1

Γραφικά Υπολογιστών: Εισαγωγικά

1.1 Γενικές Πληροφορίες

Σκοπός του μαθήματος αυτού είναι να παράσχει στους φοιτητές του Τμήματος Πληροφορικής μια εισαγωγή στα Γραφικά Υπολογιστών. Η διδακτέα ύλη του μαθήματος περιλαμβάνει τα εξής αντικείμενα:

Κεφάλαιο 1 Εισαγωγή στα Γραφικά. Επανάληψη βασικών εννοιών από τη Γραμμική Αλγεβρα και τις Συναρτήσεις Πολλών Μεταβλητών.
Κεφάλαιο 2 Βασικοί αλγόριθμοι σάρωσης και μετατροπής (scan-conversion) διδιάστατων αρχετύπων (primitives).
Κεφάλαιο 3 Γεωμετρικοί Μετασχηματισμοί.
Κεφάλαιο 4 Προβολές του τρισδιάστατου χώρου.
Κεφάλαιο 5 Εισαγωγή στο περιβάλλον PHIGS.
Κεφάλαιο 6 Αναπαράσταση καμπυλών και επιφανειών[1]
Κεφάλαιο 7 Προσδιορισμός ορατής επιφάνειας.
Κεφάλαιο 8 Φωτισμός και σκίαση.

Ενδεικτικό χρονοδιάγραμμα διδασκαλίας του μαθήματος δίνεται στον Πίνακα 1.1. Μοναδική απαίτηση από τους φοιτητές είναι η εξοικείωσή τους με τη γλώσσα προγραμματισμού C και το λειτουργικό σύστημα UNIX. Για τις ασκήσεις του μαθήματος μπορούν να χρησιμοποιούν τα τερματικά της νησίδας υπολογιστών ΑΡΙΣΤΟΤΕΛΗΣ.

Σαν βασικό εγχειρίδιο για το μάθημα θα χρησιμοποιηθεί το [1]. Κυκλοφορεί επίσης και νεώτερη έκδοσή του από τον ίδιο εκδοτικό οίκο το 1994. Διαθέσιμα είναι επίσης τα συγγράμματα [2,3,8]. Αλλα καλά εγχειρίδια στην περιοχή είναι τα [4]–[6] και [9].

[1] Στην παρούσα μορφή των σημειώσεων τα κεφάλαια 5 και 6 δεν καλύπτονται.

Κεφάλαιο	Παραδόσεις
1	1
2	3
3	1
4	2
7	2
8	3

Πίνακας 1.1 Χρονοδιάγραμμα διδασκαλίας του μαθήματος

Στο μάθημα δίνονται προαιρετικά βιβλιογραφικές εργασίες, καθώς και μία συλλογή τεσσάρων ασκήσεων υλοποίησης αλγορίθμων γραφικών υπό μορφή θεμάτων. Οι φοιτητές καλούνται να συγκροτήσουν ομάδες των δύο ατόμων και να αναλάβουν μία βιβλιογραφική εργασία και/ή την επίλυση της συλλογής των ασκήσεων. Η προαιρετική βιβλιογραφική εργασία συνεισφέρει δύο (2) προσθετικές μονάδες σε προβιβάσιμο βαθμό της τελικής γραπτής εξέτασης. Η παράδοση των λυμένων ασκήσεων συνεισφέρει άλλες δύο (2) επιπλέον μονάδες σε προβιβάσιμο βαθμό της τελικής γραπτής εξέτασης με την εξής αναλογία: 0.35 μονάδες για την πρώτη άσκηση και 0.55 μονάδες για καθεμιά από τις υπόλοιπες τρείς.

2

Υπολογιστικές ασκήσεις γραφικών

2.1 1η Ασκηση: Αλγόριθμος Koch για τη δημιουργία νιφάδων χιονιού

Στην πρώτη άσκηση ζητείται να γράψετε ένα πρόγραμμα που δημιουργεί τις **νιφάδες χιονιού του Koch** [2, p.426]. Θα περιγράψουμε στη συνέχεια συνοπτικά την τριαδική καμπύλη του Koch (triadic Koch curve). Ξεκινούμε από ένα ευθύγραμμο τμήμα, το υποδιαιρούμε σε τρία μέρη και αντικαθιστούμε το μεσαίο τμήμα με τις δύο προσκείμενες πλευρές ενός ισόπλευρου τριγώνου. Μια τέτοια διαδικασία μας δίνει μια καμπύλη η οποία ξεκινά και τελειώνει στα ίδια σημεία με το αρχικό ευθύγραμμο τμήμα, αλλά συντίθεται από τέσσερα ισομήκη τμήματα, που το καθένα έχει μήκος ίσο με το 1/3 του αρχικού. Επομένως, η νέα καμπύλη έχει μήκος ίσο με τα 4/3 του αρχικού μήκους. Επαναλαμβάνοντας τη διαδικασία για καθένα από τα τέσσερα τμήματα, προκύπτει μια καμπύλη με περισσότερες γωνίες και μήκος ίσο με τα 16/9 του αρχικού. Προφανώς, η ίδια διαδικασία μπορεί να γίνει αν χωριστεί το αρχικό ευθύγραμμο τμήμα σε πέντε, επτά κ.ο.κ. κομμάτια και αντικαταστήσουμε το μεσαίο με τις δύο προσκείμενες πλευρές ενός ισοπλεύρου τριγώνου. Ανάλογη γενίκευση μπορεί να γίνει και για άρτιο αριθμό τμημάτων.

Σκοπός της άσκησης είναι να σας εξοικειώσει με τη σχεδίαση γραμμών στα X-Windows και τον χειρισμό διανυσμάτων (vectors). Σας παρέχεται ο κώδικας για το interface χαμηλού επιπέδου με τα X-Windows στο αρχείο `/apps/CG/a1/simpleX.c`[1][10].

[1] Στο περιβάλλον "ΑΡΙΣΤΟΤΕΛΗΣ" εκτελείτε την εντολή **attach apps**, οπότε έχετε πρόσβαση στο filesystem `/apps/CG`. Η διαχειρίστρια του συστήματος, κ. Λευτεριώτη, τροποποίησε τα `Makefile` για κάθε υπολογιστικό σύστημα. Ετσι, ανάλογα με το σύστημα στο οποίο δουλεύετε μεταβαίνετε στα subdirectories `CG.sun`, `CG.hp`, `CG.alpha` κάτω από το `CG` όπου θα βρείτε τα τέσσερα επιμέρους subdirectories `a1`, `a2`, `a3` `a4` για τις τέσσερις ασκήσεις.

```
/*
 *  Simple X interface; John Amanatides, July 1992
 */

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xatom.h>

extern void exit();

#define BORDER      2
#define HINT_POS_X  100
#define HINT_POS_Y  100

static Display *display;
static int screen;
static Visual *visual;
static Window root, window;
static GC blackGC;

void OpenX(windowName, xRes, yRes)
char *windowName;
int xRes, yRes;
{
    unsigned long black, white;
    XEvent event;
    display= XOpenDisplay(NULL);

    /* do inits */
    if(display == NULL) {
        (void) fprintf(stderr, "Sorry, couldn't talk to X server\n");
        exit(1);
    }
```

Σχήμα 2.1: Συνεχίζεται

```
    screen= XDefaultScreen(display);
    root= XDefaultRootWindow(display);
    visual= XDefaultVisual(display, screen);
    black= XBlackPixel(display, screen);
    white= XWhitePixel(display, screen);
    blackGC= XDefaultGC(display, screen);
    window= XCreateSimpleWindow(display, root, HINT_POS_X, HINT_POS_Y,
                    xRes, yRes, BORDER, black, white);
    XStoreName(display, window, windowName);

    /* wait until window is on screen before returning */
    XMapWindow(display, window);
    XSelectInput(display, window, ExposureMask);
    XNextEvent(display, &event);
    XSelectInput(display, window, NoEventMask);
}
void DrawLineX(x1, y1, x2, y2)
int x1, y1, x2, y2;
{
    XDrawLine(display, window, blackGC, x1, y1, x2, y2);
}
void FlushX()
{
    XFlush(display);
}
void ClearX()
{
    XClearWindow(display, window);
    XFlush(display);
}
void CloseX()
{
    XCloseDisplay(display);
}
```

Σχήμα 2.1 Αρχείο simpleX.c: Συναρτήσεις για interface με X-Windows

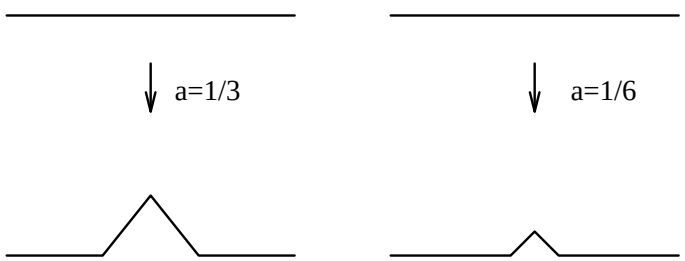
Οι ορισμοί των συναρτήσεων περιέχονται στο αρχείο `/apps/CG/a1/simpleX.h` και είναι οι εξής:

```
extern void OpenX(), CloseX(), DrawLineX(), ClearX(), FlushX();
```

Σχήμα 2.2 Αρχείο simpleX.h: Ορισμοί συναρτήσεων για interface με X-Windows

Μπορείτε να καλέσετε αυτές τις ρουτίνες ζωγραφίζοντας γραμμές. Περισσότερες πληροφορίες για τα X-Windows μπορείτε να αναζητήσετε στο [11].

Εστω `a` η παράμετρος που ελέγχει το βαθμό "συρρίκνωσης" της καμπύλης, δηλαδή, το μήκος καθενός τμήματος. Γράψτε ένα πρόγραμμα που θα δέχεται έξι παραμέτρους ως είσοδο, δηλαδή, τα τερματικά σημεία της καμπύλης `(x1,y1)` και `(x2,y2)`, τον αριθμό επαναλήψεων και την παράμετρο `a`. Η πρώτη επανάληψη του αλγορίθμου για δύο ενδεικτικές τιμές της παραμέτρου `a` δίνεται παραστατικά στο Σχήμα 2.3: Κατά την παράδοση της λύσης της άσκησης να επισυνάψετε μια εικόνα του αποτελέσματος χρησιμοποιώντας τα προγράμματα **xwd** και **xpr**.



