objVertices.size());

}
```

Τα πρώτα έξι ορίσματα της DrawingPlanets(…) είναι αυτά που χρειάζονται για να υπολογιστούν οι πίνακες model matrix, projection matrix και view matrix. Η επόμενη μεταβλητή είναι η θέση της πηγής φωτός (πάντα 0_{1x3} αφού είναι η θέση του ήλιου), ακολουθεί η μεταβλητή που μας ξεχωρίζει αν θα υλοποιηθεί ο αλγόριθμος του Phong ή όχι στους shaders και τέλος είναι οι pointers στην εικόνα που θα χρησιμοποιηθεί για texture mapping.

Στο body της συνάρτησης αρχικά υπολογίζεται ο model matrix, στην συνέχεια γίνονται propagate όλες οι uniform μεταβλητές πέρα από αυτές που χρησιμοποιούνται για σκίαση από την ComputeShadowPrism() και τέλος γίνονται bind τα textures (το αντικείμενο γίνεται bind στην πρώτη γραμμή του σώματος της συνάρτησης) και καλείται η glDrawArrays(…) ώστε να γίνει το rendering.

Πολλές μεταβλητές χρησιμοποιήθηκαν ως external έτσι ώστε να χρησιμοποιηθούν όσο το δυνατόν λιγότερες μεταβλητές στο πρότυπο της συνάρτησης DrawingPlanets(…).

Αφού αναλύθηκε η λειτουργία και ο τρόπος κλήσης κάθε συνάρτησης θα αναλυθεί η αλληλουχία των υπολογισμών στην mainLoop().

Βήμα 1: Υπολογισμός του χρόνου (ανεξάρτητα της υπολογιστικής δύναμης του μηχανήματος που τρέχει ο κώδικας) :

```
static double lastTime = glfwGetTime();
double currentTime = glfwGetTime();
float deltaTime = float(currentTime - lastTime);
time += deltaTime*scalingFactor;
```

Στο τέλος της λούπας γίνεται και ανανέωση της lastTime.

Βήμα 2: Υπολογισμός των πινάκων projection matrix και view matrix μέσω της συνάρτησης update() της κλάσης camera:

```
// camera
camera->update();
mat4 projectionMatrix = camera->projectionMatrix;
mat4 viewMatrix = camera->viewMatrix;
```

Βήμα 3: Υπολογισμός των ηλιοκεντρικών συντεταγμένων των πλανητών :

```
// Bodies position
vec3 earthP = EarthPos((time - 2451545.0) / 365250.0);
vec3 moonP = MoonPos(time) + earthP;
```

Για την γη, η μεταβλητή time μετατρέπεται σύμφωνα με το Ιουλιανό ημερολόγιο.

Βήμα 4: Κλήση των ComputeShadowPrism() (αν χρειάζεται) και DrawingPlanets() για κάθε πλανήτη χωριστά :

```
// SUN //
ComputeShadowPrism(earthP, moonP, other);
DrawingPlanets(projectionMatrix, viewMatrix, vec3(0.0), 0.0f, 0.0f,
    0.1f, lightPos, sun, diffuseTextureSun, specularTextureSun);

// EARTH //
ComputeShadowPrism(earthP, moonP, earth);
DrawingPlanets(projectionMatrix, viewMatrix, earthP, time*3.14, radians(23.4f),
    0.05f, lightPos, planet, diffuseTextureEarth, specularTextureEarth);

// Moon //
ComputeShadowPrism(earthP, moonP, moon);
float moonRot = radians((27.3)*time) ;
DrawingPlanets(projectionMatrix, viewMatrix, moonP, moonRot, radians(-5.0f),
    0.01f, lightPos, planet, diffuseTextureMoon, specularTextureMoon);

// BACKGROUND //
ComputeShadowPrism(earthP, moonP, other);
DrawingPlanets(projectionMatrix, viewMatrix, vec3(0), 0.0f, 0.0f,
    30.0f, lightPos, background, diffuseTextureBackground, specularTextureBackground);
```

Όταν καλείται η ComputeShadowPrism(…) γίνονται propagate στους shaders τα κέντρα των πλανητών και η στερεά γωνία και ύστερα γίνονται propagate οι υπόλοιπες uniform μεταβλητές καθώς και τα object attributes και τα texture maps μέσω της DrawingPlanets(…).

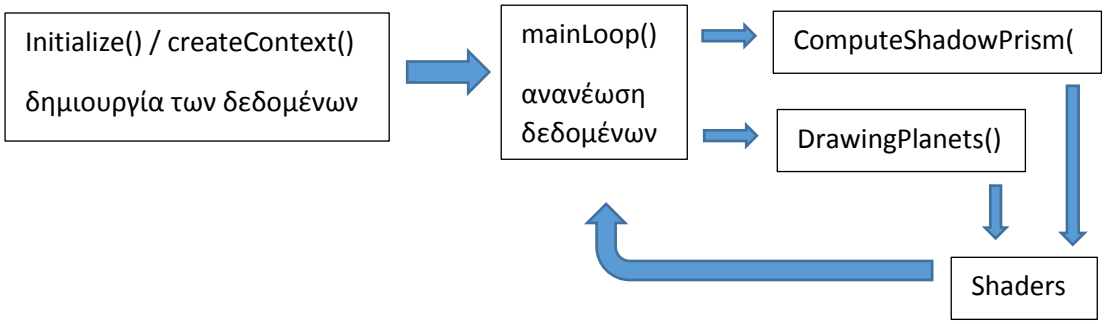
Βήμα 5: Ανανέωση του χρόνου στην λούπα, προβολή του ζωγραφισμένου παραθύρου και poll για κάποια ενέργεια από τον χρήστη :

```
lastTime = currentTime;

glfwSwapBuffers(window);

