ECE433: ΓΡΑΦΙΚΑ ΗΥ

Χειμερινό Εξάμηνο 2021-2022

Εργασία 2

Ομάδα φοιτητών #4

Χωροπανίτης Πασχάλης – 2453

# Γενικές πληροφορίες για την εργασία.

Συνολικός χρόνος ολοκλήρωσης της εργασίας: 6 ημέρες (3-4 ημέρες για τις θεωρητικές και 2 ημέρες για την προγραμματιστική άσκηση 6 ).

Στο παραδοτέο περιέχονται τα εξής:

* clipping\_polygons.c άσκηση 6.
* Makefile
* template\_gr.docx
* \*freeglut φάκελος για την μεταγλώττιση στα Windows
* \*freeglut.dll .dll αρχείο για την εκτέλεση στα Windows

\*(Προαιρετικά) Απαραίτητα μόνο σε περιβάλλον Windows.

Γενικές πληροφορίες για τις προγραμματιστικές ασκήσεις:

* Οι ασκήσεις είναι προγραμματισμένες σε C, σε περιβάλλον Windows Subsystem for Linux (WSL 2) με εγκατεστημένο OS το Ubuntu 20.04 (LTS).
* Οι ασκήσεις έχουν δοκιμαστεί και σε native linux install (Ubuntu 20.04 (LTS)), καθώς και στα Windows 10 Home, και δουλεύουν χωρίς κανένα πρόβλημα.

Χρήσιμα χαρακτηριστικά του μηχανήματος στο οποίο δούλεψα:

* Laptop Lenovo Ideapad Gaming 3
* Renderer: NVIDIA GeForce GTX 1650 Ti (4096 MB vram)
* OpenGL Version: 4.6 (4.6.0 NVIDIA 497.29)

Οδηγίες για compile/link/execution των ασκήσεων σε περιβάλλον Linux:

* Στην παρακάτω εντολή βρίσκονται όλα τα πακέτα που χρειάστηκα για την ανάπτυξη των ασκήσεων:
  + sudo apt-get install mesa-utils freeglut3 freeglut3-dev
* Στο .zip αρχείο παράδοσης περιέχεται και ένα Makefile για την μεταγλώττιση των αρχείων μας.
* Για την μεταγλώττιση και των ασκήσεων εκτελέστε την παρακάτω εντολή:
  + make ή make all
* Για την εκτέλεση των ασκήσεων, εκτελέστε την παρακάτω εντολή:
  + ./clipping\_polygons για την άσκηση 6.
* Για την εκκαθάριση του φακέλου από τα εκτελέσιμα και τα object αρχεία της μεταγλώττισης, εκτελέστε την παρακάτω εντολή:
  + make clean

(Προαιρετικό) Οδηγίες για compile/link/execution των ασκήσεων σε περιβάλλον Windows:

* Η μεταγλώττιση των ασκήσεων στα Windows γίνεται πάλι με το ίδιο Makefile, αλλά προϋποθέτει την εγκατάσταση κάποιου third-party μεταγλωττιστή (gcc). Στην περίπτωσή μου χρησιμοποιώ το MinGW.
* Στο .zip αρχείο παράδοσης περιέχεται επίσης ένας φάκελος με όνομα freeglut και ένα αρχείο freeglut.dll, τα οποία είναι απαραίτητα για την μεταγλώττιση και εκτέλεση των ασκήσεων στα Windows.
* Για την μεταγλώττιση και των ασκήσεων εκτελέστε την παρακάτω εντολή:
  + make ή make all
* Για την εκτέλεση των ασκήσεων, απλά τρέξτε τα εκτελέσιμα .exe αρχεία.
* Για την εκκαθάριση του φακέλου από τα εκτελέσιμα και τα object αρχεία της μεταγλώττισης, εκτελέστε την παρακάτω εντολή:
  + make clean

# ΑΣΚΗΣΕΙΣ

## Άσκηση 1

Δισδιάστατος αλγόριθμος αποκοπής ευθυγράμμων τμημάτων Liang – Barsky, ελαφρώς τροποποιημένος από τις διαλέξεις.

Η παραμετρική εξίσωση μιας γραμμής δίνετε από:

* X = x1 + t(x2-x1)
* Y = y1 + t(y2-y1)  
  , όπου 0 <= t <= 1.

Οπότε έχουμε:

* xwmin <= x1 + t(x2-x1) <= xwmax
* ywmin <= y1 + t(y2-y1) <= ywmax

Οι παραπάνω ανισότητες, μπορούνε να εκφραστούν ως εξής:

* tpk <= qk   
  , όπου k = 1, 2, 3, 4 που αντιστοιχούν στο αριστερό, δεξί, κάτω και πάνω όριο.

Το p και q, μπορούν να οριστούν ως εξής:

* p1 = -(x2-x1), q1 = x1 - xwmin (Left Boundary)
* p2 = (x2-x1), q2 = xwmax - x1 (Right Boundary)
* p3 = -(y2-y1), q3 = y1 - ywmin (Bottom Boundary)
* p4 = (y2-y1), q4 = ywmax - y1 (Top Boundary)

Επίσης, ισχύουν οι παρακάτω κανόνες:

* Όταν η γραμμή είναι παράλληλη με ένα από τα όρια του παραθύρου, τότε pk = 0.
* Όταν pk < 0, η γραμμή πηγαίνει από έξω προς τα μέσα (μπαίνει).
* Όταν pk > 0, η γραμμή πηγαίνει από μέσα προς τα έξω (βγαίνει).
* Όταν pk = 0 και qk < 0, τότε η γραμμή βρίσκεται έξω από τα όρια του παραθύρου.
* Όταν pk = 0 και qk > 0, τότε η γραμμή βρίσκεται μέσα στα όρια του παραθύρου.

