

OpenGL

## OpenGL documentation

| https://www.opengl.org/sdk/docs/man2/ |
| --- |

##

## **GLUT documentation**

| https://www.opengl.org/resources/libraries/glut/spec3/spec3.html |
| --- |

##

Ασκήσεις Εισαγωγικού μαθήματος

## Άσκηση #1

α. Τί είναι η OpenGL; Με ποιον τρόπο προκύπτουν οι νέες εκδόσεις της OpenGL (π.χ. η πλέον πρόσφατη έκδοση OpenGL 4.5 | Αυγουστος 2014) από τη βιομηχανία της πληροφορικής;

β. Τι είναι η GLUT; Για ποιο λόγο δημιουργήθηκε και ποιες βασικές λειτουργίες προσφέρει;

 **Άσκηση #2**

Εξετάστε την αλήθεια των παρακάτω προτάσεων δηλώνοντας ΣΩΣΤΗ ή ΛΑΘΟΣ και εξηγώντας την επιλογή σας.

1. Τα επικρατέστερα Λειτουργικά Συστήματα υποστηρίζουν την OpenGL.
2. Η OpenGL είναι κλητή (callable) με χρήση της γλώσσας C / C++ αποκλειστικά (δηλαδή υπάρχει μοναδικό language binding)
3. Το μοναδικό περιβάλλον ανάπτυξης προγραμμάτων OpenGL είναι το DEV C++
4. Η OpenGL περιέχει εντολές επιλογής (τύπου If then else)
5. Οι εντολές της OpenGL ξεκινούν με το πρόθεμα gl

**Άσκηση #3**

α. Αναζητήστε στο Web βιβλιοθήκες μεταγενέστερες της GLUT, οι οποίες καλύπτουν την ανάγκη διευκόλυνσης προγραμματισμού σε OpenGL και έχουν ουσιαστικά παραγκωνίσει τη GLUT.

β. Αναζητήστε στο web οδηγίες για τη διαμόρφωση διαφορετικού (από το DEV C ++) περιβάλλοντος συγγραφής OpenGL προγραμμάτων με χρήση της GLUT βιβλιοθήκης. Συγκρίνετε τις οδηγίες με αυτές που ακολουθήσατε για τη διαμόρφωση του περιβάλλοντος DEV C ++ / GLUT; Να εστιάσετε στις ομοιότητες.



# Τέλος ασκήσεων εισαγωγικού μαθήματος



Ασκήσεις 1ου μαθήματος: Initialization

## Εκτέλεση προγράμματος #1.1

Δημιουργήστε έργο (project) σε DEV C++ περιβάλλον και αντιγράψτε τον επόμενο πηγαίο κώδικα (source file) σύμφωνα με τα όσα αναφέρονται στον οδηγό εγκατάστασης / ελέγχου εγκατάστασης GLUT βιβλιοθήκης σε DEV C++ περιβάλλον.

| #ifdef \_\_APPLE\_\_#include <GLUT/glut.h>#else#include <GL/glut.h>#endifvoid renderScene(void) { glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); glBegin(GL\_TRIANGLES); glVertex3f(-0.5,-0.5,0.0); glVertex3f(0.5,0.0,0.0); glVertex3f(0.0,0.5,0.0); glEnd(); glutSwapBuffers();}int main(int argc, char \*\*argv) { // init GLUT and create Window glutInit(&argc, argv); glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA); glutInitWindowPosition(100,100); glutInitWindowSize(320,320); glutCreateWindow("IEK OpenGL"); // register callbacks glutDisplayFunc(renderScene); // enter GLUT event processing cycle glutMainLoop();  return 1;} |
| --- |

Επιβεβαιώστε ότι εμφανίζεται στην οθόνη σας το επόμενο παράθυρο.



Καταγράψτε τα βήματα που ακολουθήσατε από τη δημιουργία του έργου μέχρι και την εκτέλεση.

## Κατανόηση προγράμματος #1.2

Ποιος ο ρόλος του κώδικα

#ifdef \_\_APPLE\_\_
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

## Κατανόηση προγράμματος #1.3

Για ποιο λόγο η συνάρτηση renderScene τοποθετείται πριν την main;

## Κατανόηση προγράμματος #1.4

Τα ονόματα των GLUT συναρτήσεων προθεματίζονται με glut ή glutinit, π.χ. glutDestroyWindow, glutInitWindowPosition.

Θα μπορούσαμε να προθεματίσουμε με glut μια από τις custom συναρτήσεις του προγράμματος; Πειραματιστείτε. Θεωρείτε καλή προγραμματιστική ταχτική τον προθεματισμό συναρτήσεων του προγραμματιστή με “δεσμευμένα” προθέματα;

## Κατανόηση προγράμματος #1.5

Ποιος ο ρόλος της glutInitWindowPosition(int x, int y); Τί είναι τα int x, int y; Ανατρέξτε στο GLUT documentation link και βρείτε ποιες είναι οι προκαθορισμένες τιμές για τα x, y. Πειραματιστείτε με αυτές τις τιμές και με άλλες.