glfwPollEvents();
```

Η διοχέτευση δεδομένων μέσω των συναρτήσεων συμβολικά φαίνεται παρακάτω :



Η συνάρτηση update() της κλάσης κάμερα καθώς και οι συναρτήσεις EarthPos(…), MoonPos(…) δεν υπάρχουν στο παραπάνω σχεδιάγραμμα διότι δεν γίνεται διακίνηση δεδομένων μέσω αυτών αλλά αυτές οι συναρτήσεις κάνουν την ανανέωση των δεδομένων πριν διακινηθούν.

Οι συναρτήσεις ComputeShadowPrism(), DrawingPlanets() εκτός από υπολογισμούς δημιουργούν και έναν κόμβο ελέγχου για την ροή των δεδομένων από το κυρίως εκτελέσιμο πρόγραμμα (apallaktikh.cpp) μέχρι την εκτέλεση του προγράμματος στους shaders.

Ο παραδοτέος κώδικας περιέχει σχόλια σε όλα τα αρχεία για ότι έχει ειπωθεί μέχρι στιγμής για το μέρος Α της απαλλακτικής εργασίας.

Στο παράρτημα βρίσκονται έννοιες και μπλοκ κώδικα αμελητέα για την γενική εικόνα της απαλλακτικής εργασίας αλλά αποτελούν λίγο πιο τεχνικά θέματα.

Τέλος Μέρους Α'

3. Μέρος Β’

Σε αυτό το μέρος της εργασίας η θέση των πλανητών δεν υπολογίζεται από κάποιον αλγόριθμο κλειστής μορφής αλλά γίνεται δυναμική προσομοίωση με βάση τις βαρυτικές δυνάμεις μεταξύ των πλανητών. Ο υπολογισμός των δυνάμεων, ανεξαρτήτως ερωτήματος, γίνεται για το πλήρες ηλιακό σύστημα. Θεωρήθηκε περιττό να γίνει προσομοίωση για το σύστημα τριών σωμάτων ήλιος - γη - σελήνη αφού είναι πληρέστερη η προσομοίωση με όλους τους πλανήτες του ηλιακού συστήματος.

Η προσομοίωση αφορά μόνο την θέση των πλανητών στον χώρο, μιας και η περιστροφή γύρω από των εαυτό τους αποτελεί θέμα ‘πρακτικά’ αμελητέο στα πλαίσια αυτής της εργασίας.

3.1) Επέκταση των εκτελέσιμων αρχείων :

Για το Β’ μέρος της εργασίας προστέθηκαν στον φάκελο /common τα εξής εκτελέσιμα αρχεία :

RigidBody.h	RigidBody.cpp
Sphere.h	Sphere.cpp
Collision.h	Collision.cpp

Σε αυτά τα αρχεία υπάρχουν οι μέθοδοι που χρησιμοποιούνται για την επίλυση της δυναμικής προσομοίωσης που περιγράφεται στις επόμενες παραγράφους.

3.2) Αρχικοποίηση του συστήματος :

Για την προσομοίωση του ηλιακού συστήματος χρησιμοποιήθηκε η κλάση sphere, με κάθε έναν πλανήτη να είναι ένα στιγμιότυπο αυτής της κλάσης.

```
Sphere* MercuryPhysics;
Sphere* VenusPhysics;
Sphere* EarthPhysics;
Sphere* MoonPhysics;
Sphere* MarsPhysics;
Sphere* JupiterPhysics;
Sphere* SaturnPhysics;
Sphere* UranusPhysics;
Sphere* NeptunePhysics;
```

Η αρχικοποίηση έγινε ως global μεταβλητές έτσι ώστε να χειρίζονται με μεγαλύτερη ευελιξία οι μεταβλητές κατά την προσπέλαση του κώδικα σε διάφορα ερωτήματα.

Στην συνέχεια δημιουργήθηκε μία δομή και τα μέλη της, έτσι ώστε να αποθηκευτούν οι αρχικές τιμές και διάφορες μεταβλητές που χρησιμοποιούνται αργότερα στον κώδικα.

```
struct PlanetsData {
    float PlanetRad;
    double PlanetMass;
    float TrajectoryRad;
    vec3 InitPosition;
    vec3 InitVelocity;
} MercuryS, VenusS, EarthS, MoonS, MarsS, JupiterS,
 SaturnS, UranusS, NeptuneS;
```

Στην παραπάνω δομή αποδόθηκαν τιμές για κάθε μέλος της στην συνάρτηση createContext(). Ενδεικτικά φαίνονται οι τιμές για τον Άρη έτσι ώστε να αναλυθούν κάποια παραπάνω ζητήματα.