Οπότε ο αλγόριθμος έχει ως εξής:

for each line segment to be clipped  
 pre-calculate pk and qk;  
 if p1 = p0 then line is degenerate so clip as a point;  
 else  
 begin  
 tmin = 0, tmax = 1;  
 for each candidate intersection with a clip edge  
 if pk = 0 then {Ignore edges parallel to line}  
 begin  
 calculate t; {of line and clip edge intersection}  
 use sign of pk to categorize as pstart or pend;  
 if pstart then t1 = max(0, t1, t);  
 if pend then t2 = min(1, t2, t);  
 end  
 if t1 > t2 then return nil  
 else return p(t1) and p(t2) as true clip intersections  
 end

## Άσκηση 2

Δισδιάστατος αλγόριθμος αποκοπής ευθυγράμμων τμημάτων Cohen - Sutherland, ελαφρώς τροποποιημένος από τις διαλέξεις.

Ο Cohen – Sutherland αλγόριθμος χωρίζει ένα δισδιάστατο χώρο σε 9 περιοχές και καθορίζει τις γραμμές και τα κομμάτια των γραμμών που βρίσκονται μέσα στα όρια του παραθύρου.

Δημιουργούνται λοιπόν 9 περιοχές, 8 εξωτερικές περιοχές και 1 εσωτερική.  
Κάθε περιοχή έχει ένα 4-bit αριθμό.  
Για ένα σημείο (x, y) μιας γραμμής, ο 4-bit αριθμός της περιοχής που ανήκει, μπορεί να υπολογιστεί συγκρίνοντας το x, y με το xmin, xmax, ymin, ymax.

* Αν το x είναι μικρότερο από το xmin, τότε ο bit αριθμός είναι το 1.
* Αν το x είναι μεγαλύτερο από το xmax, τότε ο bit αριθμός είναι το 2.
* Αν το y είναι μικρότερο από το ymin, τότε ο bit αριθμός είναι το 3.
* Αν το y είναι μεγαλύτερο από το ymax, τότε ο bit αριθμός είναι το 4.

Άρα για κάθε γραμμή, υπάρχουν 3 πιθανές περιπτώσεις:

1. Η γραμμή βρίσκεται ολόκληρη μέσα στα όρια του παραθύρου: Και τα δυο σημεία της ευθείας βρίσκονται μέσα στα όρια του παραθύρου (Bitwise OR = 0).
2. Η γραμμή βρίσκεται ολόκληρη έξω από τα όρια του παραθύρου: Και τα δυο σημεία της ευθείας βρίσκονται έξω από τα όρια του παραθύρου (Bitwise AND != 0).
3. Ένα μέρος της γραμμής βρίσκεται μέσα και ένα μέρος της γραμμής βρίσκεται έξω από τα όρια του παραθύρου:  
   Σε αυτή την περίπτωση ο αλγόριθμος βρίσκει το σημείο που είναι έξω από το παράθυρο. Το σημείο τομής της γραμμής από το εξωτερικό σημείο με το παράθυρο, γίνεται το νέο γωνιακό σημείο της ευθείας και ο αλγόριθμος επαναλαμβάνεται.

Οπότε ο αλγόριθμος έχει ως εξής:

ComputeOutCode(x0, y0, outcode0)

ComputeOutCode(x1, y1, outcode1)

repeat

check for trivial reject or trivial accept

pick the point that is outside the clip rectangle

if TOP then

x = x0 + (x1 – x0) \* (ymax – y0)/(y1 – y0); y = ymax;

else if BOTTOM then

x = x0 + (x1 – x0) \* (ymin – y0)/(y1 – y0); y = ymin;

else if RIGHT then

y = y0 + (y1 – y0) \* (xmax – x0)/(x1 – x0); x = xmax;

else if LEFT then

y = y0 + (y1 – y0) \* (xmin – x0)/(x1 – x0); x = xmin;

end {calculate the line segment}

if (x0, y0 is the outer point) then

x0 = x; y0 = y; ComputeOutCode(x0, y0, outcode0)

else

x1 = x; y1 = y; ComputeOutCode(x1, y1, outcode1)

end {Subdivide}

until done

## Άσκηση 3

Εφαρμογή του αλγόριθμου Cohen – Sutherland για την αποκοπή του παρακάτω ευθύγραμμου τμήματος P0 P1 ως προς το (μπλε) παράθυρο αποκοπής.

Βήμα 1:

|  |  |
| --- | --- |
|  | P0(0.5, 0) P1(5.5, 5) slope, m = (y1 – y0) / (x1 – x0) = 1  P0 🡪 0101 P1 🡪 1010 P0 AND P1 = 0000 🡪 line is partially visible, so needs clipping. |

Βήμα 2:

|  |  |
| --- | --- |
|  | P0(0.5, 0) P1(5.5, 5)  Yi = y0 + m(xleft – x0) = 0 + 1(1 – 0.5) = 0.5 Xi = xleft = 1 P0’(1, 0.5)  Yi = y1 + m(xright – x1) = 5 + 1(5 – 5.5) = 4.5 Xi = xright = 5 P1’(5, 4.5)  P0’ 🡪 0100 P1’ 🡪 1000 P0’ AND P1’ = 0000 🡪 line is partially visible, so it needs clipping again. |

Βήμα 3:

|  |  |
| --- | --- |
|  | P0’(1, 0.5) P1’(5, 4.5)  Xi = x0’ + (ybottom – y0’)/m = 1 + (1 – 0.5)/1 = 1.5 Yi = ybottom = 1 P0’’(1.5, 1)  Xi = x1’ + (ytop – y1’)/m = 5 + (4 – 4.5)/1 = 4.5 Yi = ytop = 4 P1’’(4.5, 4)  P0’’ 🡪 0000 P1’’ 🡪 0000 Line is completely visible. End of Algorithm. |

## Άσκηση 4

Εφαρμογή του αλγόριθμου Liang – Barsky για την αποκοπή του παρακάτω ευθύγραμμου τμήματος P0 P1 ως προς το (μπλε) παράθυρο αποκοπής.