Σχήμα 2.3 Πρώτη επανάληψη του αλγορίθμου του Koch

2.2 2η Ασκηση: Διδιάστατοι μετασχηματισμοί και ψαλίδισμα πολυγώνων

Στην άσκηση αυτή ζητείται η ανάπτυξη του προγράμματος **subscript** το οποίο υλοποιεί διδιάστατους μετασχηματισμούς και ψαλίδισμα πολυγώνων (polygon clipping). Το πρόγραμμα αυτό θα σας οδηγήσει στη θεώρηση διαφόρων συστημάτων συντεταγμένων. Αποτελεί επίσης παράδειγμα ιεραρχικής σχεδίασης λογισμικού (layered software design).

Σκοπός της άσκησης είναι να υλοποιήσετε μια πολλή απλή γλώσσα περιγραφής σελίδας (page description language) που μοιάζει με την Postscript. Η γλώσσα αυτή ασχολείται με πολύγωνα και κείμενο. Δευτερεύον παράγωγο της άσκησης είναι η "σχεδίαση" ορισμένων χαρακτήρων. Σας δίνεται ένα σημαντικό μέρος του κώδικα. Συγκεκριμένα:

1. ο διερμηνέας (interpreter) της γλώσσας **subscript** στο αρχείο `/apps/CG/a2/scanner.c` (Σχήμα 2.9). Οι ορισμοί σταθερών και συναρτήσεων που εμφανίζονται στο αρχείο αυτό παρατίθενται στο αρχείο `/apps/CG/a2/scanner.h`:
2. ο κώδικας για τον αλγόριθμο σάρωσης & μετατροπής πολυγώνου στο αρχείο `/apps/CG/a2/poly.c` (Σχήμα 2.10).
3. ο σκελετός του κώδικα που ζητείται να συμπληρώσετε στο αρχείο `/apps/CG/a2/graf2d.c` (Σχήμα 2.11) όπου οι ορισμοί των συναρτήσεων δίδονται στο αρχείο `/apps/CG/a2/graf2d.h` (Σχήμα 2.5).
4. Οι βοηθητικές ρουτίνες για τα X-Windows, όπως στην Ασκηση 1 (αρχεία `/apps/CG/a1/simpleX.h, /apps/CG/a1/simpleX.c`).

```
/* commands */
#define ERROR       0
#define QUIT        1
#define INCLUDE     2
#define SCALE       3
#define ROTATE      4
#define TRANSLATE   5
#define SHEAR       6
#define INIT_TM     7
#define PUSH_TM     8
#define POP_TM      9
#define MOVETO      10
#define LINETO      11
#define MOVE        12
```

Σχήμα 2.4: Συνεχίζεται

```
#define LINE        13
#define NEWPATH     14
#define FILL        15
#define TEXT        16
#define DEFINE_CHAR 17
#define CLEAR       18
#define CLIP        19
#define COMMENT     20
typedef struct {
    char *word;
    int code;
} Words;

static Words Commands[]= {
    "quit", QUIT,
    "include", INCLUDE,
    "scale", SCALE,
    "rotate", ROTATE,
    "translate", TRANSLATE,
    "shear", SHEAR,
    "initTM", INIT_TM,
    "pushTM", PUSH_TM,
    "popTM", POP_TM,
    "moveto", MOVETO,
    "lineto", LINETO,
    "move", MOVE,
    "line", LINE,
    "newpath", NEWPATH,
    "fill", FILL,
    "text", TEXT,
    "definechar", DEFINE_CHAR,
    "clear", CLEAR,
    "clip", CLIP,
    "%", COMMENT,
    NULL, ERROR
};
```

Σχήμα 2.4 Αρχείο scanner.h: Ορισμοί σταθερών και συναρτήσεων για τον διερμηνέα της γλώσσας **subscript**

```
typedef struct {
    double x, y;
} Point;

typedef struct {
    double minX, maxX, minY, maxY;
} ClippingRegion;

typedef struct {
    double m[3][3];
} Matrix;

typedef struct Polygon {
    Point v;
    struct Polygon *next;
} Polygon;

typedef struct PolygonPath {
    Polygon *p;
    struct PolygonPath *next;
} PolygonPath;

typedef struct StackNode {
    Matrix CTM;
    struct StackNode *next;
} StackNode;
extern void GrafStart(), GrafEnd();
extern void Scale(), Rotate(), Translate(), Shear();
extern void InitTM(), PushTM(), PopTM();
extern void Moveto(), Lineto(), Move(), Line();
extern void Newpath(), Fill(), Text(), DefineChar(), Clear();
extern void SetClipRegion(), DrawSpan();
```

Σχήμα 2.5 Αρχείο graf2d.h: Ορισμοί συναρτήσεων διδιάστατων γραφικών

Ας εξετάσουμε συνοπτικά τον διερμηνέα της γλώσσας **subscript**. Οι εντολές εισόδου έχουν ως εξής (Σημειώστε ότι επιτρέπεται μια εντολή ανά γραμμή εισόδου):

`scale` $sx\ sy$
`rotate` $angle$
`translate` $tx\ ty$
`shear` $sx\ sy$
`initTM`
`pushTM`
`popTM`

`newpath`
`moveto` $x\ y$
`lineto` $x\ y$
`move` $dx\ dy$
`line` $dx\ dy$

`fill`
`definechar` c

`text` $string$

`clip` $minX\ minY\ maxX\ maxY$
`include` $filename$
`quit`

Οι πρώτες επτά εντολές ασχολούνται με τον τρέχοντα πίνακα μετασχηματισμού (current transformation matrix - CTM). Ολα τα αντικείμενα προτού σχεδιαστούν μετασχηματίζονται από τον πίνακα αυτό. Θεωρητικά, ένας καινούριος μετασχηματισμός λαμβάνει χώρα μετά τους μετασχηματισμούς που έχουν ήδη αποθηκευτεί στο CTM. Σημειώνεται ότι η γωνία περιστροφής ορίζεται ανθωρολογιακά και ότι η αρχή των αξόνων είναι στην κάτω αριστερή γωνία του παραθύρου.

Οι υπόλοιπες πέντε εντολές ορίζουν μονοπάτια `paths`, που είναι γενικεύσεις των πολυγώνων. Τα μονοπάτια μπορούν να περιέχουν περισσότερα τμήματα και έτσι μπορούν να ορίζονται και οπές. Μπορεί κανείς να φτιάξει μιαν οπή, όταν το μονοπάτι αποτελείται από περισσότερα επικαλυπτόμενα τμήματα σε διαφορετικούς προσανατολισμούς (π.χ. ένα κατά τη φορά διαγραφής των δειχτών του ρολογιού και ένα άλλο κατά την αντίθετη φορά). Η εντολή `newpath` ξεκινά τη δημιουργία ενός μονοπατιού. Οι εντολές `move[to]` και `line[to]` μας βοηθούν να δημιουργήσουμε μονοπάτια. Προφανώς οι εντολές `move` και `line` ορίζουν σχετικές μετατοπίσεις, ενώ οι `moveto` και `lineto` ορίζουν απόλυτες μετατοπίσεις.

Η εντολή `fill` παίρνει το τρέχον μονοπάτι και το παραγεμίζει. Χρησιμοποιεί τον τρέχοντα πίνακα μετασχηματισμού για να μετασχηματίσει το μονοπάτι. Στη συνέχεια, καταστρέφει το τρέχον μονοπάτι και επιτρέπει τη δημιουργία ενός νέου.

Η εντολή `definechar` παίρνει το τρέχον μονοπάτι και το χρησιμοποιεί για να ορίσει το γράμμα *c*. Στη συνέχεια καταστρέφει το μονοπάτι.
Η εντολή `text` σχεδιάζει τη συμβολοσειρά *string* χρησιμοποιώντας τον CTM.
Η εντολή `clip` επιτρέπει τον ορισμό μιας περιοχής ψαλιδίσματος. Η εντολή `include` επιτρέπει την ενσωμάτωση και ερμηνεία εντολών από αρχείο. Μπορεί να χρησιμοποιηθεί για να εισαχθούν οι ορισμοί χαρακτήρων π.χ.

```
include F
```

όπου το γράμμα F έχει περιγραφεί σε ένα αρχείο (π.χ. `/apps/CG/a2/F`) ως εξής:

```
% Note: letters are max 1 unit high and 0.5 units wide.
newpath
moveto 0 0
lineto 0 1
lineto 0.5 1
lineto 0.5 0.9
lineto 0.1 0.9
lineto 0.1 0.6
lineto 0.5 0.6
lineto 0.5 0.5
lineto 0.1 0.5
lineto 0.1 0
definechar F
```

Σχήμα 2.6 Περιγραφή του χαρακτήρα F

Η εντολή `quit` σταματά την εκτέλεση του προγράμματος. Δύο παραδείγματα απλών γραφικών παραστάσεων που μπορείτε να εκτελέσετε με το πρόγραμμα **subscript** δίνονται στο Σχήμα 2.7.