```
MarsS.PlanetRad = 0.00002264186;
MarsS.TrajectoryRad = 227939200000.0;
MarsS.InitPosition = vec3(1, 0, 0);
MarsS.InitVelocity = vec3(0, 0, -24007.0);
MarsS.PlanetMass = 6.4171e+23;
```

Η ακτίνα τροχιάς (TrajectoryRad), η αρχική ταχύτητα (InitVelocity) και η μάζα (PlanetMass) δίνονται σε μονάδες S.I. (m, m/s και kg αντίστοιχα) σε αντίθεση με την αρχική θέση που δίνεται κανονικοποιημένη ως προς την τροχιά του πλανήτη αλλά και την ακτίνα που δίνεται σε αστρονομικές μονάδες (a.u.). Στην περίπτωση της αρχικής θέσης, η πραγματική θέση βρίσκεται μέσω πολλαπλασιασμού με την ακτίνα τροχιάς ενώ όπου χρησιμοποιείται η πραγματική ακτίνα μετατρέπεται πολύ εύκολα.

Οι αρχικές τιμές δεν αναπαριστούν κάποια συγκεκριμένη στιγμή του ηλιακού συστήματος στον χρόνο. Αυτές οι τιμές είναι προσεγγιστικές στην μέση τιμή της κάθε μεταβλητής που χρησιμοποιήθηκε. Αυτό μπορεί να υλοποιηθεί διότι λόγω της χαοτικής συμπεριφοράς (Poincare - παράρτημα) ενός τέτοιου συστήματος, οι αρχικές τιμές αρκεί να είναι ‘επαρκής’ έτσι το σύστημα να παραμένει ευσταθές (αριθμητικά αλλά και ποιοτικά γύρω από τις ελλειπτικές τροχιές). Με τον όρο ευσταθές εννοούμε κάθε πλανήτης να μην ξεπερνά μία μέγιστη ταχύτητα (ταχύτητα απόδρασης), με την οποία μπορεί να ξεφύγει από την τροχιά που διαγράφει γύρω από τον ήλιο.

Αφού έχουν υλοποιηθεί τα παραπάνω ο constructor της κλάσης sphere καλείται στην mainLoop() όταν κληθεί το αντίστοιχο ερώτημα.

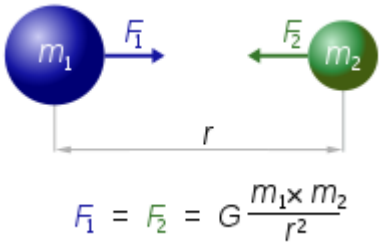
```
//Initialization of Physics
MercuryPhysics = new Sphere(MercuryS.InitPosition*MercuryS.TrajectoryRad, MercuryS.InitVelocity, MercuryS.PlanetRad, MercuryS.PlanetMass);
VenusPhysics = new Sphere(VenusS.InitPosition*VenusS.TrajectoryRad, VenusS.InitVelocity, VenusS.PlanetRad, VenusS.PlanetMass);
EarthPhysics = new Sphere(EarthS.InitPosition*EarthS.TrajectoryRad, EarthS.InitVelocity, EarthS.PlanetRad, EarthS.PlanetMass);
MoonPhysics = new Sphere(MoonS.InitPosition*MoonS.TrajectoryRad, MoonS.InitVelocity, MoonS.PlanetRad, MoonS.PlanetMass);
MarsPhysics = new Sphere(MarsS.InitPosition*MarsS.TrajectoryRad, MarsS.InitVelocity, MarsS.PlanetRad, MarsS.PlanetMass);
JupiterPhysics = new Sphere(JupiterS.InitPosition*JupiterS.TrajectoryRad, JupiterS.InitVelocity, JupiterS.PlanetRad, JupiterS.PlanetMass);
SaturnPhysics = new Sphere(SaturnS.InitPosition*SaturnS.TrajectoryRad, SaturnS.InitVelocity, SaturnS.PlanetRad, SaturnS.PlanetMass);
UranusPhysics = new Sphere(UranusS.InitPosition*UranusS.TrajectoryRad, UranusS.InitVelocity, UranusS.PlanetRad, UranusS.PlanetMass);
NeptunePhysics = new Sphere(NeptuneS.InitPosition*NeptuneS.TrajectoryRad, NeptuneS.InitVelocity, NeptuneS.PlanetRad, NeptuneS.PlanetMass);
MoonPhysics = new Sphere(MoonS.InitPosition*MoonS.TrajectoryRad + EarthS.InitPosition*EarthS.TrajectoryRad,
    MoonS.InitVelocity + EarthS.InitVelocity, MoonS.PlanetRad, MoonS.PlanetMass);
```

Σημείωση : Για το φεγγάρι συγκεκριμένα οι αρχικές τιμές που αποθηκεύτηκαν στην δομή ήταν με κέντρο αναφοράς την γη. Για αυτό λοιπόν, (αγνοώντας την καθαρά αναλυτική μορφή της θεωρίας της σχετικότητας) παίρνουμε πρακτικά τις αρχικές τιμές του φεγγαριού με βάση τον ήλιο προσθέτοντας την ταχύτητα της γης και την θέση της γης σε αυτές. Πρακτικά οι τιμές είναι σωστές (σφάλμα < 1%).

Σημείωση 2 : Στο αρχείο RigidBody.cpp έχει επιλεχθεί η μέθοδος RK-4 μέσω μιας πρακτικής σύμβασης από δοκιμές της προσομοίωσης. Η μέθοδος Euler ήταν αριθμητικά ασταθής σε σύντομο χρονικά διάστημα ενώ η adaptive μέθοδος Euler έριχνε πολύ τον ρυθμό ανανέωσης frames στο παράθυρο.

3.3) Δυναμική προσομοίωση :

Για την προσομοίωση της κίνησης των πλανητών χρησιμοποιήθηκε ο νόμος της παγκόσμιας έλξης του Νεύτωνα :



$$F_1 = F_2 = G\frac{m_1 \times m_2}{r^2}$$

Λόγω της σφαιρικής συμμετρίας που διέπει τα πλανητικά σώματα (πρακτικά οι πλανήτες είναι τέλειες σφαίρες) οι δυνάμεις υπολογίστηκαν σαν δυνάμεις σημειακών μαζών.

Για τον υπολογισμό της δύναμης μεταξύ δύο πλανητών υλοποιήθηκε μία συνάρτηση vec3 ForceCalc(float M1, vec3 Pos1, float M2, vec3 Pos2){. . .} στο αρχείο PlanetPosition.cpp.

```
glm::vec3 ForceCalc(float M1, vec3 Pos1, float M2, vec3 Pos2) {
    return vec3(-6.674e-11*M1*M2*(1 / (length(Pos1 - Pos2)*length(Pos1 - Pos2))))*normalize(Pos1 - Pos2);
}
```

Η συνάρτηση αυτή επιστρέφει την δύναμη μεταξύ δύο σωμάτων ανηγμένη διανυσματικά σε world coordinates.

Για τον υπολογισμό των δυνάμεων λοιπόν πρέπει να κάνουμε $O(n^2)$ υπολογισμούς μεταξύ πλανητικών σωμάτων όπου n ο αριθμός των πλανητών. Οι υπολογισμοί γίνονται στην mainLoop() όπως φαίνεται παρακάτω για την γη ενδεικτικά :

```
EarthPhysics->forcing = [&](float time, const vector<float>& y)->vector<float> {
    vector<float> f(6, 0.0f);

    f[0] = (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, 1.98855e+30, vec3(0.0))).x+
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, MercuryS.PlanetMass, MercuryPhysics->x)).x +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, VenusS.PlanetMass, VenusPhysics->x)).x +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, MoonS.PlanetMass, MoonPhysics->x)).x +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, MarsS.PlanetMass, MarsPhysics->x)).x +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, JupiterS.PlanetMass, JupiterPhysics->x)).x +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, SaturnS.PlanetMass, SaturnPhysics->x)).x +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, UranusS.PlanetMass, UranusPhysics->x)).x +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, NeptuneS.PlanetMass, NeptunePhysics->x)).x;
    f[1] = (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, 1.98855e+30, vec3(0.0))).y+
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, MercuryS.PlanetMass, MercuryPhysics->x)).y +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, VenusS.PlanetMass, VenusPhysics->x)).y +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, MoonS.PlanetMass, MoonPhysics->x)).y +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, MarsS.PlanetMass, MarsPhysics->x)).y +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, JupiterS.PlanetMass, JupiterPhysics->x)).y +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, SaturnS.PlanetMass, SaturnPhysics->x)).y +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, UranusS.PlanetMass, UranusPhysics->x)).y +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, NeptuneS.PlanetMass, NeptunePhysics->x)).y;
    f[2] = (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, 1.98855e+30, vec3(0.0))).z +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, MercuryS.PlanetMass, MercuryPhysics->x)).z +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, VenusS.PlanetMass, VenusPhysics->x)).z +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, MoonS.PlanetMass, MoonPhysics->x)).z +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, MarsS.PlanetMass, MarsPhysics->x)).z +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, JupiterS.PlanetMass, JupiterPhysics->x)).z +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, SaturnS.PlanetMass, SaturnPhysics->x)).z +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, UranusS.PlanetMass, UranusPhysics->x)).z +
        (ForceCalc(EarthS.PlanetMass, EarthPhysics->x, NeptuneS.PlanetMass, NeptunePhysics->x)).z;
    return f;
};
```

Τα παραπάνω υλοποιούνται για όλους τους πλανήτες του ηλιακού συστήματος.

Αφού εκτελεστεί το παραπάνω κομμάτι κώδικα για κάθε πλανήτη του ηλιακού συστήματος γίνεται ανανέωση της θέσης των πλανητών μέσω της RK4 όπως φαίνεται παρακάτω :