Βήμα 1:

|  |  |
| --- | --- |
|  | P0(0.5, 0) P1(5.5, 5)  Xwmin = 1, Ywmin = 1, Xwmax = 5, Ywmax = 4  Calculate pk and qk for k=1, 2, 3, 4 as p1 = -(x1–x0) = -5, q1 = x0 – Xwmin = -0.5 (left) p2 = (x1–x0) = 5, q2 = Xwmax – x0 = 4.5 (right) p3 = -(y1–y0) = -5, q3 = y0 – Ywmin = -1 (bottom) p4 = (y1–y0) = 5, q4 = Ywmax – y0 = 4 (top)  pk != 0 so calculate rk = qk/pk for k=1, 2, 3, 4 r1 = -0.5/-5 = 0.1 r2 = 4.5/5 = 0.9 r3 = -1/-5 = 0.2 r4 = 4/5 = 0.8  Determine u1 for pk<0 and u2 for pk > 0: u1 = {r1, r3, 0}max = 0.2 u2 = {r2, r4, 1}min = 0.8  u1 < u2 so calculate end points of clipped line  Xi = x0 + u1Δx Yi = y0 + u1Δy P0’(Xi, Yi) Xi = 0.5 + 0.2\*5 = 1.5 Yi = 0 + 0.2\*5 = 1 P0’(1.5, 1)  Xi = x0 + u2Δx Yi = y0 + u2Δy P1’(Xi, Yi) Xi = 0.5 + 0.8\*5 = 4.5 Yi = 0 + 0.8\*5 = 4 P1’(4.5, 4) |

Βήμα 2:

|  |  |
| --- | --- |
|  | P0’(1.5, 1) P1’(4.5, 4)  Xwmin = 1, Ywmin = 1, Xwmax = 5, Ywmax = 4  Calculate pk and qk for k=1, 2, 3, 4 as p1 = -(x1–x0) = -3, q1 = x0 – Xwmin = 0.5 (left) p2 = (x1–x0) = 3, q2 = Xwmax – x0 = 3.5 (right) p3 = -(y1–y0) = -3, q3 = y0 – Ywmin = 0 (bottom) p4 = (y1–y0) = 3, q4 = Ywmax – y0 = 3 (top)  pk != 0 so calculate rk = qk/pk for k=1, 2, 3, 4 r1 = 0.5/-3 = -0.16 r2 = 3.5/3 = 1.16 r3 = 0/-3 = 0 r4 = 3/3 = 1  Determine u1 for pk<0 and u2 for pk > 0: u1 = {r1, r3, 0}max = 0 u2 = {r2, r4, 1}min = 1  The line is entirely inside the window. Algorithm ends. |

## Άσκηση 5

Τροποποίηση του αλγόριθμου αποκοπής ευθυγράμμων τμημάτων Cohen – Sutherland από 2Δ σε 3Δ χώρο.

Όπως και με τις 2 διαστάσεις, χωρίζουμε τον χώρο πάλι σε περιοχές. Αυτή τη φορά χρησιμοποιούμε 6-bit κωδικούς περιοχής.

Αντιστοιχούμε του κωδικούς περιοχής στα endpoints για Canonical Parallel View Volume ορισμένη από:  
x = xmin, x = max; y = ymin, y = ymax; z = zmin, z = zmax;

Οι bit κωδικοί είναι αντίστοιχα false(0) ή true(1), ανάλογα με τις παρακάτω εξισώσεις:

* Bit 1 = endpoint is Left view volume = sign (-x)
* Bit 2 = endpoint is Right view volume = sign (x-1)
* Bit 3 = endpoint is Bottom view volume = sign (-y)
* Bit 4 = endpoint is Top view volume = sign (y-1)
* Bit 5 = endpoint is Behind view volume = sign (z-1)
* Bit 6 = endpoint is Front view volume = sign (-z)

Οπότε ο αλγόριθμος έχει ως εξής:

ComputeOutCode(x0, y0, z0 outcode0)

ComputeOutCode(x1, y1, z1 outcode1)