Για τη διευκόλυνσή σας παρατίθεται και το `/apps/CG/a2/Makefile` που παράγει το εκτελέσιμο πρόγραμμα στο Σχήμα 2.8. Δίδονται πλήρως οι κώδικες για τα `/apps/CG/a2/scanner.c`, `/apps/CG/a2/poly.c`, `/apps/CG/a2/simpleX.c` (ίδιο με αυτό της 1ης άσκησης) και τμήματα του κώδικα `/apps/CG/a2/graf2d.c` και `/apps/CG/a2/ clip.c`. Ζητούμενα: συμπλήρωση του κώδικα `graf2d.c` και `clip.c`. Οι δοσμένοι κώδικες παρατίθενται στα Σχήματα 2.9–2.12. Κατά την παράδοση της λύσης της άσκησης να επισυνάψετε μια εικόνα αποτελεσμάτων εφαρμογής που έχετε υλοποιήσει με το πρόγραμμα **subscript** χρησιμοποιώντας τα προγράμματα **xwd** και **xpr**.

```
% draw a simple triangle
scale 0.1 0.1
newpath
moveto 2 2
lineto 8 2
lineto 5 7
fill

% draw the text 'Foo' along the left edge going up vertically
include F
include o
scale 2 1   % really stretch out the text
rotate 90
translate 0.15 0.1
text Foo
```

Σχήμα 2.7 Απλές γραφικές παραστάσεις με τη χρήση του προγράμματος **subscript**

```
CFLAGS= -g
OBJ= scanner.o graf2d.o poly.o simpleX.o clip.o
SRC= scanner.c graf2d.c poly.c simpleX.c clip.c
LIB= -lX11 -lm

subscript: $(OBJ)
    cc -o subscript $(CFLAGS) $(OBJ) $(LIB)

clean:
    rm -f subscript $(OBJ)
lint:
    lint -u $(SRC)

graf2d.o poly.o clip.o: graf2d.h
scanner.o: scanner.h
simpleX.o graf2d.o scanner.o: simpleX.h
```

Σχήμα 2.8 Makefile που παράγει το εκτελέσιμο πρόγραμμα **subscript**

```
/*
 * front end for 2d graphics program; scan input and call appropriate functions
 *
 * written by John Amanatides, copyright (c) 1987, 1995
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "scanner.h"
#include "graf2d.h"


#define MAX_IN_DEPTH 8
#define LINE_LENGTH 256
#define OUTPUT(string) if(input == stdin) ((void) printf(string))

/*
 * decode first word of string
 */
static int
DecodeWord(list, string)
Words *list;
char *string;
{
int i, result;
char command[LINE_LENGTH];

if(string == NULL || *string == '\n')
return COMMENT;
(void) sscanf(string, "%s", command);

result= ERROR;
for(i=0;list[i].word != NULL && result == ERROR; i++) {
if(strcmp(command,list[i].word) == 0)
result= list[i].code;
}
return result;
}
```

Σχήμα 2.9: Συνεχίζεται

```
/*
 * perform scanning of input
 */
static void
Dialogue(input, indirectDepth)
FILE *input;
int indirectDepth;
{
FILE *newFile;
char line[LINE_LENGTH], stringArg[LINE_LENGTH];
double x, y, x2, y2;

while(1) {
OUTPUT("? ");
if(fgets(line, LINE_LENGTH, input) == NULL)
return; /* End of File */

switch(DecodeWord(Commands, line)) {

case QUIT:
return;
case INCLUDE:
if(indirectDepth >= MAX_IN_DEPTH) {
(void) printf("Sorry, the depth of indirection is too great.\n");
break;
}
(void) sscanf(line,"%*s %s", stringArg);
if((newFile= fopen(stringArg,"r")) == NULL)
(void) printf("Sorry, cannot open file '%s' for input.\n",stringArg);
else {
Dialogue(newFile, indirectDepth+1);
(void) fclose(newFile);
}
break;
case SCALE:
(void) sscanf(line, "%*s %lf %lf", &x, &y);
Scale(x, y);
break;
case ROTATE:
(void) sscanf(line, "%*s %lf", &x);
Rotate(x);
break;
```

Σχήμα 2.9: Συνεχίζεται

```
case TRANSLATE:
(void) sscanf(line, "%*s %lf %lf", &x, &y);
Translate(x, y);
break;
case SHEAR:
(void) sscanf(line, "%*s %lf %lf", &x, &y);
Shear(x, y);
break;
case INIT_TM:
InitTM();
break;
case PUSH_TM:
PushTM();
break;
case POP_TM:
PopTM();
break;
case MOVETO:
(void) sscanf(line, "%*s %lf %lf", &x, &y);
Moveto(x, y);
break;
case LINETO:
(void) sscanf(line, "%*s %lf %lf", &x, &y);
Lineto(x, y);
break;
case MOVE:
(void) sscanf(line, "%*s %lf %lf", &x, &y);
Move(x, y);
break;
case LINE:
(void) sscanf(line, "%*s %lf %lf", &x, &y);
Line(x, y);
break;
case NEWPATH:
Newpath();
break;
case FILL:
Fill();
break;
```

Σχήμα 2.9: Συνεχίζεται

```
case TEXT:
(void) sscanf(line,"%*s %s", stringArg);
Text(stringArg);
break;
case DEFINE_CHAR:
(void) sscanf(line,"%*s %s", stringArg);
DefineChar(stringArg[0]);
break;
case CLEAR:
Clear();
break;
case CLIP:
(void) sscanf(line, "%*s %lf %lf %lf %lf",
&x, &y, &x2, &y2);
SetClipRegion(x, y, x2, y2);
break;
case COMMENT:
break;
case ERROR:
default:
(void) printf("Eh?\n");
break;
}
}
}


main(argc, argv)
int argc;
char *argv[];
{
GrafStart(argv[0]);
Dialogue(stdin, 1);
GrafEnd();
return 0;
}
```

Σχήμα 2.9 Αρχείο scanner.c: Διερμηνέας της γλώσσας **subscript**

```
/*
Copyright (c) 1988, 1995 John Amanatides, all rights reserved.

This module scan converts polygons that can be self-intersecting
and have holes.  The holes must be of opposite orientation.
(For example, if the vertices of the exterior edge are clockwise,
the verticies of the hole(s) must be counter-clockwise.)

Note: it draws spans in the interval [minY...maxY) by calling the
function DrawSpan(int y, double lx, double rx)

The algorithm was taken out of the Newell and Sequin paper:
"The inside story on self-intersecting polygons"
LAMBDA Second Quarter 1980 pp 20-24.
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "graf2d.h"

extern void DrawSpan();


#define UP 1
#define DOWN -1

#define ON_X 1
#define ON_Y 2

typedef struct {
int direction;
double xcurr, dx;
int ymin;
double yupper;
} Edge;

typedef struct List {
struct List *next;
Edge *edge;
} List;
```

Σχήμα 2.10: Συνεχίζεται

```
static void *mem1, *mem2;
static int memoryAllocated;


static void
SortEdges(flag, list)
int flag;
List *list;
{
List * ptr;
Edge * ePtr;

/*
The sort algorithm is O(n*n).
A better one should by used for the y sort.
*/

if (flag == ON_Y) {
for (; list != NULL; list = list->next) {
for (ptr = list->next; ptr != NULL; ptr = ptr->next) {
if (list->edge->ymin > ptr->edge->ymin) {
ePtr = list->edge;
list->edge = ptr->edge;
ptr->edge = ePtr;
}
}
}
} else  {
for (; list != NULL; list = list->next) {
for (ptr = list->next; ptr != NULL; ptr = ptr->next) {
if (list->edge->xcurr > ptr->edge->xcurr) {
ePtr = list->edge;
list->edge = ptr->edge;
ptr->edge = ePtr;
}
}
}
}
}
```

Σχήμα 2.10: Συνεχίζεται

```
static List *
AddEdgesToList(poly)
PolygonPath *poly;
{
int i, numEdges = 0, totalVert = 0;
List *list;
Edge *edges;
Point *v1, *v2;
double dx, ymin;
Polygon *p;
PolygonPath *pp;

/* allocate and arrange memory */
for (pp= poly; pp != NULL; pp= pp->next)
for (p= pp->p; p != NULL; p= p->next)
totalVert++;

if (memoryAllocated < totalVert) {
/* free old memory */
if (mem1 != NULL) {
free(mem1);
free(mem2);
}

/* get new memory */
if (( mem1 = malloc(totalVert * sizeof(List))) == NULL ||
(mem2 = malloc(totalVert * sizeof(Edge))) == NULL) {
(void) fprintf(stderr, "Sorry, couldn't allocate memory\n");
exit(1);
}
memoryAllocated = totalVert;
}

list = (List * ) mem1;
edges = (Edge * ) mem2;

for (i = 0; i < totalVert; i++) {
list[i].next = &list[i+1];
list[i].edge = &edges[i];
}
```

Σχήμα 2.10: Συνεχίζεται

```
/* initialize edges */
for (pp= poly; pp != NULL; pp= pp->next)
for (p= pp->p; p != NULL; p= p->next) {
v1 = &p->v;
if (p->next != NULL)
v2 = &p->next->v;
else v2 = &pp->p->v;

/*
We're throwing out edges which don't intersect at
least one scan line. This includes horizontal edges.
*/
if (ceil(v1->y) != ceil(v2->y)) {
if (v1->y < v2->y) {
edges[numEdges].direction = UP;
dx = (v2->x - v1->x) / (v2->y - v1->y);
edges[numEdges].dx = dx;
edges[numEdges].yupper = v2->y;
ymin = ceil(v1->y);
edges[numEdges].ymin = ymin;
edges[numEdges].xcurr = v1->x + dx *
(ymin - v1->y);
numEdges++;
} else if (v1->y > v2->y) {
edges[numEdges].direction = DOWN;
dx = (v2->x - v1->x) / (v2->y - v1->y);
edges[numEdges].dx = dx;
edges[numEdges].yupper = v1->y;
ymin = ceil(v2->y);
edges[numEdges].ymin = ymin;
edges[numEdges].xcurr = v2->x + dx *
(ymin - v2->y);
numEdges++;
}
}
}

/* check if polygon remains */
if (numEdges == 0)
return NULL;
```

Σχήμα 2.10: Συνεχίζεται

```
/* make sure link list stops at last element */
list[numEdges-1].next = NULL;

SortEdges(ON_Y, list);
return list;
}


static List *
UpdateEdges(newY, list)
int newY;
List *list;
{
List *next, *newList;

newList = NULL;
while (list != NULL) {
next = list->next;
if (list->edge->yupper > (double) newY) {
/* add to new list */
list->next = newList;
newList = list;
newList->edge->xcurr += newList->edge->dx;
}
list = next;
}

return newList;
}


void
ScanConvertPolyPath(poly)
PolygonPath *poly;
{
register List *activeList, *edgeList, *temp;
register List *left, *edge;
register int ycurr, wrap;

edgeList = AddEdgesToList(poly);
```