```
MarsPhysics->update(time, (86250.0)*deltaTime*scalingFactor);
EarthPhysics->update(time, (86250.0)*deltaTime*scalingFactor);
JupiterPhysics->update(time, (86250.0)*deltaTime*scalingFactor);
VenusPhysics->update(time, (86250.0)*deltaTime*scalingFactor);
MercuryPhysics->update(time, (86250.0)*deltaTime*scalingFactor);
SaturnPhysics->update(time, (86250.0)*deltaTime*scalingFactor);
UranusPhysics->update(time, (86250.0)*deltaTime*scalingFactor);
NeptunePhysics->update(time, (86250.0)*deltaTime*scalingFactor);
MoonPhysics->update(time, (86250.0)*deltaTime*scalingFactor);
```

Η μεταβλητή scalingFactor είναι για έλεγχο της ταχύτητας προσομοίωσης μέσω του πληκτρολογίου, deltaTime είναι ο πραγματικός χρόνος που πέρασε για ένα iteration της προσομοίωσης ενώ ο συντελεστής 86250 χρησιμοποιείται για να υπάρχει αντιστοιχία 1 real-time second = 1 simulation day * scalingFactor.

3.4) Αναπαράσταση των αποτελεσμάτων - Συμπεράσματα)

Για την αναπαράσταση των αποτελεσμάτων χρησιμοποιείται η συνάρτηση DrawingPlanets του μέρους α’.

Για την πραγματική αναπαράσταση των αποτελεσμάτων αντιμετωπίζουμε τα εξής προβλήματα. Το μέγεθος των πλανητών είναι πολύ μικρό συγκριτικά με το μέγεθος του ηλιακού συστήματος και η περίοδος πλήρους περιστροφής κάποιων πλανητών γύρω από τον ήλιο είναι 2 τάξεις μεγέθους μικρότερη από κάποιους άλλους.

Το δεύτερο πρόβλημα είναι ‘άλυτο’ αν θέλουμε να έχουμε ρεαλιστική προσομοίωση μιας και κάποιοι πλανήτες φαίνονται ακίνητοι σε σχέση με κάποιους άλλους. Δεν μπορούμε να επιταχύνουμε πολύ την προσομοίωση διότι δημιουργείται αριθμητική αστάθεια (αναλύεται στα συμπεράσματα).

Το πρώτο πρόβλημα λύνεται με ‘μεγέθυνση’ των πλανητών αλλά δημιουργείται ένα νέο πρόβλημα. Στον κώδικα επιλέγεται μεγέθυνση x500 αλλά υπάρχει έλεγχος για το μέγεθος των πλανητών. Το πρόβλημα που δημιουργείται είναι ότι η τροχιά της σελήνης έχει μέση ακτίνα μικρότερη από την ακτίνα της γης επί 500. Το πρόβλημα λύνεται αν βαθμώσουμε την απόσταση γης - σελήνης. Στον κώδικα επιλέχθηκε 0.1 a.u για βάθμωση όπως φαίνεται παρακάτω :