// Calculate a and b

a = (x1 – x0)/(y1 – y0);

b = (y1 – y0)/(z1 – z0);

repeat

check for trivial reject or trivial accept

pick the point that is outside the clip rectangle

if TOP then  
 x = x0 + (ymax – y0)\*a;  
 z = z0 + (ymax – y0)/b;  
 y = ymax;

else if BOTTOM then

x = x0 + (ymin – y0)\*a;  
 z = z0 + (ymin – y0)/b;  
 y = ymin;

else if RIGHT then

y = y0 + (xmax – x0)/a;  
 z = z0 + (xmax – x0)/(a\*b);

x = xmax;

else if LEFT then  
 y = y0 + (xmin – x0)/a;  
 z = z0 + (xmin – x0)/(a\*b);

x = xmin;  
 else if BEHIND then

x = x0 + (zmax - z0)\*a\*b;

y = y0 + (zmax – z0)\*b;

z = zmax;  
 else if FRONT then  
 x = x0 + (zmin - z0)\*a\*b;

y = y0 + (zmin – z0)\*b;

z = zmin;

end {calculate the line segment}

if (x0, y0, z0 is the outer point) then

x0 = x; y0 = y; z0 = z ComputeOutCode(x0, y0, z0 outcode0)

else

x1 = x; y1 = y; z1 = z ComputeOutCode(x1, y1, z1 outcode1)

end {Subdivide}

until done

## Άσκηση 6

Πρόγραμμα που υλοποιεί τον αλγόριθμο αποκοπής πολυγώνων Sutherland – Hodgman ως προς ένα ορθογώνιο παράθυρο σε περιβάλλον OpenGL/GLUT.

Ο χρήστης ξεκινάει το πρόγραμμα, επιλέγοντας με το αριστερό κλικ του ποντικιού του τις κορυφές ενός τυχαίου κυρτού πολυγώνου.  
A screenshot of a computer

Description automatically generated with medium confidence

Για την διευκόλυνση του χρήστη, ειδικά στην επιλογή των σημείων για τον σχεδιασμό του πολυγώνου, το πρόγραμμα έχει ως φόντο τους άξονες x, y σε καρτεσιανό σύστημα συντεταγμένων. Επίσης, όταν επιλεχτεί ένα σημείο, εμφανίζονται στην οθόνη οι x και y συντεταγμένες του.

Στη συνέχεια, πατώντας το πλήκτρο F1, ζωγραφίζεται το πολύγωνο με τις κορυφές που έχουμε επιλέξει στο προηγούμενο βήμα και γίνεται η μετάβαση από την κατάσταση εισαγωγής πολυγώνου στην κατάσταση εισαγωγής παραλληλόγραμμου αποκοπής.  
Graphical user interface, chart, surface chart

Description automatically generated

Έπειτα, ο χρήστης μπορεί να ορίσει το ορθογώνιο αποκοπής σέρνοντας το ποντίκι με πατημένο το αριστερό πλήκτρο.  
Shape

Description automatically generated with low confidence

Η αποκοπή του πολυγώνου ενεργοποιείται πατώντας το πλήκτρο c ή C, υλοποιώντας τον αλγόριθμο των Sutherland – Hodgeman.  
A picture containing shape

Description automatically generated

Ο χρήστης μπορεί να αλλάξει το ορθογώνιο αποκοπής όσες φορές επιθυμεί και εφόσον υπάρχει ακόμα το πολύγωνο μπορεί να κάνει και όσες αποκοπές θελήσει.  
A picture containing shape

Description automatically generated  
A picture containing logo

Description automatically generated

Επίσης, ο χρήστης έχει τη δυνατότητα για εμφάνιση – απόκρυψη του ορθογώνιου αποκοπής πατώντας το space bar.  
Shape

Description automatically generated with low confidence

!!Λόγω, της φύσης του αλγορίθμου και τον τρόπο με τον οποίο τον υλοποίησα, το πρόγραμμα δουλεύει σωστά μόνο όταν ο ορισμός του παραθύρου αποκοπής γίνεται από κάτω αριστερά προς τα πάνω δεξιά. Δηλαδή η κατεύθυνση με την οποία σχηματίζεται το ορθογώνιο παραλληλόγραμμο αποκοπής είναι από την κάτω αριστερή γωνία του στην πάνω δεξιά του.!!

Η άσκηση δουλεύει και όταν αλλάζουμε το μέγεθος του παραθύρου (resize, fullscreen). Απλά, κάθε φορά που ο χρήστης αλλάζει το μέγεθος του παραθύρου, γίνεται καθαρισμός της οθόνης για να γίνει συγχρονισμός των καινούργιων συντεταγμένων του παραθύρου.

Ο χρήστης έχει την δυνατότητα καθαρισμού της οθόνης με το δεξί πλήκτρο του ποντικιού και μπορεί να αρχίσει την παραπάνω διαδικασία από την αρχή ορίζοντας νέο πολύγωνο.

Τέλος, το πρόγραμμα τερματίζεται πατώντας το πλήκτρο q ή Q.

## Άσκηση 7

Εφαρμογή του αλγόριθμου Sutherland-Hodgman για την αποκοπή του παρακάτω πολυγώνου ως προς το (μπλε) παράθυρο αποκοπής.

4 Cases of different outputs:

1. Outside 🡪 Inside : P’i, Pj
2. Inside 🡪 Inside : Pj
3. Inside 🡪 Outside : P’i
4. Outside 🡪 Outside : None

Βήμα 1:

|  |  |
| --- | --- |
|  | Left Clipper:  Initial edges order: A B C D E F  AB 🡪 inside to outside 🡪 A’ BC 🡪 outside to inside 🡪 B’, C CD 🡪 inside to inside 🡪 D DE 🡪 inside to inside 🡪 E EF 🡪 inside to inside 🡪 F FA 🡪 inside to inside 🡪 A  So, the new output edges will be: A’ B’ C D E F A |

Βήμα 2:

|  |  |
| --- | --- |
|  | Right Clipper:  Initial edges order: A’ B’ C D E F A  A’B’ 🡪 inside to inside 🡪 B’ B’C 🡪 inside to inside 🡪 C CD 🡪 inside to outside 🡪 C’ DE 🡪 outside to inside 🡪 D’, E EF 🡪 inside to inside 🡪 F FA 🡪 inside to inside 🡪 A AA’ 🡪 inside to inside 🡪 A’  So, the new output edges will be: B’ C C’ D’ E F A A’ |