Ποιος ο ρόλος της glutInitWindowSize(int width, int height); Τί είναι τα int width, int height; Ανατρέξτε στο GLUT documentation link και βρείτε ποιες είναι οι προκαθορισμένες τιμές για τα width, height. Πειραματιστείτε με αυτές τις τιμές και με άλλες.

Διαγράψτε τις γραμμές κώδικα

glutInitWindowPosition(100,100);
 glutInitWindowSize(320,320);

Παρατηρείτε αλλαγές κατά την εκτέλεση του προγράμματος;

Πως εξηγείται το ότι η εφαρμογή εκτελείται “κανονικά” παρ’ ότι διαγράφηκαν δύο βασικές γραμμές κώδικα; Πως αναπληρώνεται αυτή η διαγραφή;

## **Κατανόηση προγράμματος #1.**6

Περιγράψτε την Indexed color και την RGB τεχνική (αναζήτηση στο WEB).

Ποια είναι η λειτουργία της εντολής glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

Εξηγείστε τον τρόπο “χωρισμού” των παραμέτρων της συνάρτησης glutInitDisplayMode.

Υπάρχει διαφορά στη χρήση της παραμέτρου GLUT\_RGBA από τη χρήση της GLUT\_RGB; Εξηγείστε (GLUT documentation link και WEB search)

**Κατανόηση προγράμματος #1.7**

Εξηγείστε τη λειτουργία των συναρτήσεων που καλούνται στο σώμα της custom συνάρτησης renderScene (συμβουλευτείτε το OpenGL documentation link).

void renderScene(void) {

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

 glBegin(GL\_TRIANGLES);
 glVertex3f(-0.5,-0.5,0.0);
 glVertex3f(0.5,0.0,0.0);
 glVertex3f(0.0,0.5,0.0);
 glEnd();

 glutSwapBuffers();
}

## Νέο πρόγραμμα #1.8

## Δημιουργήστε νέο πρόγραμμα το οποίο να εμφανίζει σε παράθυρο 400Χ400 δύο λευκά τρίγωνα.

## Αλλαγή διαστάσεων παραθύρου #1.9

Αλλάξτε τις διαστάσεις του παραθύρου που εμφανίζεται κατά την εκτέλεση του προγράμματος με όχι ανάλογο τρόπο (να μη διατηρείται η αναλογία ύψους/πλάτους). Τι παρατηρείτε;



# Τέλος ασκήσεων 1ου μαθήματος



Ασκήσεις 2ου μαθήματος: Preparing the window for a reshape

## Επέκταση προγράμματος #2.1

Ακολουθεί το project του 1ου μαθήματος με προσθήκες ώστε το τρίγωνο να μην παραμορφώνεται κατά την δυσανάλογη αλλαγή διαστάσεων του παραθύρου.

void changeSize(int w, int h) {

 // Prevent a divide by zero, when window is too short
 // (you cant make a window of zero width).
 if (h == 0)
 h = 1;

 float ratio = w \* 1.0 / h;

 // Use the Projection Matrix
 glMatrixMode(GL\_PROJECTION);

 // Reset Matrix
 glLoadIdentity();

 // Set the viewport to be the entire window
 glViewport(0, 0, w, h);

 // Set the correct perspective.
 gluPerspective(45,ratio,1,100);

 // Get Back to the Modelview
 glMatrixMode(GL\_MODELVIEW);
}
#ifdef \_\_APPLE\_\_
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

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

 // init GLUT and create window
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(320,320);
 glutCreateWindow("Lighthouse3D - GLUT Tutorial");

 // register callbacks
 glutDisplayFunc(renderScene);
 glutReshapeFunc(changeSize);

 // enter GLUT event processing loop
 glutMainLoop();

 return 1;
}

void renderScene(void) {

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

 glBegin(GL\_TRIANGLES);
 glVertex3f(-2,-2,-5.0);
 glVertex3f(2,0.0,-5.0);
 glVertex3f(0.0,2,-5.0);
 glEnd();

 glutSwapBuffers();
}

Ποια είναι η σωστή σειρά τοποθέτησης των χρωματιστών κομματιών κώδικα;

α. κόκκινο, μπλε, πράσινο, κίτρινο

β. κίτρινο, μπλε, πράσινο, κόκκινο

γ. πράσινο, μπλε κόκκινο, κίτρινο

δ. κόκκινο, πράσινο, μπλε, κίτρινο

ε. κόκκινο, κίτρινο, πράσινο, μπλε

Τοποθετήστε στη σωστή σειρά και εκτελέστε το επεκταμένο πρόγραμμα. Συνεχίζει να υπάρχει παραμόρφωση στο σχήμα όταν αλλάζουμε (δυσανάλογα) τις διαστάσεις του παραθύρου;

## Κατανόηση προγράμματος #2.2

α. Εξηγήστε αναλυτικά τον τρόπο λειτουργίας της εντολής (συνάρτησης) glutMainLoop(). Επιχειρήστε να συντάξετε ψευδοκώδικα για να διατυπώσετε τον αλγόριθμο λειτουργίας της.