```
moonP += vec3(0.1) * normalize(moonP - earthP);
```

Οι θέσεις των πλανητών έχουν υπολογιστεί σε μέτρα. Για να υπολογίσουμε την θέση που θα μετατρέψουμε τα μέτρα σε (a.u.).

Για την θέση της γης και της σελήνης απλά ξαναγράφονται οι μεταβλητές earthP, moonP από το μέρος α’ που χρησιμοποιούνται ήδη.

Αν επιλεχθεί να τυπωθούν και οι υπόλοιποι πλανήτες του ηλιακού συστήματος τότε καλείται η DrawingPlanets όπως φαίνεται στην επόμενη σελίδα :

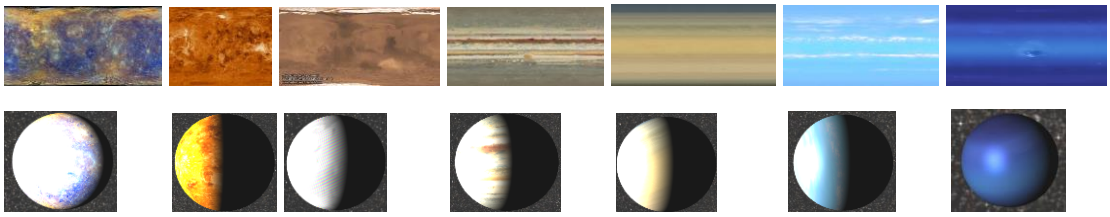
```
// Mercury
DrawingPlanets(projectionMatrix, viewMatrix, (MercuryPhysics->x)/ (vec3(149597871000.0)), time*3.14/58.0, radians(2.04f),
    RadiiFactor*MercuryS.PlanetRad, lightPos, planet, diffuseTextureMercury, specularTextureMercury);
// Venus
DrawingPlanets(projectionMatrix, viewMatrix, (VenusPhysics->x) / (vec3(149597871000.0)), time*3.14/243.0, radians(2.64f),
    RadiiFactor*VenusS.PlanetRad, lightPos, planet, diffuseTextureVenus, specularTextureVenus);
// Mars
DrawingPlanets(projectionMatrix, viewMatrix, (MarsPhysics->x) / (vec3(149597871000.0)), time*3.14/1.025, radians(25.19f),
    RadiiFactor*MarsS.PlanetRad, lightPos, planet, diffuseTextureMars, specularTextureMars);
// Jupiter
DrawingPlanets(projectionMatrix, viewMatrix, (JupiterPhysics->x) / (vec3(149597871000.0)), time*3.14*24.0/9.925, radians(3.13f),
    RadiiFactor*JupiterS.PlanetRad, lightPos, planet, diffuseTextureJupiter, specularTextureJupiter);
// Saturn
DrawingPlanets(projectionMatrix, viewMatrix, (SaturnPhysics->x) / (vec3(149597871000.0)), time*24.0/10.55, radians(26.73f),
    RadiiFactor*SaturnS.PlanetRad, lightPos, planet, diffuseTextureSaturn, specularTextureSaturn);
// Uranus
DrawingPlanets(projectionMatrix, viewMatrix, (UranusPhysics->x) / (vec3(149597871000.0)), -time*3.14/ 0.718, radians(97.77f),
    RadiiFactor*UranusS.PlanetRad, lightPos, planet, diffuseTextureUranus, specularTextureUranus);
// Neptune
DrawingPlanets(projectionMatrix, viewMatrix, (NeptunePhysics->x) / (vec3(149597871000.0)), time*3.14/ 0.6713, radians(28.32f),
    RadiiFactor*NeptuneS.PlanetRad, lightPos, planet, diffuseTextureNeptune, specularTextureNeptune);
```

Οι τιμές της συνάρτησης που υπολογίζονται δυναμικά καλούνται μέσω στιγμιοτύπου της κλάσης Sphere ενώ οι υπόλοιπες έχουν γραφτεί hard coded σύμφωνα με την βιβλιογραφία.

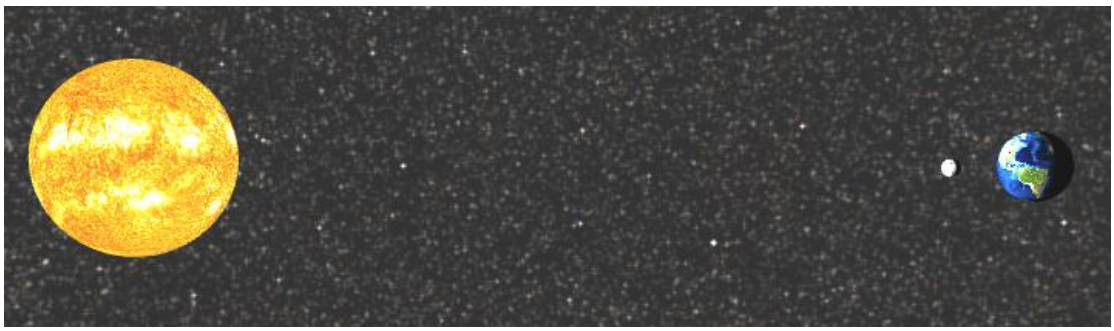
Επίσης, φορτώνονται νέα texture maps στην αρχή του project για τους υπόλοιπους πλανήτες όπως ακριβώς και στο μέρος α’.

Το texture maps και οι πλανήτες φαίνονται παρακάτω :

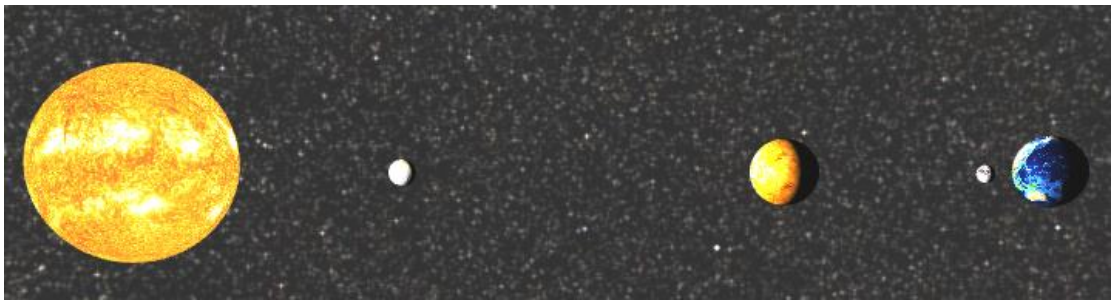
Mercury Venus Mars Jupiter Saturn Uranus Neptune

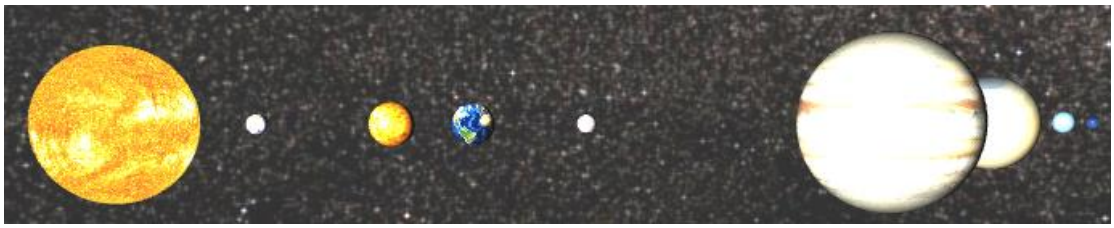


3 - Body Simulation :



All Bodies :





Συμπεράσματα :

Η προσομοίωση του ηλιακού συστήματος οδηγεί σε ένα ευσταθές σύστημα όσο η προσομοίωση γίνεται με μικρό βήμα. Αν βέβαια αυξήσουμε το scalingFactor τότε οι πλανήτες κάνουν μία σπείρα με αυξανόμενη ακτίνα. Αυτό οφείλεται στην πολύ μεγάλη απόσταση που πρέπει να καλυφθεί σε πεπερασμένο αριθμό βημάτων με αριθμητική ολοκλήρωση.

3.5) Σύγκρουση σελήνης - αστεροειδή :

Για την υλοποίηση της σύγκρουσης εκτελούνται τα εξής βήματα :

Αρχικά, δημιουργείται ο αστεροειδής :

Global :

```
GLuint diffuseTextureComet, specularTextureComet;
```

```
Sphere* Comet;
```

createContext() :

```
diffuseTextureComet = loadSOIL("comet.jpg");
specularTextureComet = loadSOIL("comet.jpg");
```

mainLoop() :

```
Comet = new Sphere(startloc * vec3(149597870700.0), vec3(0.0,0.0,1.0)* vec3(20000.0), MoonS.PlanetRad / 2, MoonS.PlanetMass / 2);
```

Τέλος, γίνεται poll για το αν συγκρούστηκε με την σελήνη στην mainLoop(), update η θέση του αστεροειδή και καλείται η DrawingPlanets(. . .) :

```
Comet->forcing = [&](float time, const vector<float>& y)->vector<float> {
    vector<float> f(6, 0.0f);
    f[0] = 0;
    f[1] = 0;
    f[2] = 0;
    return f;
};
Comet->update(time, (86250.0)*deltaTime*scalingFactor);
handleSphereSphereCollision(*MoonPhysics, *Comet);
DrawingPlanets(projectionMatrix, viewMatrix, (Comet->x) / (vec3(149597871000.0))+vec3(0.005,0.0,-0.1), time, radians(50.4f),
    RadiiFactor*(Comet->r), lightPos, planet, diffuseTextureComet, specularTextureComet);
```

Ο κομήτης εκτελεί γραμμική κίνηση και δεν συνυπολογίζουμε τις δυνάμεις των υπόλοιπων πλανητών μιας και θα ήταν υπερβολικά δύσκολο να προβλεφθεί μία τροχιά σύγκρουσης με την σελήνη.

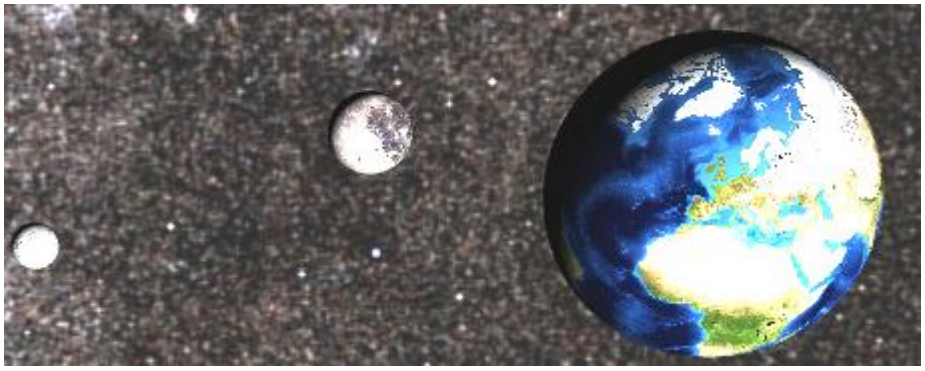
Οι αρχικές τιμές του κομήτη έχουν επιλεχθεί τυχαία με τέτοιο τρόπο ώστε να υλοποιείται σύγκρουση με την σελήνη. Το πρόβλημα είναι ότι η σελήνη έχει ένα offset σε σχέση με την πραγματική τροχιά της ως προς την γη για να μπορούμε να μεγεθύνουμε τους πλανήτες. Αυτό το πρόβλημα λύθηκε έχοντας συνυπολογίσει το offset την στιγμή της σύγκρουσης σε όλη την τροχιά του αστεροειδή.