Σχήμα 2.10: Συνεχίζεται

```
activeList = NULL;

while (activeList != NULL || edgeList != NULL) {
/* get low y */
if (activeList == NULL)
ycurr = edgeList->edge->ymin;

/* add edges to active list */
while (edgeList != NULL && edgeList->edge->ymin == ycurr) {
temp = activeList;
activeList = edgeList;
edgeList = edgeList->next;
activeList->next = temp;
}

SortEdges(ON_X, activeList);

/* generate regions to be filled in on current scan line */
wrap = 0;
left = activeList;
for (edge = activeList; edge != NULL; edge = edge->next) {
wrap += edge->edge->direction;
if (wrap == 0) {
DrawSpan(ycurr, left->edge->xcurr,
edge->edge->xcurr);
left = edge->next;
}
}
ycurr++;

/* update edges in active list */
activeList = UpdateEdges(ycurr, activeList);
}
}
```

Σχήμα 2.10 Αρχείο poly.c: Αλγόριθμος σάρωσης & μετατροπής πολυγώνου

```
/*
 * John Amanatides, Jan 1995
 */

#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "graf2d.h"
#include "simpleX.h"

extern PolygonPath *ClipPolyPath();
extern void ScanConvertPolyPath();

#define TRUE        1
#define FALSE       0
#define WINDOW_RES  512

static int doPath;
static PolygonPath *currentPath;
static PolygonPath *characterMaster[256];
static Point currentPoint;
static Matrix identity= {   1.0, 0.0, 0.0,
                0.0, 1.0, 0.0,
                0.0, 0.0, 1.0};
static Matrix CTM, NDCtoScreen, modelToScreen;
static StackNode *CTMstack;
static ClippingRegion clip;

/*
 * result= m1*m2
 */
static Matrix
MultMatrix(m1, m2)
Matrix *m1, *m2;
```

Σχήμα 2.11: Συνεχίζεται

```
{
Matrix result;
int r, c, i;

for(r=0; r < 3; r++)
for(c=0; c < 3; c++) {
result.m[r][c]= 0.0;
for(i=0; i < 3; i++)
result.m[r][c] += m1->m[r][i]*m2->m[i][c];
}

return result;
}

/*
 * result= matrix*p
 */
static Point
TransformPoint(p, matrix)
Point p;
Matrix *matrix;
{
Point result;

result.x= matrix->m[0][0]*p.x + matrix->m[0][1]*p.y +
               matrix->m[0][2];
result.y= matrix->m[1][0]*p.x + matrix->m[1][1]*p.y +
               matrix->m[1][2];

return result;
}

static void ResetMapping()
{
modelToScreen= MultMatrix(&NDCtoScreen, &CTM);
}
```

Σχήμα 2.11: Συνεχίζεται

```
void
InitTM()
{
CTM= identity;
ResetMapping();
}

/*
 * defined in NDC
 */
void
SetClipRegion(minX, minY, maxX, maxY)
double minX, minY, maxX, maxY;
{
if (minX  < 0.0 || maxX > 1.0 || minX >= maxX ||
    minY  < 0.0 || maxY > 1.0 || minY >= maxY) {
(void) fprintf(stderr, "clipping region is incorrect\n");
return;
}
clip.minX= minX*WINDOW_RES;
clip.maxX= maxX*WINDOW_RES;
clip.minY= (1.0-maxY)*WINDOW_RES;
clip.maxY= (1.0-minY)*WINDOW_RES;
}

void
GrafStart(name)
char *name;
{
OpenX(name, WINDOW_RES, WINDOW_RES);

CTM= identity;
CTMstack= NULL;

NDCtoScreen= identity;
NDCtoScreen.m[0][0]=  WINDOW_RES;
NDCtoScreen.m[1][1]= -WINDOW_RES;
NDCtoScreen.m[1][2]=  WINDOW_RES;
```

Σχήμα 2.11: Συνεχίζεται

```
ResetMapping();

SetClipRegion(0.0, 0.0, 1.0, 1.0);
}

void
GrafEnd()
{
CloseX();
}

/*
 * called by scan-converter in poly.c
 */
void
DrawSpan(y, x1, x2)
int y;
double x1, x2;
{
DrawLineX((int) x1, y, (int) x2, y);
}

void
Clear()
{
ClearX();
}

/*
 * for debugging purposes
 */
Print(pp)
PolygonPath *pp;
{
Polygon *p;

while(pp) {
(void) fprintf(stderr, "polygon\n");
for(p= pp->p; p != NULL; p= p->next)
(void) fprintf(stderr, "\t%g %g\n", p->v.x, p->v.y);
pp= pp->next;
}
}
```

Σχήμα 2.11: Συνεχίζεται

```
/*
 * internal function to free PolygonPath
 */
static void
FreePolyPath(pp)
PolygonPath **pp;
{
PolygonPath *tempPP;
Polygon *p;

while(*pp) {
tempPP= *pp;
while(tempPP->p) {
p= tempPP->p;
tempPP->p= tempPP->p->next;
free(p);
}
*pp= (*pp)->next;
free(tempPP);
}
}

/*
 *  transform polygonPath from model to screenCoords
 */

static void
TransformPolyPath(pp)
PolygonPath *pp;
{
Polygon *p;

while(pp) {
for(p= pp->p; p != NULL; p= p->next)
p->v= TransformPoint(p->v, &modelToScreen);
pp= pp->next;
}
}
static Polygon *
DuplicatePoly(origPoly)
Polygon *origPoly;
```

Σχήμα 2.11: Συνεχίζεται

```
{
Polygon *newPoly, **currentNode;

currentNode= &newPoly;
while(origPoly) {
*currentNode= (Polygon *) malloc(sizeof(Polygon));
(*currentNode)->v= origPoly->v;
origPoly= origPoly->next;
currentNode= &(*currentNode)->next;
}
*currentNode= NULL;
return newPoly;
}


static PolygonPath *
DuplicatePolyPath(origPP)
PolygonPath *origPP;
{
PolygonPath *newPP, **currentNode;

currentNode= &newPP;
while(origPP) {
*currentNode= (PolygonPath *) malloc(sizeof(PolygonPath));
(*currentNode)->p= DuplicatePoly(origPP->p);
origPP= origPP->next;
currentNode= &(*currentNode)->next;
}
*currentNode= NULL;
return newPP;
}

void
PushTM()
{
StackNode *sp;

sp= (StackNode *) malloc(sizeof(StackNode));
sp->CTM= CTM;
sp->next= CTMstack;
CTMstack= sp;
}
```

Σχήμα 2.11: Συνεχίζεται

```
void
PopTM()
{
StackNode *sp;

if(CTMstack != NULL) {
CTM= CTMstack->CTM;
sp= CTMstack;
CTMstack= CTMstack->next;
free(sp);
}
else
(void) fprintf(stderr, "stack underflow\n");
ResetMapping();
}


void DefineChar(c)
unsigned char c;
{
FreePolyPath(&characterMaster[c]); /* just in case */
characterMaster[c]= currentPath;
currentPath= NULL;
doPath= FALSE;
}


void Newpath()
{
/* your code goes here */
}


void Fill()
{
/* your code goes here */
}
```

Σχήμα 2.11: Συνεχίζεται

```
void
Scale(sx, sy)
double sx, sy;
{
/* your code goes here */
}

void
Rotate(angle)
double angle;
{
/* your code goes here */
}

void
Translate(tx, ty)
double tx, ty;
{
/* your code goes here */
}

void
Shear(sx, sy)
double sx, sy;
{
/* your code goes here */
}

void
Moveto(x, y)
double x, y;
{
PolygonPath *pp;

currentPoint.x = x;
currentPoint.y = y;

if(doPath) {
pp= (PolygonPath *) malloc(sizeof(PolygonPath));
pp->next= currentPath;
currentPath= pp;
```

Σχήμα 2.11: Συνεχίζεται

```
currentPath->p= (Polygon *) malloc(sizeof(Polygon));
currentPath->p->v= currentPoint;
currentPath->p->next= NULL;
} else {
(void) fprintf(stderr, "you forgot to call newpath\n");
}
}

void
Move(x, y)
double x, y;
{
/* your code goes here */
}

void
Lineto(x, y)
double x, y;
{
/* your code goes here */
}

void
Line(x, y)
double x, y;
{
/* your code goes here */
}

void
Text(string)
char *string;
{
/* your code goes here */
}
```

Σχήμα 2.11 Αρχείο graf2d.c: Σκελετός του κώδικα που ζητείται να συμπληρωθεί

```
#include <stddef.h>
#include "graf2d.h"


static Polygon *
ClipPolygon(p, clip)
Polygon *p;
ClippingRegion *clip;
{
/* your code goes here */

return p;
}

/*
 * clip polygon path;  to keep it simple, PolygonPath nodes that are empty
 * are retained.
 */

PolygonPath *
ClipPolyPath(path, clip)
PolygonPath *path;
ClippingRegion *clip;
{
PolygonPath *current;

for(current= path; current != NULL; current= current->next)
current->p= ClipPolygon(current->p, clip);
return path;
}
```

Σχήμα 2.12 Σκελετός κώδικα clip.c

2.3 3η Ασκηση: Graf3d, ένα απλό πακέτο τρισδιάστατων γραφικών

Στην άσκηση αυτή θα εργαστείτε πάνω στους μετασχηματισμούς και την οπτική αντίληψη (viewing) στις τρείς διαστάσεις. Θα υλοποιήσετε ένα πολύ απλό πακέτο τρισδιάστατων γραφικών **graf3d** που ασχολείται με πολύγωνα.