β. Στο επόμενο τμήμα κώδικα

// register callbacks
 glutDisplayFunc(renderScene);
 glutReshapeFunc(changeSize);

το πρόγραμμα κάνει callback registrations. Τι σημαίνει αυτό;

γ. Πότε εκτελούνται οι εντολές glutDisplayFunc(renderScene) και glutReshapeFunc(changeSize);

## Κατανόηση προγράμματος #2.3

Πως εξηγείται ότι στην εντολή glViewport(0, 0, w, h) αν και δεν έχουν ευθέως αποδοθεί τιμές στις μεταβλητές w,h το πρόγραμμα μεταγλωττίσεται και εκτελείται κανονικά;



# Τέλος ασκήσεων 2ου μαθήματος



Ασκήσεις 3ου μαθήματος: Animation

## Επέκταση προγράμματος #3.1

Ακολουθεί το project του 2ου μαθήματος με προσθήκες / αλλαγές ώστε η εκτέλεσή του να καταλήγει σε περιστροφή του τριγώνου (animation)

1η αλλαγή

float angle = 0.0f;

void renderScene(void) {

 // Clear Color and Depth Buffers
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

 // Reset transformations
 glLoadIdentity();
 // Set the camera
 gluLookAt( 0.0f, 0.0f, 10.0f,
 0.0f, 0.0f, 0.0f,
 0.0f, 1.0f, 0.0f);

 glRotatef(angle, 0.0f, 1.0f, 0.0f);

 glBegin(GL\_TRIANGLES);
 glVertex3f(-2.0f,-2.0f, 0.0f);
 glVertex3f( 2.0f, 0.0f, 0.0);
 glVertex3f( 0.0f, 2.0f, 0.0);
 glEnd();

 angle+=0.1f;

 glutSwapBuffers();
}

2η αλλαγή

// register callbacks
 glutIdleFunc(renderScene);

Διατυπώστε την αλγοριθμική τεχνική που υιοθετείται προκειμένου να δοθεί περιστροφή στο τρίγωνο. Εστιάστε στο ρόλο των εντολών glutIdleFunc(renderScene) και glRotatef(angle, 0.0f, 1.0f, 0.0f) (και των παραμέτρων τους).



# Τέλος ασκήσεων 3ου μαθήματος



Ασκήσεις 4ου μαθήματος:

3D Shapes

## Κώδικας προγράμματος #4.1

Ακολουθεί ο κώδικας του προγράμματος 3D Shapes

#include <windows.h> // for MS Windows
#include <GL/glut.h> // GLUT, include glu.h and gl.h

/\* Global variables \*/
char title[] = "3D Shapes";

/\* Initialize OpenGL Graphics \*/
void initGL() {
 glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
 glClearDepth(1.0f); // Set background depth to farthest
 glEnable(GL\_DEPTH\_TEST); // Enable depth testing for z-culling
 glDepthFunc(GL\_LEQUAL); // Set the type of depth-test
 glShadeModel(GL\_SMOOTH); // Enable smooth shading
 glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST); // Nice perspective corrections
}

/\* Handler for window-repaint event. Called back when the window first appears and

 whenever the window needs to be re-painted. \*/
void display() {
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear color and depth buffers
 glMatrixMode(GL\_MODELVIEW); // To operate on model-view matrix

 // Render a color-cube consisting of 6 quads with different colors
 glLoadIdentity(); // Reset the model-view matrix
 glTranslatef(1.5f, 0.0f, -7.0f); // Move right and into the screen

 glBegin(GL\_QUADS); // Begin drawing the color cube with 6 quads
 // Top face (y = 1.0f)
 // Define vertices in counter-clockwise (CCW) order with normal pointing out
 glColor3f(0.0f, 1.0f, 0.0f); // Green
 glVertex3f( 1.0f, 1.0f, -1.0f);
 glVertex3f(-1.0f, 1.0f, -1.0f);
 glVertex3f(-1.0f, 1.0f, 1.0f);
 glVertex3f( 1.0f, 1.0f, 1.0f);

 // Bottom face (y = -1.0f)
 glColor3f(1.0f, 0.5f, 0.0f); // Orange
 glVertex3f( 1.0f, -1.0f, 1.0f);
 glVertex3f(-1.0f, -1.0f, 1.0f);
 glVertex3f(-1.0f, -1.0f, -1.0f);
 glVertex3f( 1.0f, -1.0f, -1.0f);