```
(Comet->x) / (vec3(149597871000.0))+vec3(0.005,0.0,-0.1)
```

Η θέση που καλείται στην DrawingPlanets(…).

Λόγω του offset όμως είναι δυνατόν σε μία σύγκρουση να φαίνεται ότι η σελήνη εκτελεί μία πολύ γρήγορη τοξοειδή κίνηση. Αυτό συμβαίνει διότι η επιτρόχια ταχύτητα που παρατηρείται ως προς την γη είναι v x r. Όμως, το r το αυξάνουμε χάριν καλύτερου οπτικού αποτελέσματος και μπορεί για ‘λίγο’ η κίνηση της σελήνης να φαίνεται ανορθόδοξη.

Γη - σελήνη - αστεροειδής :



Επίσης, επειδή η προσομοίωση είναι εξαρτημένη από το βήμα ολοκλήρωσης της μεθόδου RK4 μπορεί για μεγαλύτερο scalingFactor λόγω μικρών αλλαγών ο αστεροειδής να αστοχήσει.

Οι αρχικές τιμές ανταποκρίνονται για το scalingFactor που είναι αρχικοποιημένο στο project.

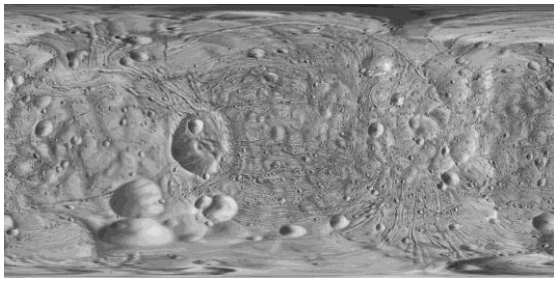
Για την σύγκρουση των σωμάτων χρησιμοποιήθηκε η συνάρτηση

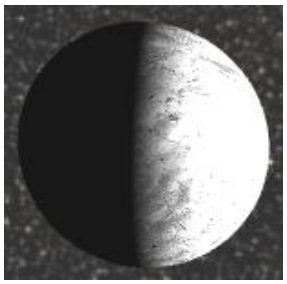
handleSphereSphereCollision(*MoonPhysics, *Comet)

του εργαστηρίου με ένα μικρό correction στην θέση των πλανητών :