Βήμα 3:

|  |  |
| --- | --- |
|  | Bottom Clipper:  Initial edges order: B’ C C’ D’ E F A A’  B’C 🡪 inside to inside 🡪 C CC’ 🡪 inside to inside 🡪 C’ C’D’ 🡪 inside to inside 🡪 D’ D’E 🡪 inside to outside 🡪 D’’ EF 🡪 outside to inside 🡪 E’, F FA 🡪 inside to outside 🡪 F’ AA’ 🡪 outside to inside 🡪 A’’, A’ A'B’ 🡪 inside to inside 🡪 B’  So, the new output edges will be: C C’ D’ D’’ E’ F F’ A’’ A’ B’ |

Βήμα 4:

|  |  |
| --- | --- |
|  | Top Clipper:  Initial edges order: C C’ D’ D’’ E’ F F’ A’’ A’ B’  CC’ 🡪 outside to inside 🡪 C’’, C’ C’D’ 🡪 inside to inside 🡪 D’ D’D’’ 🡪 inside to inside 🡪 D’’ D’’E’ 🡪 inside to inside 🡪 E’ E’F 🡪 inside to inside 🡪 F FF’ 🡪 inside to inside 🡪 F’ F’A’’ 🡪 inside to inside 🡪 A’’ A'’A’ 🡪 inside to inside 🡪 A’ A'B’ 🡪 inside to inside 🡪 B’ B’C 🡪 inside to outside 🡪 B’’  So, the new output edges will be: C’’ C’ D’ D’’ E’ F F’ A’’ A’ B’ B’’ |

Βήμα 5:

|  |  |
| --- | --- |
|  | So, clipped polygon edges are: C’’ C’ D’ D’’ E’ F F’ A’’ A’ B’ B’’  End of algorithm. |

## Άσκηση 8

## Άσκηση 9

Εξισώσεις για την μετατροπή παραμέτρων RGB σε τιμές HSV και αντίστροφα.

1. Από RGB σε HSV έχουμε:  
   Διαιρούμε τις R, G, B τιμές με 255 για να αλλάξουμε το εύρος τιμών από 0..255 σε 0..1.  
   R’ = R / 255  
   G’ = G / 255  
   B’ = B / 255  
   Cmax = max(R’, G’, B’)  
   Cmin = min(R’, G’, B’)  
   Δ = Cmax – Cmin  
     
   Υπολογισμός απόχρωσης (Hue):  
   Text

   Description automatically generated  
   Υπολογισμός κορεσμού (Saturation):  
   Text, letter

   Description automatically generated  
   Υπολογισμός τελικής τιμής:  
   V = Cmax
2. Από HSV σε RGB έχουμε:  
   Όταν 0 <= H <= 360, 0 <= S <= 1 και 0 <= V <= 1:  
   C = V × S  
   X = C × (1 - |(H/60°) mod 2 – 1|)  
   m = V – C  
   Text, letter

   Description automatically generated  
   (R, G, B) = ((R’ + m) × 255, (G’ + m) × 255, (B’ + m) × 255)

# ΒONUS PROJECT

Στο παραδοτέο επίσης περιέχεται ένας φάκελος με το όνομα bonus\_project που έχει τα εξής:

* views.cpp
* visiting3Dhouse.cpp
* Makefile
* textures/ φάκελος που περιέχει όλα τα textures που χρειάζονται.

Οι οδηγίες μεταγλώττισης είναι ίδιες με παραπάνω.

Στο views.cpp υλοποιούνται όλες οι προβολές που ζητήθηκαν από την εκφώνηση.

Ο χρήστης με το που τρέξει το πρόγραμμα, πρέπει να πατήσει δεξί κλικ για να επιλέξει από το μενού ποια προβολή θέλει να εμφανιστεί.

Στο visiting3Dhouse.cpp υλοποιείται η κίνηση της κάμερας και ο φωτισμός της σκηνής.

Ο χρήστης με το που τρέξει το πρόγραμμα μπορεί να περιηγηθεί στο χώρο χρησιμοποιώντας τα εξής κουμπιά:

* w για μπροστά
* s για πίσω
* a για αριστερά
* d για δεξιά
* q για κάτω
* e για πάνω

Επίσης, ο χρήστης έχει την δυνατότητα να ελέγχει και που κοιτάει η κάμερα με τα παρακάτω κουμπιά:

* right arrow 🡪 η κάμερα στρίβει δεξιά
* left arrow 🡪 η κάμερα στρίβει αριστερά
* up arrow 🡪 η κάμερα στρίβει προς τα πάνω
* down arrow 🡪 η κάμερα στρίβει προς τα κάτω

Το μόνο πρόβλημα που έχει το project είναι όταν ο χρήστης κουνάει την κάμερα για κάποιο λόγο κάποια textures τρεμοπαίζουν (texture flickering).

Επίσης, για να χρησιμοποιήσω textures μαζί με το glutSolidCube έπρεπε να υλοποιήσω ξανά την συνάρτηση στον κώδικα μου γιατί για κάποιο λόγο η glutSolidCube δεν κάνει generate texture coordinates, οπότε έπρεπε να τα περάσω manually μέσα στην συνάρτηση που την ονόμασα myGlutSolidCube. Εδώ φαντάζομαι πως δημιουργήθηκε και το πρόβλημα που ανέφερα παραπάνω.