 // Front face (z = 1.0f)
 glColor3f(1.0f, 0.0f, 0.0f); // Red
 glVertex3f( 1.0f, 1.0f, 1.0f);
 glVertex3f(-1.0f, 1.0f, 1.0f);
 glVertex3f(-1.0f, -1.0f, 1.0f);
 glVertex3f( 1.0f, -1.0f, 1.0f);

 // Back face (z = -1.0f)
 glColor3f(1.0f, 1.0f, 0.0f); // Yellow
 glVertex3f( 1.0f, -1.0f, -1.0f);
 glVertex3f(-1.0f, -1.0f, -1.0f);
 glVertex3f(-1.0f, 1.0f, -1.0f);
 glVertex3f( 1.0f, 1.0f, -1.0f);

 // Left face (x = -1.0f)
 glColor3f(0.0f, 0.0f, 1.0f); // Blue
 glVertex3f(-1.0f, 1.0f, 1.0f);
 glVertex3f(-1.0f, 1.0f, -1.0f);
 glVertex3f(-1.0f, -1.0f, -1.0f);
 glVertex3f(-1.0f, -1.0f, 1.0f);

 // Right face (x = 1.0f)
 glColor3f(1.0f, 0.0f, 1.0f); // Magenta
 glVertex3f(1.0f, 1.0f, -1.0f);
 glVertex3f(1.0f, 1.0f, 1.0f);
 glVertex3f(1.0f, -1.0f, 1.0f);
 glVertex3f(1.0f, -1.0f, -1.0f);
 glEnd(); // End of drawing color-cube

 // Render a pyramid consists of 4 triangles
 glLoadIdentity(); // Reset the model-view matrix
 glTranslatef(-1.5f, 0.0f, -6.0f); // Move left and into the screen

 glBegin(GL\_TRIANGLES); // Begin drawing the pyramid with 4 triangles
 // Front
 glColor3f(1.0f, 0.0f, 0.0f); // Red
 glVertex3f( 0.0f, 1.0f, 0.0f);
 glColor3f(0.0f, 1.0f, 0.0f); // Green
 glVertex3f(-1.0f, -1.0f, 1.0f);
 glColor3f(0.0f, 0.0f, 1.0f); // Blue
 glVertex3f(1.0f, -1.0f, 1.0f);

 // Right
 glColor3f(1.0f, 0.0f, 0.0f); // Red
 glVertex3f(0.0f, 1.0f, 0.0f);
 glColor3f(0.0f, 0.0f, 1.0f); // Blue
 glVertex3f(1.0f, -1.0f, 1.0f);
 glColor3f(0.0f, 1.0f, 0.0f); // Green
 glVertex3f(1.0f, -1.0f, -1.0f);

 // Back
 glColor3f(1.0f, 0.0f, 0.0f); // Red
 glVertex3f(0.0f, 1.0f, 0.0f);
 glColor3f(0.0f, 1.0f, 0.0f); // Green
 glVertex3f(1.0f, -1.0f, -1.0f);
 glColor3f(0.0f, 0.0f, 1.0f); // Blue
 glVertex3f(-1.0f, -1.0f, -1.0f);

 // Left
 glColor3f(1.0f,0.0f,0.0f); // Red
 glVertex3f( 0.0f, 1.0f, 0.0f);
 glColor3f(0.0f,0.0f,1.0f); // Blue
 glVertex3f(-1.0f,-1.0f,-1.0f);
 glColor3f(0.0f,1.0f,0.0f); // Green
 glVertex3f(-1.0f,-1.0f, 1.0f);
 glEnd(); // Done drawing the pyramid

 glutSwapBuffers(); // Swap the front and back frame buffers (double buffering)
}

/\* Handler for window re-size event. Called back when the window first appears and whenever the window is re-sized with its new width and height \*/
void reshape(GLsizei width, GLsizei height) { // GLsizei for non-negative integer
 // Compute aspect ratio of the new window
 if (height == 0) height = 1; // To prevent divide by 0
 GLfloat aspect = (GLfloat)width / (GLfloat)height;

 // Set the viewport to cover the new window
 glViewport(0, 0, width, height);

 // Set the aspect ratio of the clipping volume to match the viewport
 glMatrixMode(GL\_PROJECTION); // To operate on the Projection matrix
 glLoadIdentity(); // Reset
 // Enable perspective projection with fovy, aspect, zNear and zFar
 gluPerspective(45.0f, aspect, 0.1f, 100.0f);
}

/\* Main function: GLUT runs as a console application starting at main() \*/
int main(int argc, char\*\* argv) {
 glutInit(&argc, argv); // Initialize GLUT
 glutInitDisplayMode(GLUT\_DOUBLE); // Enable double buffered mode
 glutInitWindowSize(640, 480); // Set the window's initial width & height
 glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
 glutCreateWindow(title); // Create window with the given title
 glutDisplayFunc(display); // Register callback handler for window re-paint event
 glutReshapeFunc(reshape); // Register callback handler for window re-size event
 initGL(); // Our own OpenGL initialization
 glutMainLoop(); // Enter the infinite event-processing loop
 return 0;
}