Σας δίνεται ένα σημαντικό μέρος του κώδικα. Συγκεκριμένα:

1. ο διερμηνέας (interpreter) του πακέτου **graf3d** στο αρχείο `/apps/CG/a3/scanner.c` (Σχήμα 2.18). Οι ορισμοί σταθερών και συναρτήσεων που εμφανίζονται στο αρχείο αυτό παρατίθενται στο αρχείο `/apps/CG/a3/scanner.h`:

```
/* commands */
#define ERROR       0
#define QUIT        1
#define COMMENT     2
#define INCLUDE     3
#define CLEAR       4
#define CAMERA      5
#define PERSPECTIVE 6
#define VIEWPORT    7
#define SCALE       8
#define ROTATE      9
#define TRANSLATE   10
#define SHEAR       11
#define INIT_TM     12
#define PUSH_TM     13
#define POP_TM      14
#define POLYGON     15

typedef struct {
    char *word;
    int code;
} Words;

static Words Commands[]= {
    "quit", QUIT,
    "%", COMMENT,
    "include", INCLUDE,
```

Σχήμα 2.13: Συνεχίζεται

```
    "clear", CLEAR,
    "camera", CAMERA,
    "perspective", PERSPECTIVE,
    "viewport", VIEWPORT,
    "scale", SCALE,
    "rotate", ROTATE,
    "translate", TRANSLATE,
    "shear", SHEAR,
    "initTM", INIT_TM,
    "pushTM", PUSH_TM,
    "popTM", POP_TM,
    "polygon", POLYGON,
    NULL, ERROR
};
```

Σχήμα 2.13 Αρχείο scanner.h: Ορισμοί σταθερών και συναρτήσεων για τον διερμηνέα του πακέτου **graf3d**

2. ο σκελετός του κώδικα για το πακέτο **graf3d** στο αρχείο `/apps/CG/a3/graf3d.c` (Σχήμα 2.19). Οι ορισμοί των συναρτήσεων δίδονται στο αρχείο `/apps/CG/a3/graf3d.h` (Σχήμα 2.14).

```
#define G3D_MAX_VERT_IN_POLY        (16)

typedef struct {
    double x, y, z;
} Point, Vector;

extern char *G3D_Start();
extern char *G3D_End();
extern char *G3D_Clear();
extern char *G3D_Camera();
extern char *G3D_Perspective();
```

Σχήμα 2.14: Συνεχίζεται

```
extern char *G3D_Viewport();
extern char *G3D_Scale();
extern char *G3D_Rotate();
extern char *G3D_Translate();
extern char *G3D_Shear();
extern char *G3D_InitTM();
extern char *G3D_PushTM();
extern char *G3D_PopTM();
extern char *G3D_Polygon();
```

Σχήμα 2.14 Αρχείο graf3d.h: Ορισμοί συναρτήσεων τρισδιάστατων γραφικών

3. Οι βοηθητικές ρουτίνες για τα X-Windows, όπως στην Ασκηση 1 (αρχεία `/apps/CG/a1/simpleX.h`, `/apps/CG/a1/simpleX.c`).

Ζητείται να υλοποιηθούν οι συναρτήσεις γραφικών που δίνονται παρακάτω ως εντολές του διερμηνέα για το πακέτο **graf3d**. (Σημειώστε ότι ο διερμηνέας του πακέτου "διαβάζει" μόνο μια εντολή ανά γραμμή εισόδου.)

`scale` $sx\ sy\ sz$
`rotate` $axis\ angle$
`translate` $tx\ ty\ tz$
`shear` $axis1\ axis2\ amount$
`initTM`
`pushTM`
`popTM`

`camera` $from.x\ from.y\ from.z\ at.x\ at.y\ at.z\ [up.x\ up.y\ up.z]$
`perspective` $width\ height\ d\ [near\ far]$
`viewpoint` $llx\ lly\ urx\ ury$

`polygon` $x1\ y1\ z1\ x2\ y2\ z2\ \ldots\ xn\ yn\ zn$ `clear`

`include` $filename$
`%` $any\ string$
`quit`

Οι πρώτες επτά εντολές ασχολούνται με τον τρέχοντα πίνακα μετασχηματισμού (current transformation matrix - CTM). Ολα τα αντικείμενα προτού σχεδιαστούν μετασχηματίζονται από τον πίνακα αυτό. Σημειώνεται ότι η γωνία περιστροφής ορίζεται σε μοίρες, ανθωρολογιακά και ότι παράμετροι των αξόνων είναι x, y και z.

Η εντολή `camera` υποδηλώνει τη θέση και τον προσανατολισμό της εικονικής (virtual) κάμερας. Οι εξ ορισμού τιμές των παραμέτρων είναι `(0.0, -10.0,`

`0.0) (0.0, 0.0, 0.0) (0.0, 0.0, 1.0)`. Η εντολή `perspective` περιγράφει την προβολή προοπτικής (perspective projection) και προσδιορίζει το κοντινό (near) και μακρινό (far) επίπεδο ψαλιδίσματος. Οι εξ ορισμού τιμές είναι `1, 1, 1, 0.1, 1000.0`. Η εντολή `viewpoint` προσδιορίζει πού θα τοποθετηθεί επί της οθόνης το ορατό τμήμα του επιπέδου. Οι συντεταγμένες δίνονται στο σύστημα N.D.C. (normalized device coordinates), δηλαδή μεταξύ `(0.0, 0.0)` και `(1.0,1.0)`. Σημειώστε επίσης ότι γι' αυτή την εντολή ως αρχή εκλαμβάνεται η κάτω αριστερή γωνία.

Η εντολή `polygon` περιέχει τις κορυφές του πολυγώνου που θα σχεδιαστεί.

Η εντολή `include` επιτρέπει την ενσωμάτωση και ερμηνεία εντολών από αρχείο. Μπορεί να χρησιμοποιηθεί για να εισαχθεί το μοντέλο ενός σπιτιού, όπως δίνεται στο Σχήμα 2.15, το οποίο ακολούθως να επαναληφθεί αρκετές φορές. Η εντολή `%`, είναι εντολή που επιτρέπει την εισαγωγή σχολίων στα αρχεία εισόδου. Τέλος, η εντολή `quit` σταματά την εκτέλεση του προγράμματος. Ενα απλό παρά-

```
% house of width 2 and length 4 at origin; door on south side
%
pushTM
translate 0 0 1
polygon 1 -2 -1 1 -2 1 0 -2 2 -1 -2 1 -1 -2 -1
polygon -1 2 -1 -1 2 1 0 2 2 1 2 1 1 2 -1
polygon 1 2 -1 1 2 1 1 -2 1 1 -2 -1
polygon -1 -2 -1 -1 -2 1 -1 2 1 -1 2 -1
polygon 1 2 1 0 2 2 0 -2 2 1 -2 1
polygon -1 -2 1 0 -2 2 0 2 2 -1 2 1
polygon -.25 -2 -1 -.25 -2 .5 .2
5 -2 .5 .25 -2 -1
polygon -1 -2 -1  1 -2 -1 1 2 -1 -1 2 -1
popTM
```

Σχήμα 2.15 Περιγραφή ενός σπιτιού στο πακέτο **graf3d**

δειγμα που μπορείτε να επεξεργαστείτε με το πακέτο **graf3d** είναι το εξής:

Για τη διευκόλυνσή σας παρατίθεται και το `/apps/CG/a3/Makefile` που παράγει το εκτελέσιμο πρόγραμμα στο Σχήμα 2.17.

Στα παραδοτέα της άσκησης πέρα από μια εκτύπωση εικόνας εξόδου του πακέτου να συμπεριλάβετε και ένα σύντομο, αλλά περιεκτικό εγχειρίδιο χρήστη του πακέτου **graf3d**. Ο δοσμένος πλήρης κώδικας για τον διερμηνέα του πακέτου παρατίθεται στο Σχήμα 2.18, ενώ ο πυρήνας του κώδικα του πακέτου που πρέπει να υλοποιήσετε περιγράφεται στο Σχήμα 2.19.

```
camera 2 -5 3 0 0 0
perspective 1 1 1
% draw an F on the XZ plane
polygon 0 0 0  0 0 1  0.5 0 1  0.5 0 0.9  0.1 0 0.9  0.1 0 0.6  0.5 0 0.6
    0.5 0 0.5  0.1 0 0.5  0.1 0 0
rotate x -90
% draw an F on the XY plane
polygon 0 0 0  0 0 1  0.5 0 1  0.5 0 0.9  0.1 0 0.9  0.1 0 0.6  0.5 0 0.6
    0.5 0 0.5  0.1 0 0.5  0.1 0 0
```

Σχήμα 2.16 Απλό παράδειγμα εφαρμογής για επεξεργασία με το πακέτο **graf3d**

```
CFLAGS= -g
OBJ= scanner.o graf3d.o simpleX.o
SRC= scanner.c graf3d.c simpleX.c
LIB= -lX11 -lm

graf3d: $(OBJ)
    cc -o graf3d $(CFLAGS) $(OBJ) $(LIB)

clean:
    rm -f graf3d $(OBJ) a3.ps
lint:
    lint -u $(SRC)

graf3d.o poly.o clip.o: graf3d.h
scanner.o: scanner.h
simpleX.o graf3d.o: simpleX.h
```

Σχήμα 2.17 Makefile που παράγει το εκτελέσιμο πρόγραμμα **graf3d**

```
/*
 * front end for 3d graphics program; scan input and call appropriate functions
 *
 * written by John Amanatides, copyright (c) 1987, 1994
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "scanner.h"
#include "graf3d.h"


#define MAX_INCLUDE_DEPTH 8
#define MAX_LINE_LENGTH 1024
#define OUTPUT(string) if(input == stdin) ((void) printf(string))

#define DEFAULT_NEAR_DISTANCE 0.1
#define DEFAULT_FAR_DISTANCE 1000.0
/*
 * decode first word of string
 */
static int
DecodeWord(list, string)
Words *list;
char *string;
{
int i, result;
char command[MAX_LINE_LENGTH];

if(string == NULL || *string == '\n')
return COMMENT;
(void) sscanf(string, "%s", command);

result= ERROR;
for(i=0;list[i].word != NULL && result == ERROR; i++) {
if(strcmp(command,list[i].word) == 0)
result= list[i].code;
}
return result;
}
```