```
void handleSphereSphereCollision(Sphere& sphere1, Sphere& sphere2) {
    vec3 n1, n2;
    vec3 v1, v2;
    if (checkForSphereSphereCollision(sphere1.x, sphere1.r, sphere2.x, sphere2.r, n1, n2)) {
        v1 = sphere1.v;
        v2 = sphere2.v;
        sphere1.x += v1*vec3(10.0);
        sphere1.v = v1 - 2 * sphere2.m / (sphere1.m + sphere2.m) * dot(v1-v2,n1) * n1 ;
        sphere1.P = sphere1.m * sphere1.v;
        sphere2.x += v2*vec3(10.0);
        sphere2.v = v2 - 2 * sphere1.m / (sphere1.m + sphere2.m) * dot(v2 - v1, n2) * n2;
        sphere2.P = sphere2.m * sphere2.v;
    }
}
```

Τέλος αυτός είναι χάρτης υφής του αστεροειδή :





3.6) Αποτελέσματα :

Τα αποτελέσματα των προσομοιώσεων είναι αρκετά τυχαία μιας και εξαρτώνται από τις αρχικές τιμές του αστεροειδή, την γωνία σύγκρουσης αλλά και την ταχύτητα της προσομοίωσης.

Γενική αρχή είναι πως αν η σελήνη δεν ξεπεράσει μία κρίσιμη γεωκεντρική ταχύτητα (2.38 km/s) τότε θα παραμείνει δορυφόρος της γης. Αν αυτή η ταχύτητα ξεπεραστεί θα 'ξεφύγει' από την έλξη της γης με πιο συχνό αποτέλεσμα να κάνει μία περίεργη τροχιά γύρω από τον ήλιο.

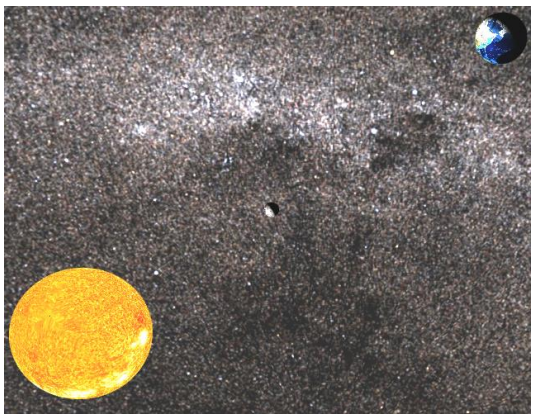
Γενικά δεν έχει γίνει έλεγχος σύγκρουσης με άλλα σώματα γιατί απαιτούνται $O(n^2)$ υπολογισμοί, όπου n ο αριθμός των ηλιακών σωμάτων.

Τέλος, δεν παρατηρείται καμία διαφορά στην κίνηση της γης. Αρχικά, το rotation της γης γύρω από τον εαυτό της είναι αδύνατον να αλλάξει. Όσον αφορά την γραμμική θέση - ταχύτητα (κίνηση) της γης δεν παρατηρείται καμία αλλαγή για τρεις λόγους :

1. Η δύναμη που ασκεί η σελήνη στην γη είναι πολλές τάξεις μεγέθους μικρότερη από την δύναμη που ασκεί ο ήλιος στην γη.

2. Η αριθμητική ακρίβεια σε τόσο μεγάλο βήμα ολοκλήρωσης (ανάλογο της μιας μέρας) είναι συγκρίσιμη με την ελκτική δύναμη σελήνης - γης.

3. Η μεγέθυνση που έχει χρησιμοποιηθεί για να φαίνονται καθαρότερα οι πλανήτες κάνουν πραγματικά αμελητέες όποιες μικρές μεταβολές υπάρχουν στην κίνηση της γης.



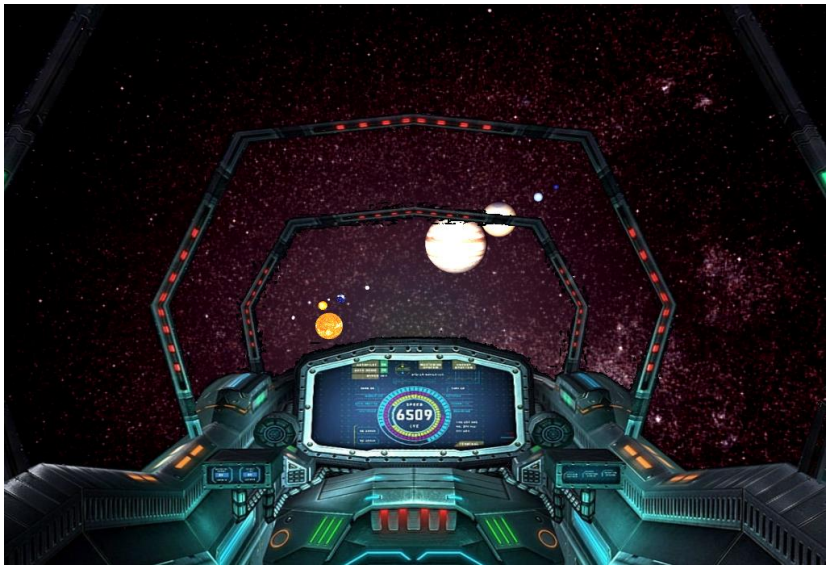
Αστάθεια της κίνησης της σελήνης ως προς την γη και ελκτική τροχιά προς τον ήλιο μετά την σύγκρουση για τις αρχικές τιμές των screenshots.

Τέλος Μέρους Β'

4. Διαστημόπλοιο

Στα πλαίσια του project υλοποιήθηκε ο αλγόριθμος chroma key (ή CSO) και τοποθετήθηκε ένα διαστημόπλοιο μέσω του οποίου μπορούν να προβληθούν όλα τα ερωτήματα της εργασίας.

Αποτέλεσμα (Προσομοίωση όλου του ηλιακού συστήματος) :



α. Περιγραφή του αλγορίθμου :

Ο αλγόριθμος chroma key λειτουργεί ως εξής. Επιλέγεται ένα χρώμα αναφοράς σε μία εικόνα. Αυτό το χρώμα χρησιμοποιείται για όλο το κομμάτι της εικόνας που θέλουμε να τυπώσουμε μια άλλη εικόνα στην θέση της. Γίνεται έλεγχος των δύο εικόνων και όπου υπάρχει το επιλεγμένο χρώμα στην πρώτη εικόνα γίνεται αντικατάσταση με το αντίστοιχο κομμάτι της άλλης εικόνας. Τα παραπάνω υλοποιούνται για κάθε frame.

β. Υλοποίηση του αλγορίθμου στο project :

Αρχικά, επιλέχθηκε ως χρώμα αναφοράς το μαύρο. Ύστερα, έγινε propagate μέσω uniform μεταβλητής το διαστημόπλοιο στον fragment shader.

Διαστημόπλοιο :

Κώδικας :

```
uniform sampler2D diffuseSpaceship;
uniform sampler2D specularSpaceship;
```

Ο έλεγχος για το background υλοποιήθηκε ‘ανάποδα’ από την περιγραφή επειδή υπάρχει ήδη το frame του σύμπαντος στο frame. Ο έλεγχος λοιπόν έγινε για το αν η θέση κάθε fragment της εικόνας του ηλιακού συστήματος ‘πέφτει’ πάνω σε μαύρο κομμάτι του διαστημοπλοίου (threshold = vec3(0.02) για rgb χρώματα). Αν η συνθήκη **δεν** ισχύει τότε αυτό το fragment αντικαθίσταται από το αντίστοιχο της εικόνας του διαστημοπλοίου :