**Κατανόηση προγράμματος #4.2**

Στην εντολή glutDisplayFunc(display)

1. Ποιο είναι το συμβάν (event) που πυροδοτεί την glutDisplayFunc();
2. Σε ποιες περιπτώσεις εμφανίζεται το παραπάνω συμβάν;
3. Ποια είναι η συμπεριφορά της μηχανής (υποσύστημα γραφικών) κάθε φορά που εκδηλώνεται το συμβάν;

**Κατανόηση προγράμματος #4.3**

Τι κάνουν οι εντολές glClearColor(0.0f, 0.0f, 0.0f, 1.0f), glClearDepth(1.0f) και glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); Αναφερθείτε αναλυτικά τόσο στις εντολές όσο και στις παραμέτρους.

**Κατανόηση προγράμματος #4.4**

Τι κάνουν οι εντολές glEnable(GL\_DEPTH\_TEST) και glDepthFunc(GL\_LEQUAL); Γιατί τις χρειάζεται η εντολή glClear(GL\_DEPTH\_BUFFER\_BIT) - περιγράψτε με λεπτομέρεια.

**Κατανόηση προγράμματος #4.5**

Περιγράψτε όσο το δυνατό πληρέστερα πώς αναπαρίσταται ένα 3D γραφικό στην οθόνη. Χρησιμοποιείστε

* το σχήμα (αγνοήστε τα επίπεδα Lighting και Scan Conversion).



* Το εναλλακτικό σχήμα



Ποιες εντολές του προγράμματος υλοποιούν τους επόμενους μετασχηματισμούς (οι οποίοι φυσικά αντιστοιχίζονται στα επίπεδα των παραπάνω σχημάτων).

* MODEL το WORLD μετασχηματισμό
* WORLD to VIEW μετασχηματισμό
* VIEW to PROJECTION μετασχηματισμό

**Κατανόηση προγράμματος #4.6**

Ποια είδη Projection γνωρίζετε; Αναζητήστε στο WEB σχήματα που βοηθούν στην κατανόησή τους.

Σε ποια περίπτωση και με ποιες εντολές χρησιμοποιούμε το κάθε είδος Projection;

**Κατανόηση προγράμματος #4.7**

Κάντε τις απαραίτητες προγραμματιστικές επεμβάσεις στο πρόγραμμα ώστε

1. το χρώμα φόντου να γίνει λευκό
2. οι διαστάσεις του παραθύρου να μεγαλώσουν (επιλέξτε εσείς το μέγεθος)
3. ο τίτλος του παραθύρου να γίνει “Γ’ ΕΞΑΜΗΝΟ”
4. η πυραμίδα να μετακινηθεί ελαφρώς πιο αριστερά και ο κύβος ελαφρώς πιο δεξιά
5. οι έδρες της πυραμίδας να είναι μονόχρωμες και του κύβου gradient (μεταξύ χρωμάτων της επιλογής σας)
6. η πυραμίδα να περιστρέφεται (συνδυάστε με το προηγούμενο μάθημα - animation)



# Τέλος ασκήσεων 4ου μαθήματος



Ασκήσεις 5ου μαθήματος:

[**Κίνηση / Animation #**](http://users.sch.gr/papalouk/index.php/2015-11-30-14-39-41/2015-12-01-11-35-19/opengl/108-opengl-5-animation)

**ΑΣΚΗΣΗ 1η**

Εγκαταστήστε στον προσωπικό σας Η/Υ ένα περιβάλλον ανάπτυξης OpenGL προγραμμάτων (DEV C++, Eclipse CDK, Visual Studio).

**ΑΣΚΗΣΗ 2η**

Αντιγράψτε τον κώδικα του προγράμματος Animation#2 στο περιβάλλον ανάπτυξης OpenGL προγραμμάτων. Για διευκόλυνσή σας παρατίθεται ο κώδικας παρακάτω.

/\*
 \* OGL02Animation.cpp: 3D Shapes with animation
 \*/
#include <windows.h> // for MS Windows
#include <GL/glut.h> // GLUT, include glu.h and gl.h

/\* Global variables \*/
char title[] = "3D Shapes with animation";
GLfloat anglePyramid = 0.0f; // Rotational angle for pyramid [NEW]
GLfloat angleCube = 0.0f; // Rotational angle for cube [NEW]
int refreshMills = 15; // refresh interval in milliseconds [NEW]

/\* Initialize OpenGL Graphics \*/
void initGL() {
 glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
 glClearDepth(1.0f); // Set background depth to farthest
 glEnable(GL\_DEPTH\_TEST); // Enable depth testing for z-culling
 glDepthFunc(GL\_LEQUAL); // Set the type of depth-test
 glShadeModel(GL\_SMOOTH); // Enable smooth shading
 glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST); // Nice perspective corrections
}