Σχήμα 2.18: Συνεχίζεται

```
/*
 * read in polygon verticies from input; assumes input is correct
 */
void ReadPolygon(inputString, vertex, numVertex)
char *inputString;
Point vertex[];
int *numVertex;
{
double x, y, z;
char *x_string, *y_string, *z_string;

*numVertex= 0;
(void) strtok(inputString, " "); /* skip over string 'polygon' */

x_string= strtok(NULL, " ");
y_string= strtok(NULL, " ");
z_string= strtok(NULL, " ");
while ((x_string != NULL) && (y_string != NULL) && (z_string != NULL)){
x= atof(x_string);
y= atof(y_string);
z= atof(z_string);
vertex[*numVertex].x = x;
vertex[*numVertex].y = y;
vertex[*numVertex].z = z;
(*numVertex)++;
x_string= strtok(NULL, " ");
y_string= strtok(NULL, " ");
z_string= strtok(NULL, " ");
}
}

/*
 * perform scanning of input
 */
static void
Dialogue(input, indirectDepth)
FILE *input;
int indirectDepth;
{
FILE *newFile;
int numVertex;
char axis1, axis2, *result;
char line[MAX_LINE_LENGTH], stringArg[MAX_LINE_LENGTH];
double x, y, z, x2, y2, near, far;
```

Σχήμα 2.18: Συνεχίζεται

```
Point vertex[G3D_MAX_VERT_IN_POLY];
Point from, at;
Vector up;

while(1) {
result = NULL;
OUTPUT("? ");
if(fgets(line, MAX_LINE_LENGTH, input) == NULL)
return; /* End of File */

switch(DecodeWord(Commands, line)) {

case QUIT:
return;
case COMMENT:
break;
case INCLUDE:
if(indirectDepth >= MAX_INCLUDE_DEPTH) {
(void) printf("Sorry, the depth of indirection is too great.\n");
break;
}
(void) sscanf(line,"%*s %s", stringArg);
if((newFile= fopen(stringArg,"r")) == NULL)
(void) printf("Sorry, cannot open file '%s' for input.\n",stringArg);
else {
Dialogue(newFile, indirectDepth+1);
(void) fclose(newFile);
}
break;
case CLEAR:
result= G3D_Clear();
break;
case CAMERA:
up.x= 0.0;
up.y= 0.0;
up.z= 1.0;
 (void) sscanf(line, "%*s %lf %lf %lf %lf %lf %lf %lf %lf %lf",
&from.x, &from.y, &from.z, &at.x, &at.y, &at.z,
&up.x, &up.y, &up.z);
result= G3D_Camera(from, at, up);
break;
```

Σχήμα 2.18: Συνεχίζεται

```
case PERSPECTIVE:
near= DEFAULT_NEAR_DISTANCE;
far= DEFAULT_FAR_DISTANCE;
(void) sscanf(line, "%*s %lf %lf %lf %lf %lf",
&x, &y, &z, &near, &far);
result= G3D_Perspective(x, y, z, near, far);
break;
case VIEWPORT:
(void) sscanf(line, "%*s %lf %lf %lf %lf",
&x, &y, &x2, &y2);
result= G3D_Viewport(x, y, x2, y2);
break;
case SCALE:
(void) sscanf(line, "%*s %lf %lf %lf", &x, &y, &z);
result= G3D_Scale(x, y, z);
break;
case ROTATE:
(void) sscanf(line, "%*s %c %lf", &axis1, &x);
result= G3D_Rotate(axis1, x);
break;
case TRANSLATE:
(void) sscanf(line, "%*s %lf %lf %lf", &x, &y, &z);
result= G3D_Translate(x, y, z);
break;
case SHEAR:
(void) sscanf(line, "%*s %c %c %lf",&axis1, &axis2, &x);
result= G3D_Shear(axis1, axis2, x);
break;
case INIT_TM:
result= G3D_InitTM();
break;
case PUSH_TM:
result= G3D_PushTM();
break;
case POP_TM:
result= G3D_PopTM();
break;
```

Σχήμα 2.18: Συνεχίζεται

```
case POLYGON:
ReadPolygon(line, vertex, &numVertex);
result= G3D_Polygon(vertex, numVertex);
break;
case ERROR:
default:
(void) printf("Eh?\n");
break;
}
if (result != NULL)
(void) fprintf(stderr, "Error encountered in G3D command: %s\n",
result);
}
}


main(argc, argv)
int argc;
char *argv[];
{
(void) G3D_Start(argv[0]);
Dialogue(stdin, 1);
(void) G3D_End();
exit(0);
}
```

Σχήμα 2.18 Αρχείο scanner.c: Διερμηνέας εντολών του πακέτου **graf3d**

```
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "graf3d.h"
#include "simpleX.h"

#define WINDOW_RES 512

#define DOT(a,b) ((a).x*(b).x + (a).y*(b).y + (a).z*(b).z)

typedef struct {
double m[4][4];
} Matrix;

typedef struct StackNode {
Matrix CTM;
struct StackNode *next;
} StackNode;

static Matrix identity= { 1.0, 0.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 1.0};

static Matrix CTM;
static StackNode *CTMstack;

/*
 * result= axb
 */
static Vector
Cross(a,b)
Vector a, b;
{
Vector result;

result.x= a.y*b.z - a.z*b.y;
result.y= a.z*b.x - a.x*b.z;
result.z= a.x*b.y - a.y*b.x;

return result;
}
```

Σχήμα 2.19: Συνεχίζεται

```
/*
 *
This function normalizes the vector and returns the original length
 */
static double
Normalize(a)
Vector *a;
{
double length, scale;

length= DOT(*a,*a);
length= sqrt(length);
if(length != 0.0) {
scale= 1.0/length;
a->x *= scale;
a->y *= scale;
a->z *= scale;
}
return length;
}


/*
 * result= m1*m2
 */

static Matrix
MultMatrix(m1, m2)
Matrix *m1, *m2;
{
Matrix result;
int r, c, i;

for(r=0; r < 4; r++)
for(c=0; c < 4; c++) {
result.m[r][c]= 0.0;
for(i=0; i < 4; i++)
result.m[r][c] += m1->m[r][i]*m2->m[i][c];
}

return result;
}
```

Σχήμα 2.19: Συνεχίζεται

```
/*
 * result= matrix*p
 */
static Point
TransformPoint(p, matrix)
Point p;
Matrix *matrix;
{
Point result;

result.x= matrix->m[0][0]*p.x + matrix->m[0][1]*p.y +
matrix->m[0][2]*p.z + matrix->m[0][3];
result.y= matrix->m[1][0]*p.x + matrix->m[1][1]*p.y +
matrix->m[1][2]*p.z + matrix->m[1][3];
result.z= matrix->m[2][0]*p.x + matrix->m[2][1]*p.y +
matrix->m[2][2]*p.z + matrix->m[2][3];

return result;
}

/*
 * This routine must be called first with the name of the window
 * to be created
 */
char *
G3D_Start(name)
char *name;
{
OpenX(name, WINDOW_RES, WINDOW_RES);

CTM= identity;
CTMstack= NULL;

/* your code goes here */

return NULL;
}
```

Σχήμα 2.19: Συνεχίζεται

```
/*
 * This is called last
 */
char *
G3D_End()
{
CloseX();
return NULL;
}

/*
 * Clear the screen
 */
char *
G3D_Clear()
{
ClearX();
return NULL;
}

/*
 * Indicate the position and orientation of the camera
 */
char *
G3D_Camera(from, to, up)
Point from, to;
Vector up;
{
/* your code goes here */
return NULL;
}

/*
 * Indicate the position and orientation of the camera
 */
char *
G3D_Perspective(width, height, distance, near, far)
double width, height, distance, near, far;
```

Σχήμα 2.19: Συνεχίζεται

```
{
/* your code goes here */
return NULL;
}
/*
 * Indicate where on the screen (in NDC) the objects will be viewed
 */
char *
G3D_Viewport(llx, lly, urx, ury)
double llx, lly, urx, ury;
{
/* your code goes here */
return NULL;
}
/*
 * Initialize the CTM
 */
char *
G3D_InitTM()
{
CTM= identity;

/* your code goes here */
return NULL;
}
/*
 * push the current value of the CTM onto the CTM stack
 */
char *
G3D_PushTM()
{
StackNode *sp;

sp= (StackNode *) malloc(sizeof(StackNode));
sp->CTM= CTM;
sp->next= CTMstack;
CTMstack= sp;

return NULL;
}
```

Σχήμα 2.19: Συνεχίζεται

```
/*
 * pop from the CTM stack into the CTM
 */
char *
G3D_PopTM()
{
StackNode *sp;

if(CTMstack != NULL) {
CTM= CTMstack->CTM;
sp= CTMstack;
CTMstack= CTMstack->next;
free((void *) sp);
}
else
return "stack underflow";

/* your code goes here */

return NULL;
}
/*
 * premultiply CTM by following transformation
 */
char *
G3D_Scale(sx, sy, sz)
double sx, sy, sz;
{
/* your code goes here */
return NULL;
}
/*
 * premultiply CTM by following transformation
 */
char *
G3D_Rotate(axis, angle)
char axis;
double angle;
{
/* your code goes here */
return NULL;
}
```

Σχήμα 2.19: Συνεχίζεται

```
/*
 * premultiply CTM by following transformation
 */
char *
G3D_Translate(tx, ty, tz)
double tx, ty, tz;
{
/* your code goes here */
return NULL;
}
/*
 * premultiply CTM by following transformation
 */
char *
G3D_Shear(axis1, axis2, amount)
char axis1, axis2;
double amount;
{
/* your code goes here */
return NULL;
}
/*
 *
Draw this polygon, after tranforming by CTM, clipping, perspective
 * and doing transformation to screen.
 */
char *
G3D_Polygon(vertex, numVertex)
Point vertex[];
int numVertex;
{
/* your code goes here */
return NULL;
}
```

Σχήμα 2.19 Αρχείο graf3d.c: Πυρήνας κώδικα του πακέτου **graf3d**

2.4 4η Ασκηση: Φωτισμός και σκίαση αντικειμένων

Σκοπός της άσκησης αυτής είναι να σας εισαγάγει στους υπολογισμούς φωτισμού (ray casting) και σκίασης (shading). Ζητείται να συμπληρώσετε τον κώδικα ενός πολύ απλού φωταγωγού (ray caster) που λέγεται **rt**. Για να κατανοήσετε το αντικείμενο εργασίας σας δίνεται έτοιμη μια πρωτόγονη έκδοση ενός τέτοιου φωταγωγού στο αρχείο `/apps/CG/a4/example_rt.c` (Σχήμα 2.28).