# ΒΙΒΛΙΟΓΡΑΦΙΑ

1. «https://www.geeksforgeeks.org/polygon-clipping-sutherland-hodgman-algorithm-please-change-bmp-images-jpeg-png/»
2. «https://www.rapidtables.com/convert/color/index.html»
3. R. Kodituwakku, K.R. Wijeweera, M.A.P. Chamikara, «An Efficient Line Clipping Algorithm for 3D Space».

# ΠΑΡΑΡΤΗΜΑ

## Κώδικας Ερώτησης 6

Ακολουθεί ο κώδικας για το ερώτημα 6. Όνομα Αρχείου «clipping\_polygons.c»

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* Ergasia 2 – Askhsh 6 – 23.02.2022

 \* Choropanitis Paschalis - 2453

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <GL/glut.h>

#include <stdio.h>

#include <string.h>

#include <math.h>

#define WIDTH 800

#define HEIGHT 600

#define MAXEDGES 20

typedef struct *coords* {

    float x;

    float y;

} *PointCoord*;

*PointCoord* EdgeCoords[MAXEDGES];

*PointCoord* IntersectionCoords[MAXEDGES];

// start and end coordinates of the clipping plane

*PointCoord* start, end;

// x, y mouse coordinates of the clicked locations

*PointCoord* mouseCoords;

int winID; // windows ID for destroying window

/\* odd is the variable that measures the clicks of the user,

  depending at the mode selected by the menu,

  and do stuff accordingly in the mouse function \*/

int odd = 0, mode = 0;

/\* variable that controls if the clipping plane

should be toggled or not \*/

int toggle = 0;

int width = WIDTH;

int height = HEIGHT;

float red = 1.0, green = 0.0, blue = 0.0;

void leftClipper() {

    int i = 0, j = 0;

    for (i = 0; i < odd; i++) {

        // Case-1:  outside to inside

        if (EdgeCoords[i].x < start.x && EdgeCoords[i + 1].x >= start.x) {

            if (EdgeCoords[i + 1].x - EdgeCoords[i].x != 0) {

                // save point of intersection

                IntersectionCoords[j].y = (EdgeCoords[i + 1].y - EdgeCoords[i].y) /

                (EdgeCoords[i + 1].x - EdgeCoords[i].x) \*

                (start.x - EdgeCoords[i].x) + EdgeCoords[i].y;

            }

            else {

                IntersectionCoords[j].y = EdgeCoords[i].y;

            }

            IntersectionCoords[j].x = start.x;

            j++;

            // save the point that lie inside the clipping plane

            IntersectionCoords[j].x = EdgeCoords[i + 1].x;

            IntersectionCoords[j].y = EdgeCoords[i + 1].y;

            j++;

        }

        // Case-2: inside to inside

        if (EdgeCoords[i].x >= start.x && EdgeCoords[i + 1].x >= start.x) {

            // only save second point that lie inside our clipping plane

            IntersectionCoords[j].y = EdgeCoords[i + 1].y;

            IntersectionCoords[j].x = EdgeCoords[i + 1].x;

            j++;

        }

        // Case-3: inside to outside

        if (EdgeCoords[i].x >= start.x && EdgeCoords[i + 1].x < start.x) {

            if (EdgeCoords[i + 1].x - EdgeCoords[i].x != 0) {

                IntersectionCoords[j].y = (EdgeCoords[i + 1].y - EdgeCoords[i].y) /

                (EdgeCoords[i + 1].x - EdgeCoords[i].x) \*

                (start.x - EdgeCoords[i].x) + EdgeCoords[i].y;

            }

            else {

                IntersectionCoords[j].y = EdgeCoords[i].y;

            }

            IntersectionCoords[j].x = start.x;

            j++;

        }

    }

    for (i = 0; i < j; i++) {

        EdgeCoords[i].x = IntersectionCoords[i].x;

        EdgeCoords[i].y = IntersectionCoords[i].y;

    }

    EdgeCoords[i].x = IntersectionCoords[0].x;

    EdgeCoords[i].y = IntersectionCoords[0].y;

    odd = j;

}

void rightClipper() {

    int i = 0, j = 0;

    for (i = 0; i < odd; i++) {

        // Case-1:  outside to inside

        if (EdgeCoords[i].x > end.x && EdgeCoords[i + 1].x <= end.x) {

            if (EdgeCoords[i + 1].x - EdgeCoords[i].x != 0) {

                // save point of intersection

                IntersectionCoords[j].y = (EdgeCoords[i + 1].y - EdgeCoords[i].y) /

                (EdgeCoords[i + 1].x - EdgeCoords[i].x) \*

                (end.x - EdgeCoords[i].x) + EdgeCoords[i].y;

            }

            else {

                IntersectionCoords[j].y = EdgeCoords[i].y;

            }

            IntersectionCoords[j].x = end.x;

            j++;

            // save the point that lie inside the clipping plane

            IntersectionCoords[j].x = EdgeCoords[i + 1].x;

            IntersectionCoords[j].y = EdgeCoords[i + 1].y;

            j++;

        }

        // Case-2: inside to inside

        if (EdgeCoords[i].x <= end.x && EdgeCoords[i + 1].x <= end.x) {

            // only save second point that lie inside our clipping plane

            IntersectionCoords[j].y = EdgeCoords[i + 1].y;

            IntersectionCoords[j].x = EdgeCoords[i + 1].x;

            j++;

        }

        // Case-3: inside to outside

        if (EdgeCoords[i].x <= end.x && EdgeCoords[i + 1].x > end.x) {

            if (EdgeCoords[i + 1].x - EdgeCoords[i].x != 0) {

                IntersectionCoords[j].y = (EdgeCoords[i + 1].y - EdgeCoords[i].y) /

                (EdgeCoords[i + 1].x - EdgeCoords[i].x) \*

                (end.x - EdgeCoords[i].x) + EdgeCoords[i].y;