/\* Handler for window-repaint event. Called back when the window first appears and
 whenever the window needs to be re-painted. \*/
void display() {
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear color and depth buffers
 glMatrixMode(GL\_MODELVIEW); // To operate on model-view matrix

 // Render a color-cube consisting of 6 quads with different colors
 glLoadIdentity(); // Reset the model-view matrix
 glTranslatef(1.5f, 0.0f, -7.0f); // Move right and into the screen
 glRotatef(angleCube, 1.0f, 1.0f, 1.0f); // Rotate about (1,1,1)-axis [NEW]

 glBegin(GL\_QUADS); // Begin drawing the color cube with 6 quads
 // Top face (y = 1.0f)
 // Define vertices in counter-clockwise (CCW) order with normal pointing out
 glColor3f(0.0f, 1.0f, 0.0f); // Green
 glVertex3f( 1.0f, 1.0f, -1.0f);
 glVertex3f(-1.0f, 1.0f, -1.0f);
 glVertex3f(-1.0f, 1.0f, 1.0f);
 glVertex3f( 1.0f, 1.0f, 1.0f);

 // Bottom face (y = -1.0f)
 glColor3f(1.0f, 0.5f, 0.0f); // Orange
 glVertex3f( 1.0f, -1.0f, 1.0f);
 glVertex3f(-1.0f, -1.0f, 1.0f);
 glVertex3f(-1.0f, -1.0f, -1.0f);
 glVertex3f( 1.0f, -1.0f, -1.0f);

 // Front face (z = 1.0f)
 glColor3f(1.0f, 0.0f, 0.0f); // Red
 glVertex3f( 1.0f, 1.0f, 1.0f);
 glVertex3f(-1.0f, 1.0f, 1.0f);
 glVertex3f(-1.0f, -1.0f, 1.0f);
 glVertex3f( 1.0f, -1.0f, 1.0f);

 // Back face (z = -1.0f)
 glColor3f(1.0f, 1.0f, 0.0f); // Yellow
 glVertex3f( 1.0f, -1.0f, -1.0f);
 glVertex3f(-1.0f, -1.0f, -1.0f);
 glVertex3f(-1.0f, 1.0f, -1.0f);
 glVertex3f( 1.0f, 1.0f, -1.0f);

 // Left face (x = -1.0f)
 glColor3f(0.0f, 0.0f, 1.0f); // Blue
 glVertex3f(-1.0f, 1.0f, 1.0f);
 glVertex3f(-1.0f, 1.0f, -1.0f);
 glVertex3f(-1.0f, -1.0f, -1.0f);
 glVertex3f(-1.0f, -1.0f, 1.0f);

 // Right face (x = 1.0f)
 glColor3f(1.0f, 0.0f, 1.0f); // Magenta
 glVertex3f(1.0f, 1.0f, -1.0f);
 glVertex3f(1.0f, 1.0f, 1.0f);
 glVertex3f(1.0f, -1.0f, 1.0f);
 glVertex3f(1.0f, -1.0f, -1.0f);
 glEnd(); // End of drawing color-cube

 // Render a pyramid consists of 4 triangles
 glLoadIdentity(); // Reset the model-view matrix
 glTranslatef(-1.5f, 0.0f, -6.0f); // Move left and into the screen
 glRotatef(anglePyramid, 1.0f, 1.0f, 0.0f); // Rotate about the (1,1,0)-axis [NEW]

 glBegin(GL\_TRIANGLES); // Begin drawing the pyramid with 4 triangles
 // Front
 glColor3f(1.0f, 0.0f, 0.0f); // Red
 glVertex3f( 0.0f, 1.0f, 0.0f);
 glColor3f(0.0f, 1.0f, 0.0f); // Green
 glVertex3f(-1.0f, -1.0f, 1.0f);
 glColor3f(0.0f, 0.0f, 1.0f); // Blue
 glVertex3f(1.0f, -1.0f, 1.0f);

 // Right
 glColor3f(1.0f, 0.0f, 0.0f); // Red
 glVertex3f(0.0f, 1.0f, 0.0f);
 glColor3f(0.0f, 0.0f, 1.0f); // Blue
 glVertex3f(1.0f, -1.0f, 1.0f);
 glColor3f(0.0f, 1.0f, 0.0f); // Green
 glVertex3f(1.0f, -1.0f, -1.0f);

 // Back
 glColor3f(1.0f, 0.0f, 0.0f); // Red
 glVertex3f(0.0f, 1.0f, 0.0f);
 glColor3f(0.0f, 1.0f, 0.0f); // Green
 glVertex3f(1.0f, -1.0f, -1.0f);
 glColor3f(0.0f, 0.0f, 1.0f); // Blue
 glVertex3f(-1.0f, -1.0f, -1.0f);