Ενα πολύ μεγάλο μέρος του κώδικα σας δίνεται έτοιμο. Ειδικότερα:

1. ο διερμηνέας (interpreter) του πακέτου **rt** στο αρχείο `/apps/CG/a4/scanner.c` (Σχήμα 2.25). Οι ορισμοί σταθερών και συναρτήσεων που εμφανίζονται στο αρχείο αυτό παρατίθενται στο αρχείο `/apps/CG/a4/scanner.h`:

```
/* commands */
#define ERROR 0
#define QUIT 1
#define INCLUDE 2
#define COMMENT 3
#define SCALE 4
#define ROTATE 5
#define TRANSLATE 6
#define SHEAR 7
#define INIT_TM 8
#define PUSH_TM 9
#define POP_TM 10
#define CAMERA 11
#define PERSPECTIVE 12
#define TRACE 13
#define LIGHT 14
#define SURFACE 15
#define SPHERE 16
#define PLANE 17
#define CUBE 18
#define INIT_OBJECTS 19
```

Σχήμα 2.20: Συνεχίζεται

```
typedef struct {
char *word;
int code;
} Words;
static Words Commands[]= {
"quit", QUIT,
"include", INCLUDE,
"%", COMMENT,
"scale", SCALE,
"rotate", ROTATE,
"translate", TRANSLATE,
"shear", SHEAR,
"initTM", INIT_TM,
"pushTM", PUSH_TM,
"popTM", POP_TM,
"camera", CAMERA,
"perspective", PERSPECTIVE,
"light", LIGHT,
"surface", SURFACE,
"sphere", SPHERE,
"plane", PLANE,
"cube", CUBE,
"initObjects", INIT_OBJECTS,
"trace", TRACE,
NULL, ERROR
};
```

Σχήμα 2.20 Αρχείο scanner.h: Ορισμοί σταθερών και συναρτήσεων για τον διερμηνέα του πακέτου **rt**

2. Συλλογή διάφορων βοηθητικών βασικών συναρτήσεων για το πακέτο **rt** στο αρχείο `/apps/CG/a4/rt_Misc.c` (Σχήμα 2.26). Οι ορισμοί των βοηθητικών συναρτήσεων δίδονται στο αρχείο `/apps/CG/a4/ rt_Internal.h` (Σχήμα 2.21).

```
#include "rt.h"

#define EPSILON (1.0e-5)
#define HIT     1
#define MISS 0
```

Σχήμα 2.21: Συνεχίζεται

```
typedef struct {
Point origin;
Vector direction;
} Ray;

typedef struct {
double m[4][4];
} Matrix;

typedef struct {
Matrix TM;
Matrix inverseTM;
} Affine;

typedef struct StackNode {
Affine CTM;
struct StackNode *next;
} StackNode;

typedef struct ListNode {
int primitive;
Affine TM;
Surface surf;
struct ListNode *next;
} ListNode;

#define DOT(a,b) ((a).v[0]*(b).v[0] + (a).v[1]*(b).v[1] + (a).v[2]*(b).v[2])
#define PLUS(r,a,b) { \
(r).v[0] = (a).v[0] + (b).v[0]; \
(r).v[1] = (a).v[1] + (b).v[1]; \
(r).v[2] = (a).v[2] + (b).v[2]; \
}
#define MINUS(r,a,b) { \
(r).v[0] = (a).v[0] - (b).v[0]; \
(r).v[1] = (a).v[1] - (b).v[1]; \
(r).v[2] = (a).v[2] - (b).v[2]; \
}
#define TIMES(r,a,s) { \
(r).v[0] = (s)*(a).v[0]; \
(r).v[1] = (s)*(a).v[1]; \
(r).v[2] = (s)*(a).v[2]; \
}
```

Σχήμα 2.21 Αρχείο rtInternal.h: Ορισμοί βοηθητικών συναρτήσεων

3. Ο Υπολογισμός της τομής μεταξύ ακτίνας και σφαίρας με μοναδιαία ακτίνα και κέντρο την αρχή του συστήματος συντεταγμένων δίνεται στο αρχείο `/apps/CG/a4/sphere.c` (Σχήμα 2.27)

Ο διερμηνέας του πακέτου **rt** κατανοεί τις ακόλουθες εντολές: (Σημειώστε ότι ο διερμηνέας του πακέτου "διαβάζει" μόνο μια εντολή ανά γραμμή εισόδου.)

`scale` $sx\ sy\ sz$
`rotate` $axis\ angle$
`translate` $tx\ ty\ tz$
`shear` $axis1\ axis2\ amount$
`initTM`
`pushTM`
`popTM`

`camera` $from.x\ from.y\ from.z\ at.x\ at.y\ at.z\ [up.x\ up.y\ up.z]$
`perspective` $distance\ width\ height$

`sphere`
`plane`
`cube`
`initObjects`

`light` $x\ y\ z$
`surface` $ambient\ diffuse\ specular\ n$

`trace` $x\ y\ filename$
`include` $filename$
`quit`

Οι πρώτες επτά εντολές ασχολούνται με τον τρέχοντα πίνακα μετασχηματισμού (current transformation matrix - CTM). Οι άξονες περιστροφής και διατμησης (shear) μπορεί να είναι 0, 1 ή 2 υποδηλώνοντας τους $x-, y-, z-$ άξονες αντιστοίχως. Στην εντολή διάτμησης μπορείτε να κόβετε κατά μήκος του άξονα $axis1$ ως συνάρτηση του άξονα $axis2$.

Οι επόμενες δύο εντολές ορίζουν τον όγκο που βλέπουμε (view volume). Η εντολή `camera` προσδιορίζει τη θέση και τον προσανατολισμό της κάμερας. Το μοναδιαίο κάθετο προς τα πάνω διάνυσμα είναι το $(0, 0, 1)$. Η εντολή `perspective` επιτρέπει να προσδιορίσετε το τμήμα του επιπέδου προβολής που εμφανίζεται στην οθόνη.

Υπάρχουν τρία αρχέτυπα (primitives): `sphere` (μοναδιαία σφαίρα), `plane` (το επίπεδο $z = 0$) και `cube` ($-1 \leq x, y, z \leq 1$). Κάθε φορά που ορίζεται ένα αρχέτυπο, τοποθετείται σε ένα διασυνδεδεμένο κατάλογο (linked list) αρχετύπων, όπου κάθε κόμβος του καταλόγου αυτού περιέχει μια συναφή περιγραφή και μια περιγραφή επιφανείας. Ο διασυνδεδεμένος κατάλογος αποθηκεύει τα αντικείμενα της

σκηνής. Η εντολή `initObjects` διαγράφει όλα τα αποσύρει όλα τα αντικείμενα της σκηνής.

Η εντολή `light` τοποθετεί μια φωτεινή πηγή με τον προσανατολισμό που προσδιορίζεται. Η εντολή `surface` τροποποιεί την τρέχουσα περιγραφή επιφάνειας. Κάθε φορά που προστίθεται ένα αρχέτυπο, κληρονομεί την τρέχουσα περιγραφή επιφάνειας.

Η εντολή `trace` δημιουργεί μια εικόνα σάρωσης (raster image) της τρέχουσας σκηνής και την αποθηκεύει στο αρχείο $filename$. Η εικόνα αυτή σώζεται στο format pbm (δηλαδή, μια επικεφαλίδα με τις διαστάσεις της εικόνας και ένα byte για κάθε pixel-raw data). Οι παράμετροι x, y υποδηλώνουν τις διαστάσεις της εικόνας σάρωσης. Τυπικές τιμές είναι 64, 128, 256. Μπορείτε να δείξετε την εικόνα στην οθόνη με το πρόγραμμα **xv**. Μπορείτε επίσης να μετατρέψετε το αρχείο σε Postscript μέσω του ιδίου προγράμματος και να το στείλετε για εκτύπωση στον εκτυπωτή. Τούτο θα σας είναι χρήσιμο κατά την παράδοση των αποτελεσμάτων της άσκησης.

Η εντολή `include` επιτρέπει την ενσωμάτωση και ερμηνεία εντολών από αρχείο. Τέλος, η εντολή `quit` σταματά την εκτέλεση του προγράμματος.