```
fragment_color = vec4(texture(specularSpaceship, vec2(gl_FragCoord.x / 1024.0,1.0- gl_FragCoord.y/768.0)).rgb,1.0);
```

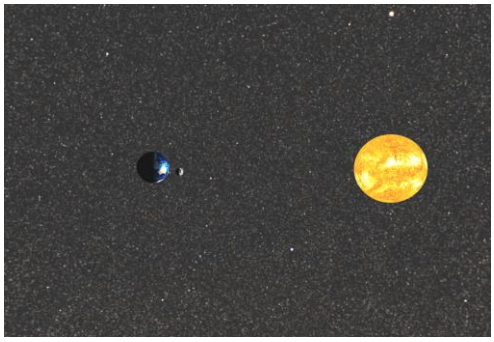
Η gl_FragCoord επιστρέφει την τιμή του fragment που γίνεται επεξεργασία εκείνη την στιγμή στο pixel space.

Ο έλεγχος είναι μεγάλη έκφραση ενώ είναι απλή η λογική του και βρίσκεται στον fragmentshader γραμμή 113.

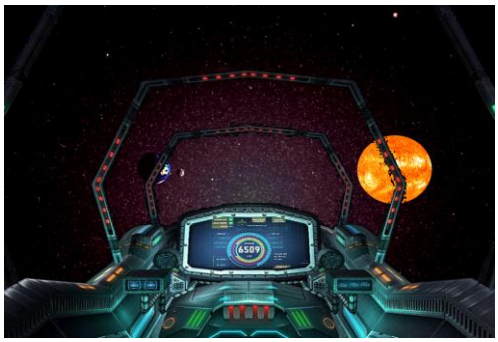
Επίσης, για την πιο ρεαλιστική απεικόνιση προστέθηκε ένα effect ‘φιμέ’ τζαμιού στην εικόνα που δεν αντικαταστάθηκε από το διαστημόπλοιο. ‘Μωβίζουμε’ την εικόνα με αύξηση του συντελεστή ακτινικά και λόγω των συντελεστών αύξησης στους άξονες x, y 16:9 για μεγαλύτερη ρεαλιστικότητα. Αυτό φαίνεται και δτην πρώτη φωτογραφία αυτής της παραγράφου.

Γραμμές 109-111 στον fragmentshader διότι είναι πολύ μεγάλη έκφραση αφού έχει υλοποιηθεί σε μία γραμμή μαζί με όρια για τις τιμές (clamp(…)).

Σύγκριση ίδιου στιγμιότυπου χωρίς και με εφαρμογή όλων των προηγούμενων :



Κανονικό animation του μέρους α’



Ίδιο στιγμιότυπο με εφαρμογή του αλγόριθμου chroma key και το ‘μώβισμα’ της εικόνας που αντικαθιστά το background.

5. Controls για τις ενέργειες - ερωτήματα στο εικονικό περιβάλλον

Όλα τα ερωτήματα και αλγόριθμοι που υλοποιήθηκαν στην εργασία εκτελούνται χωρίς επανεκκίνηση του κώδικα και αλλαγή τιμών, μέσω πλήκτρων του πληκτρολογίου.

Ερωτήματα απαλλακτικής :

Animation -> Key = 1

Simulation 3 Bodies (γη - σελήνη - ήλιος) -> Key = 2

Simulation Solar System -> Key = 3

Collision -> Key = 4

Controls που επιδρούν σε όλα τα ερωτήματα :

Περιήγηση στο περιβάλλον -> Keys = AWSD + mouse

Spaceship On / Off -> Keys = E / Q

Simulation and Animation Time Faster / Slower (scalingFacor) -> Keys = X / Z

Planets Radius Bigger / Smaller -> Keys = V / C

Star Wars Anthem -> Key = M

6) Παράρτημα

6.1) Αστρονομικά μεγέθη – έννοιες :

Astronomical units : Η Αστρονομική Μονάδα (α.μ.) είναι μονάδα μέτρησης αποστάσεων. Ισούται με 149.597.870,700 μέτρα. Παλαιότερα, η Αστρονομική μονάδα οριζόταν ως η μέση απόσταση της Γης από τον Ήλιο. Χρησιμοποιείται για τη μέτρηση αποστάσεων μέσα στο Ηλιακό σύστημα (π.χ. της απόστασης κάποιου σώματος από τον Ήλιο). Το διεθνές σύμβολό της είναι το AU(από το αγγλικό Astronomical Unit) και στην ελληνική α.μ.

VSOP : The semi-analytic planetary theory **VSOP** (French: *Variations Séculaires des Orbites Planétaires*) is a concept describing long-term changes (secular variation) in the orbits of the planets Mercury to Neptune. If one ignores the gravitational attraction between the planets and only models the attraction between the Sun and the planets, then with some further idealisations, the resulting orbits would be Keplerian ellipses. In this idealised model the shape and orientation of these ellipses would be constant in time. In reality, while the planets are at all times roughly in Keplerian orbits, the shape and orientation of these ellipses does change slowly over time. Over the centuries increasingly complex models have been made of the deviations from simple Keplerian orbits. In addition to the models, efficient and accurate numerical approximation methods have also been developed.

J2000 : Η **εποχή** στην αστρονομία είναι μια στιγμή στο χρόνο που χρησιμοποιείται ως σημείο αναφοράς για κάποια μεταβαλλόμενο χρονικό διάστημα μιας αστρονομικής ποσότητας, όπως οι Ουράνιες συντεταγμένες ή τα ελλειπτικά τροχιακά στοιχεία ενός ουράνιου σώματος, γιατί αυτά υπόκεινται σε διαταραχές και μεταβάλλονται με το χρόνο. Οι χρονικά μεταβαλλόμενες αστρονομικές ποσότητες μπορούν να περιλαμβάνουν, για παράδειγμα, το μέσο μήκος ή τη Μέση ανωμαλία ενός σώματος, τον κόμβο της τροχιάς του σε σχέση με ένα επίπεδο αναφοράς, την κατεύθυνση από το απόγειο και το αφήλιο της τροχιάς του, ή το μέγεθος του κύριου άξονα της τροχιάς του.

Keplerian τροχιά : https://en.wikipedia.org/wiki/Kepler_orbit

Poincare chaos theorem : Ο Poincare απέδειξε ότι το 3 body problem αποτελεί σύστημα εξαιρετικά ευαίσθητο σε διαταραχές και πρακτικά δεν υπάρχει λόγος να αναλύονται οι ακριβείς τροχιές των μεταβλητών του συστήματος αλλά οι τροχιές ποιοτικά.

6.2) Αλγόριθμοι υλοποιημένοι στο εργαστήριο :

Phong : Ο αλγόριθμος αυτός υπολογίζει την επίδραση της πηγής φωτός σε μία επιφάνεια. Στην εργασία έχει τροποποιηθεί ελαφρώς η επίδραση της απόστασης για να φωτίζονται επαρκώς όλοι οι πλανήτες.

Camera : Αυτός ο αλγόριθμος μας επιστρέφει τον projection και view matrix αναλόγως με την θέση που ορίζουμε μέσω του πληκτρολογίου (δεν τροποποιήθηκε).

RigidBody / Runge-Kutta 4 : Αυτή η μέθοδος χρησιμοποιείται για να γίνει αριθμητική ολοκλήρωση στην κατάσταση ενός στερεού σώματος για πεπερασμένο χρονικό διάστημα. Χρησιμοποιήθηκε με quaternions.

6.3) Shaders :

Στην εργασία χρησιμοποιήθηκαν ο vertexshader και ο fragmentshader. Ο vertex shader χρησιμοποιήθηκε μόνο για τον υπολογισμό του πίνακα projection matrix * view matrix * model matrix (MVP) και για να γίνει propagate όλων των μεταβλητών που διοχετεύονται μέσω κάποιου attribute στους shaders, στον fragmentshader.

Στον fragmentshader υλοποιούνται οι αλγόριθμοι φωτισμού (Phong), σκίασης, chroma key και το μώβισμα της εικόνας. Όλα αυτά ελέγχονται μέσω uniform μεταβλητών οι οποίες λαμβάνουν τιμή στην mainLoop() αναλόγως με το ερώτημα ή την ενέργεια.

7) Βιβλιογραφία - Πηγές

http://www.stargazing.net/kepler/ellipse.html

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

http://ceadserv1.nku.edu/longa//classes/calculus_resources/labs/lab27/lab27.pdf

https://en.wikipedia.org/wiki/Star_system

https://link.springer.com/chapter/10.1007/978-1-4684-5997-5_2

https://en.wikipedia.org/wiki/VSOP_(planets)

http://neoprogrammics.com/vsop87/

http://w.astro.berkeley.edu/~dperley/programs/ssms.html

http://www.alpieprealpi.it/valutami.com/documents/simulation-solar-system.pdf

http://neoprogrammics.com/vsop87/vsop87_theory_paper/

https://en.wikipedia.org/wiki/Geocentric_orbit

https://en.wikipedia.org/wiki/VSOP_(planets)#VSOP82

http://www.stjarnhimlen.se/comp/ppcomp.html

https://en.wikipedia.org/wiki/Chroma_key

https://en.wikipedia.org/wiki/Kepler_orbit