 // Left
 glColor3f(1.0f,0.0f,0.0f); // Red
 glVertex3f( 0.0f, 1.0f, 0.0f);
 glColor3f(0.0f,0.0f,1.0f); // Blue
 glVertex3f(-1.0f,-1.0f,-1.0f);
 glColor3f(0.0f,1.0f,0.0f); // Green
 glVertex3f(-1.0f,-1.0f, 1.0f);
 glEnd(); // Done drawing the pyramid

 glutSwapBuffers(); // Swap the front and back frame buffers (double buffering)

 // Update the rotational angle after each refresh [NEW]
 anglePyramid += 0.2f;
 angleCube -= 0.15f;
}

/\* Called back when timer expired [NEW] \*/
void timer(int value) {
 glutPostRedisplay(); // Post re-paint request to activate display()
 glutTimerFunc(refreshMills, timer, 0); // next timer call milliseconds later
}

/\* Handler for window re-size event. Called back when the window first appears and
 whenever the window is re-sized with its new width and height \*/
void reshape(GLsizei width, GLsizei height) { // GLsizei for non-negative integer
 // Compute aspect ratio of the new window
 if (height == 0) height = 1; // To prevent divide by 0
 GLfloat aspect = (GLfloat)width / (GLfloat)height;

 // Set the viewport to cover the new window
 glViewport(0, 0, width, height);

 // Set the aspect ratio of the clipping volume to match the viewport
 glMatrixMode(GL\_PROJECTION); // To operate on the Projection matrix
 glLoadIdentity(); // Reset
 // Enable perspective projection with fovy, aspect, zNear and zFar
 gluPerspective(45.0f, aspect, 0.1f, 100.0f);
}

/\* Main function: GLUT runs as a console application starting at main() \*/
int main(int argc, char\*\* argv) {
 glutInit(&argc, argv); // Initialize GLUT
 glutInitDisplayMode(GLUT\_DOUBLE); // Enable double buffered mode
 glutInitWindowSize(640, 480); // Set the window's initial width & height
 glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
 glutCreateWindow(title); // Create window with the given title
 glutDisplayFunc(display); // Register callback handler for window re-paint event
 glutReshapeFunc(reshape); // Register callback handler for window re-size event
 initGL(); // Our own OpenGL initialization
 glutTimerFunc(0, timer, 0); // First timer call immediately [NEW]
 glutMainLoop(); // Enter the infinite event-processing loop
 return 0;
}

**ΑΣΚΗΣΗ 3η**

**α.** glRotatef(angleCube, 1.0f, 1.0f, 1.0f);

Ποιο είναι το αναμενόμενο αποτέλεσμα από την εκτέλεση της εντολής glRotatef(angleCube, 1.0f, 1.0f, 1.0f);

**β.** glRotatef(anglePyramid, 1.0f, 1.0f, 0.0f);

Ποιο είναι το αναμενόμενο αποτέλεσμα από την εκτέλεση της εντολής glRotatef(anglePyramid, 1.0f, 1.0f, 0.0f);

**γ.** Αλλάξτε τους άξονες περιστροφής του κύβου και της πυραμίδας προκειμένου να κατανοήσετε το τρισδιάστατο ορθοκανονικό σύστημα συντεταγμένων (local space)

**δ.** Αλλάξτε τη γωνία περιστροφής (μεγαλώστε τη και μικρύντε τη) του κύβου και της πυραμίδας. Παρατηρείτε κάτι; Εξηγείστε.

**ΑΣΚΗΣΗ 4η**

Προσθέστε στην πυραμίδα του προγράμματος την έδρα που λείπει (την τετράγωνη βάση του).

**ΑΣΚΗΣΗ 5η**

Εξηγείστε με λεπτομέρεια τον τρόπο που οι επόμενες γραμμές κώδικα επηρεάζουν το πρόγραμμα.

/\* Called back when timer expired [NEW] \*/
void timer(int value) {
 glutPostRedisplay(); // Post re-paint request to activate display()
 glutTimerFunc(refreshMills, timer, 0); // next timer call milliseconds later

}

 και

glutTimerFunc(0, timer, 0); // First timer call immediately [NEW]

Επιμείνετε στην εξήγηση της αναδρομικότητας της συνάρτησης timer. Πειραματιστείτε δίνοντας μεγάλες τιμές στη μεταβλητή refreshMills.