            }

            else {

                IntersectionCoords[j].y = EdgeCoords[i].y;

            }

            IntersectionCoords[j].x = end.x;

            j++;

        }

    }

    for (i = 0; i < j; i++) {

        EdgeCoords[i].x = IntersectionCoords[i].x;

        EdgeCoords[i].y = IntersectionCoords[i].y;

    }

    EdgeCoords[i].x = IntersectionCoords[0].x;

    EdgeCoords[i].y = IntersectionCoords[0].y;

    odd = j;

}

void topClipper() {

    int i = 0, j = 0;

    for (i = 0; i < odd; i++) {

        // Case-1:  outside to inside

        if (EdgeCoords[i].y > end.y && EdgeCoords[i + 1].y <= end.y) {

            if (EdgeCoords[i + 1].y - EdgeCoords[i].y != 0) {

                // save point of intersection

                IntersectionCoords[j].x = (EdgeCoords[i + 1].x - EdgeCoords[i].x) /

                (EdgeCoords[i + 1].y - EdgeCoords[i].y) \*

                (end.y - EdgeCoords[i].y) + EdgeCoords[i].x;

            }

            else {

                IntersectionCoords[j].x = EdgeCoords[i].x;

            }

            IntersectionCoords[j].y = end.y;

            j++;

            // save the point that lie inside the clipping plane

            IntersectionCoords[j].x = EdgeCoords[i + 1].x;

            IntersectionCoords[j].y = EdgeCoords[i + 1].y;

            j++;

        }

        // Case-2: inside to inside

        if (EdgeCoords[i].y <= end.y && EdgeCoords[i + 1].y <= end.y) {

            // only save second point that lie inside our clipping plane

            IntersectionCoords[j].y = EdgeCoords[i + 1].y;

            IntersectionCoords[j].x = EdgeCoords[i + 1].x;

            j++;

        }

        // Case-3: inside to outside

        if (EdgeCoords[i].y <= end.y && EdgeCoords[i + 1].y > end.y) {

            if (EdgeCoords[i + 1].y - EdgeCoords[i].y != 0) {

                IntersectionCoords[j].x = (EdgeCoords[i + 1].x - EdgeCoords[i].x) /

                (EdgeCoords[i + 1].y - EdgeCoords[i].y) \*

                (end.y - EdgeCoords[i].y) + EdgeCoords[i].x;

            }

            else {

                IntersectionCoords[j].x = EdgeCoords[i].x;

            }

            IntersectionCoords[j].y = end.y;

            j++;

        }

    }

    for (i = 0; i < j; i++) {

        EdgeCoords[i].x = IntersectionCoords[i].x;

        EdgeCoords[i].y = IntersectionCoords[i].y;

    }

    EdgeCoords[i].x = IntersectionCoords[0].x;

    EdgeCoords[i].y = IntersectionCoords[0].y;

    odd = j;

}

void bottomClipper() {

    int i = 0, j = 0;

    for (i = 0; i < odd; i++) {

        // Case-1:  outside to inside

        if (EdgeCoords[i].y < start.y && EdgeCoords[i + 1].y >= start.y) {

            if (EdgeCoords[i + 1].y - EdgeCoords[i].y != 0) {

                // save point of intersection

                IntersectionCoords[j].x = (EdgeCoords[i + 1].x - EdgeCoords[i].x) /

                (EdgeCoords[i + 1].y - EdgeCoords[i].y) \*

                (start.y - EdgeCoords[i].y) + EdgeCoords[i].x;

            }

            else {

                IntersectionCoords[j].x = EdgeCoords[i].x;

            }

            IntersectionCoords[j].y = start.y;

            j++;

            // save the point that lie inside the clipping plane

            IntersectionCoords[j].x = EdgeCoords[i + 1].x;

            IntersectionCoords[j].y = EdgeCoords[i + 1].y;

            j++;

        }

        // Case-2: inside to inside

        if (EdgeCoords[i].y >= start.y && EdgeCoords[i + 1].y >= start.y) {

            // only save second point that lie inside our clipping plane

            IntersectionCoords[j].x = EdgeCoords[i + 1].x;

            IntersectionCoords[j].y = EdgeCoords[i + 1].y;

            j++;

        }

        // Case-3: inside to outside

        if (EdgeCoords[i].y >= start.y && EdgeCoords[i + 1].y < start.y) {

            if (EdgeCoords[i + 1].y - EdgeCoords[i].y != 0) {

                IntersectionCoords[j].x = (EdgeCoords[i + 1].x - EdgeCoords[i].x) /

                (EdgeCoords[i + 1].y - EdgeCoords[i].y) \*

                (start.y - EdgeCoords[i].y) + EdgeCoords[i].x;

            }

            else {

                IntersectionCoords[j].x = EdgeCoords[i].x;

            }

            IntersectionCoords[j].y = start.y;

            j++;

        }

    }

    for (i = 0; i < j; i++) {

        EdgeCoords[i].x = IntersectionCoords[i].x;

        EdgeCoords[i].y = IntersectionCoords[i].y;

    }

    EdgeCoords[i].x = IntersectionCoords[0].x;

    EdgeCoords[i].y = IntersectionCoords[0].y;

    odd = j;

}

void drawpolygon()

{

    glColor3f(red, green, blue);

    glBegin(GL\_POLYGON);

    for (int i = 0; i < odd; i++) {

        glVertex3d(EdgeCoords[i].x, EdgeCoords[i].y, 1.0);