**ΑΣΚΗΣΗ 6η**

**α.** Μετατρέψτε το πρόγραμμα Animation #1 αντικαθιστώντας την glutIdleFunc function με την glutTimerFunc.

**β.** Ποια πλεονεκτήματα προσφέρει η νέα λύση; (αναζητήστε στο WEB την απάντηση)

**Απάντηση στην ΑΣΚΗΣΗ 6η**

#ifdef \_\_APPLE\_\_

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

/\* Called back when timer expired [NEW] \*/

void timer(int value) {

 glutPostRedisplay(); // Post re-paint request to activate display()

 glutTimerFunc(0, timer, 0); // next timer call milliseconds later

}

void changeSize(int w, int h) {

 // Prevent a divide by zero, when window is too short

 // (you cant make a window of zero width).

 if (h == 0)

 h = 1;

 float ratio = w \* 1.0 / h;

 // Use the Projection Matrix

 glMatrixMode(GL\_PROJECTION);

 // Reset Matrix

 glLoadIdentity();

 // Set the viewport to be the entire window

 glViewport(0, 0, w, h);

 // Set the correct perspective.

 gluPerspective(45.0f, ratio, 0.1f, 100.0f);

 // Get Back to the Modelview

 glMatrixMode(GL\_MODELVIEW);

}

float angle = 0.0f;

void renderScene(void) {

 // Clear Color and Depth Buffers

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

 // Reset transformations

 glLoadIdentity();

 // Set the camera

 gluLookAt( 0.0f, 0.0f, 10.0f,

 0.0f, 0.0f, 0.0f,

 0.0f, 1.0f, 0.0f);

 glRotatef(angle, 0.0f, 1.0f, 0.0f);

 glBegin(GL\_TRIANGLES);

 glVertex3f(-2.0f,-2.0f, 0.0f);

 glVertex3f( 2.0f, 0.0f, 0.0);

 glVertex3f( 0.0f, 2.0f, 0.0);

 glEnd();

 angle+=0.1f;

 glutSwapBuffers();

}

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

 // init GLUT and create window

 glutInit(&argc, argv);

 glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

 glutInitWindowPosition(100,100);

 glutInitWindowSize(320,320);

 glutCreateWindow("IEK");

 // register callbacks

 glutDisplayFunc(renderScene);

 glutReshapeFunc(changeSize);

 glutTimerFunc(0, timer, 0); // First timer call immediately [NEW]

 // enter GLUT event processing cycle

 glutMainLoop();

 return 1;

}



# Τέλος ασκήσεων 5ου μαθήματος



Ασκήσεις 6ου μαθήματος: Αλληλεπίδραση χρήστη (πληκτρολόγιο)

**Άσκηση 1η**

**α.** Αντιγράψτε τον κώδικα του μαθήματος στο περιβάλλον ανάπτυξης OpenGL προγραμμάτων. Εκτελέστε τον κώδικα και αλληλεπιδράστε με την εφαρμογή με τη βοήθεια του πληκτρολογίου, προκειμένου να ελέγξετε αν η συμπεριφορά της είναι η αναμενόμενη.

**β.** Παρατηρείτε αυτομάτως με το πάτημα των F1, F2, F3 την αλλαγή στο χρώμα του τριγώνου;

**γ.** Πατήστε click στην ενεργή περιοχή του παραθύρου μετά από κάθε πάτημα F1, F2, F3 και δείτε ότι τώρα το χρώμα αλλάζει. Τροποποιήστε τον κώδικα ώστε να υπάρχει αυτόματη αλλαγή χρώματος με το πάτημα των F1, F2, F3.

**Άσκηση 2η**

Χρησιμοποιήστε το πρόγραμμα “Κίνηση / Animation #2” που εσείς φτιάξατε (στο πλαίσιο των ασκήσεων του 5ου μαθήματος). Ενσωματώστε σε αυτό τα τμήματα κώδικα του μαθήματος 6 που δίνουν αλληλεπίδραση πληκτρολογίου στην εφαρμογή. Συγκεκριμένα:

**α.** Με το πάτημα του ESC να τερματίζει η εφαρμογή

**β.** Με το πάτημα των F1, F2, F3 να αλλάζει χρώμα μία από τις έδρες του κύβου

**Άσκηση 3η**

Επεκτείνετε το πρόγραμμα (της 1ης άσκησης) ώστε να κάνει τα εξής:

Να θέτει τη μεταβλητή green σε 0.0 όταν πατιέται το πλήκτρο g και σε 1.0 όταν πατιέται το πλήκτρο ALT+g.

Για να υλοποιήσετε την αλλαγή θα πρέπει να πειράξετε τη συνάρτηση processNormalKeys. Αναζητήστε βοήθεια στο WEB σχετικά με το πως ανιχνεύουμε το ταυτόχρονο πάτημα των πλήκτρων μετατροπής (modifier keys) όπως το ALT.

**Απάντηση:**

void processNormalKeys(unsigned char key, int x, int y) {

 if (key == 27)

 exit(0);

 else if (key=='r') {

 int mod = glutGetModifiers();

 if (mod == GLUT\_ACTIVE\_ALT)

 red = 0.0;

 else

 red = 1.0;

 }

 glutPostRedisplay();

}



Τέλος ασκήσεων 6ου μαθήματος

Ασκήσεις 7ου μαθήματος: Αλληλεπίδραση χρήστη (ποντίκι)

**Άσκηση 1η**

**α.** Αντιγράψτε τον κώδικα του μαθήματος στο περιβάλλον ανάπτυξης OpenGL προγραμμάτων. Εκτελέστε