    }

    glEnd();

    glFlush();

}

void drawClippingPlane() {

    glColor3f(0.2f, 0.2f, 0.2f);

    glBegin(GL\_QUADS);

    glVertex3f(start.x, start.y, 1.0);

    glVertex3f(end.x, start.y, 1.0);

    glVertex3f(end.x, end.y, 1.0);

    glVertex3f(start.x, end.y, 1.0);

    glEnd();

    glFlush();

}

void toggleClippingPlane() {

    glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

    drawpolygon();

    if (toggle % 2 == 0) {

        drawClippingPlane();

    }

}

void redraw() {

    glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

    drawpolygon();

    drawClippingPlane();

    toggle = 1;

}

void clipPolygon() {

    if (mode == 1) {

        glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

        leftClipper();

        rightClipper();

        topClipper();

        bottomClipper();

        redraw();

    }

}

void draw\_pixel(int *x*, int *y*, float *R*, float *G*, float *B*, float *e*) {

    glBegin(GL\_POINTS);

    glColor3f(*e*\**R*, *e*\**G*, *e*\**B*);

    glVertex3i(*x*, *y*, 1);

    glEnd();

}

void \*font = GLUT\_BITMAP\_TIMES\_ROMAN\_10;

void show(int *x*, int *y*, char \**string*) {

    glColor3f(1.0, 1.0, 1.0);

    int len, i;

    glRasterPos3f(*x*, *y*, 1.0f);

    len = (int) strlen(*string*);

    for (i = 0; i < len; i++)

        glutBitmapCharacter(font, *string*[i]);

}

// reset the display and draw the axis

void resetDisplay() {

    odd = 0;

    mode = 0;

    toggle = 0;

    glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

    glClearColor(0.0, 0.0, 0.0, 0.0);

    glLoadIdentity();

    glViewport(0, 0, width, height);

    gluOrtho2D(0, width, 0, height);

    // draw axis

    for (int i = 0; i < width; i=i+10)

    {

        draw\_pixel(i, height/2, 1.0, 1.0, 1.0, 1.0);

        draw\_pixel(width/2, i, 1.0, 1.0, 1.0, 1.0);

    }

    show(0,(height/2),"- X-AXIS");

    show(width-40,(height/2),"X-AXIS");

    show((width/2),height - 10,"Y-AXIS");

    show((width/2),0,"- Y-AXIS");

    glutSwapBuffers();

    glFlush();

}

// update width and height values when on reshape

void myReshape(int *w*, int *h*) {

    width = *w*;

    height = *h*;

    resetDisplay();

}

void myDisplay() {

}

void mySpecialKeys(int *key*, int *x*, int *y*) {

    switch (*key*) {

    case GLUT\_KEY\_F1:

        if (odd >= 3) {

            drawpolygon();

            mode = 1;

        }

        break;

    default:

        break;

    }

}

void keyEvent(unsigned char *key*, int *x*, int *y*) {

    switch (*key*) {

    // Toggle Clipping Plane

    case ' ':

        if (mode == 1) {

            toggleClippingPlane();

            toggle++;

        }

        break;

    // Clip

    case 'C':

    case 'c':

        if (toggle == 1) {

            clipPolygon();

        }

        break;

    // Quit

    case 'Q':

    case 'q':

        glutDestroyWindow(winID);

        break;

    default:

        break;

    }

}

// this function takes the mouse position while moving mouse

void myMotion(*GLint* *x*, *GLint* *y*) {

    if (mode == 1) {

        end.x = *x*;

        end.y = height - *y*;

        redraw();

    }

}

void myMouse(int *button*, int *state*, *GLint* *x*, *GLint* *y*) {

    char coords[14];

    if(*button*==GLUT\_LEFT\_BUTTON && *state*==GLUT\_DOWN) {

        if (mode == 0) {

            if (odd < MAXEDGES) {

                mouseCoords.x = *x*;

                mouseCoords.y = height - *y*;

                sprintf(coords, "X=%.0f,Y=%.0f", mouseCoords.x - (width/2), mouseCoords.y - (height/2));

                show(mouseCoords.x+5, mouseCoords.y, coords);

                draw\_pixel(mouseCoords.x, mouseCoords.y, red, green, blue, 1.0);

                EdgeCoords[odd].x = mouseCoords.x;

                EdgeCoords[odd].y = mouseCoords.y;

                odd++;

            }

            EdgeCoords[odd].x = EdgeCoords[0].x;

            EdgeCoords[odd].y = EdgeCoords[0].y;

        }

        else if (mode == 1) {

            start.x = *x*;

            start.y = height - *y*;

        }

    }

    else if (*button* == GLUT\_RIGHT\_BUTTON && *state* == GLUT\_DOWN) {

        resetDisplay();

    }

    glFlush();

}

void init() {

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);

    glutInitWindowPosition(100, 100);

    glutInitWindowSize(width, height);

    winID = glutCreateWindow("Clipping Polygons");

    glEnable(GL\_DEPTH\_TEST);

    glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL\_PROJECTION);

}

int main(int *argc*, char \*\**argv*) {

    glutInit(&*argc*, *argv*);

    init();

    glutDisplayFunc(myDisplay);

    glutReshapeFunc(myReshape);

    glutSpecialFunc(mySpecialKeys);

    glutMouseFunc(myMouse);

    glutMotionFunc(myMotion);

    glutKeyboardFunc(keyEvent);

    glutMainLoop();

    return 0;

}