Οι δοσμένοι κώδικες δίνονται στα Σχήματα 2.25–2.27. Επιπλέον, παρέχεται ο κώδικας του πολύ απλού φωταγωγού `example_rt.c` (Σχήμα 2.28). Για διευκόλυνσή, σας δίνεται και το αρχείο που παράγει τα εκτελέσιμα προγράμματα `/apps/CG/a4/Makefile` στο Σχήμα 2.22.

```
CFLAGS= -g
OBJ= scanner.o rtMisc.o sphere.o rt.o
SRC= scanner.c rtMisc.c sphere.c rt.c

rt: $(OBJ)
cc -o rt $(CFLAGS) $(OBJ) -lm

example_rt: example_rt.o rtMisc.o sphere.o
cc -o example_rt $(CFLAGS) example_rt.o rtMisc.o sphere.o -lm

clean:
rm -f rt $(OBJ)

lint:
lint -u $(SRC)

$(OBJ): rt.h
rt.o rtMisc.o sphere.o: rtInternal.h
scanner.c: scanner.h
```

Σχήμα 2.22 Makefile που παράγει τα εκτελέσιμα προγράμματα

```
% example
camera 2 -8 4   0 0 0   0 0 1
perspective 2 2 1
light 1 -2 3
surface .1 .6 0 1

initTM
cube

initTM
translate -2 0 0
surface .1 .6 .6 16
sphere

initTM
translate 2 0 0
surface .1 .6 .6 150
sphere

initTM

trace 256 128 pic1
```

Σχήμα 2.23 Ενα παράδειγμα εφαρμογής που μπορεί να εκτελεστεί με το **rt**

Δοκιμάστε να εκτελέσετε το πρόγραμμα **example_rt** και να δείτε στην οθόνη το αποτέλεσμα για να αποκτήσετε μια αίσθηση του αντικειμένου της άσκησης. Ενα άλλο παράδειγμα εφαρμογής που θα μπορούσατε να εκτελέσετε με το διερμηνέα του προγράμματος **rt** είναι το ακόλουθο:

Οι ορισμοί των συναρτήσεων του αρχείου που πρόκειται να συμπληρώσετε είναι οι εξής:

```
typedef struct {
double v[3];
} Point, Vector;

typedef struct {
double Ka, Kd, Ks;
int n;
} Surface;
```

Σχήμα 2.24: Συνεχίζεται

```
extern char *rt_Start();
extern char *rt_End();
extern char *rt_Scale();
extern char *rt_Rotate();
extern char *rt_Translate();
extern char *rt_Shear();
extern char *rt_InitTM();
extern char *rt_PushTM();
extern char *rt_PopTM();
extern char *rt_Camera();
extern char *rt_Perspective();
extern char *rt_Sphere();
extern char *rt_Plane();
extern char *rt_Cube();
extern char *rt_Light();
extern char *rt_Surface();
extern char *rt_InitObjects();
extern char *rt_Trace();
```

Σχήμα 2.24 Αρχείο rt.h: Ορισμοί των συναρτήσεων που εμφανίζονται στο ζητούμενο κώδικα rt.c

Ο σκελετός του ζητούμενου κώδικα δίνεται στο Σχήμα 2.29.

```
/*
 * front end for graphics program; scan input and call appropriate functions
 *
 * written by John Amanatides, 1987, 1995
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "scanner.h"
#include "rt.h"

#define MAX_IN_DEPTH 8
#define LINE_LENGTH 256
#define OUTPUT(string) if(input == stdin) ((void) printf(string))
/*
 * decode first word of string
 */
static int
DecodeWord(list, string)
Words *list;
char *string;
{
int i, result;
char command[LINE_LENGTH];

if(string == NULL || *string == '\n')
return COMMENT;
(void) sscanf(string, "%s", command);

result= ERROR;
for(i=0;list[i].word != NULL && result == ERROR; i++) {
if(strcmp(command,list[i].word) == 0)
result= list[i].code;
}
return result;
}
/*
 * perform scanning of input
 */
```

Σχήμα 2.25: Συνεχίζεται

```
static void
Dialogue(input, indirectDepth)
FILE *input;
int indirectDepth;
{
FILE *newFile;
char line[LINE_LENGTH], stringArg[LINE_LENGTH];
char *result;
double x, y, z;
double a, d, s;
int ix, iy, n;
int axis1, axis2;
Point from, to;
Vector up;

while(1) {
result = NULL;
OUTPUT("? ");
if(fgets(line, LINE_LENGTH, input) == NULL)
return; /* End of File */

switch(DecodeWord(Commands, line)) {

case ERROR:
default:
(void) printf("Eh?\n");
break;
case QUIT:
return;
case COMMENT:
break;
case INCLUDE:
if(indirectDepth >= MAX_IN_DEPTH) {
(void) printf("Sorry, the depth of indirection is too great.\n");
break;
}
(void) sscanf(line,"%*s %s", stringArg);
if((newFile= fopen(stringArg,"r")) == NULL)

(void) printf("Sorry, cannot open file '%s' for input.\n",stringArg);
else {
```

Σχήμα 2.25: Συνεχίζεται

```
Dialogue(newFile, indirectDepth+1);
(void) fclose(newFile);
}
break;
case SCALE:
(void) sscanf(line, "%*s %lf %lf %lf", &x, &y, &z);
result= rt_Scale(x, y, z);
break;
case ROTATE:
(void) sscanf(line, "%*s %d %lf", &axis1, &x);
result= rt_Rotate(axis1, x);
break;
case TRANSLATE:
(void) sscanf(line, "%*s %lf %lf %lf", &x, &y, &z);
result= rt_Translate(x, y, z);
break;
case SHEAR:
(void) sscanf(line,"%*s %d %d %lf", &axis1, &axis2, &x);
result= rt_Shear(axis1, axis2, x);
break;
case INIT_TM:
result= rt_InitTM();
break;
case PUSH_TM:
result= rt_PushTM();
break;
case POP_TM:
result= rt_PopTM();
break;
case CAMERA:
up.v[0]= 0.0;
up.v[1]= 0.0;
up.v[2]= 1.0;
(void) sscanf(line, "%*s %lf %lf %lf %lf %lf %lf %lf %lf %lf",
&from.v[0], &from.v[1], &from.v[2],
&to.v[0], &to.v[1], &to.v[2],
&up.v[0], &up.v[1], &up.v[2]);
result= rt_Camera(from, to, up);
break;
case PERSPECTIVE:
(void) sscanf(line, "%*s %lf %lf %lf", &x, &y, &z);
result= rt_Perspective(x, y, z);
break;
```

Σχήμα 2.25: Συνεχίζεται

```
case SPHERE:
result= rt_Sphere();
break;
case PLANE:
result= rt_Plane();
break;
case CUBE:
result= rt_Cube();
break;
case LIGHT:
(void) sscanf(line, "%*s %lf %lf %lf", &x, &y, &z);
result= rt_Light(x, y, z);
break;
case SURFACE:
(void) sscanf(line, "%*s %lf %lf %lf %d", &a, &d, &s,
&n);
result= rt_Surface(a, d, s, n);
break;
case INIT_OBJECTS:
result= rt_InitObjects();
break;
case TRACE:
(void) sscanf(line, "%*s %d %d %s", &ix,&iy, stringArg);
result= rt_Trace(ix, iy, stringArg);
break;
}
if (result != NULL)
(void) fprintf(stderr, "Error in rt command-  %s\n", result);
}
}
main()
{
(void) rt_Start();
Dialogue(stdin, 1);
(void) rt_End();
return 0;
}
```

Σχήμα 2.25 Αρχείο scanner.c: Διερμηνέας του πακέτου rt

```
#include <stdio.h>
#include <math.h>
#include "rtInternal.h"

/*
 * misc vector, affine functions
 */

/*
 * result= m1*m2
 */
Matrix
MultMatrix(m1, m2)
Matrix *m1, *m2;
{
Matrix result;
int r, c, i;

for(r=0; r < 4; r++)
for(c=0; c < 4; c++) {
result.m[r][c]= 0.0;
for(i=0; i < 4; i++)
result.m[r][c] += m1->m[r][i]*m2->m[i][c];
}

return result;
}
/*
 * compute cross product
 */
Vector Cross(a,b)
Vector a, b;
{
Vector result;

result.v[0]= a.v[1]*b.v[2] - a.v[2]*b.v[1];
result.v[1]= a.v[2]*b.v[0] - a.v[0]*b.v[2];
result.v[2]= a.v[0]*b.v[1] - a.v[1]*b.v[0];
return result;
}
```

Σχήμα 2.26: Συνεχίζεται

```
/*
 * make vector unit length;  also, return length of original vector.
 */

double Normalize(a)
Vector *a;
{
double length, scale;

length= DOT(*a,*a);
length= sqrt(length);
if(length != 0.0) {
scale= 1.0/length;
a->v[0] *= scale;
a->v[1] *= scale;
a->v[2] *= scale;
}
return length;
}

/*
 * Transform point from world space to model space
 */
Point InvTransPoint(p, a)
Point p;
Affine *a;
{
int r, c;
double sum;
Point result;

for(r=0; r < 3; r++) {
sum= a->inverseTM.m[r][3];
for(c=0; c < 3; c++)
sum += a->inverseTM.m[r][c]*p.v[c];
result.v[r]= sum;
}
return result;
}
```

Σχήμα 2.26: Συνεχίζεται

```
/*
 * transform vector from world space to model space
 */

Vector InvTransVector(v, a)
Vector v;
Affine *a;
{
int r, c;
double sum;
Vector result;

for(r=0; r < 3; r++) {
sum= 0.0;
for(c=0; c < 3; c++)
sum += a->inverseTM.m[r][c]*v.v[c];
result.v[r]= sum;
}
return result;
}

/*
 * transform point from model space to world space
 */

Point TransPoint(p, a)
Point p;
Affine *a;
{
int r, c;
double sum;
Point result;

for(r=0; r < 3; r++) {
sum= a->TM.m[r][3];
for(c=0; c < 3; c++)
sum += a->TM.m[r][c]*p.v[c];
result.v[r]= sum;
}
return result;
}
```

Σχήμα 2.26: Συνεχίζεται

```
/*
 * transform normal from model space to world space
 */

Vector TransNormal(n, a)
Vector n;
Affine *a;
{
int r, c;
double sum;
Vector result;

for(r=0; r < 3; r++) {
sum= 0.0;
for(c=0; c < 3; c++)
sum += a->inverseTM.m[c][r]*n.v[c];
result.v[r]= sum;
}
return result;
}

/*
 * given incident ray and surface normal, compute reflected direction
 */

Vector ReflectRay(incident, normal)
Vector incident, normal; /* assume unit length */
{
Vector reflect;
double dotIN;

dotIN= DOT(incident, normal);
TIMES(reflect, normal, -2.0*dotIN);
PLUS(reflect, incident, reflect);

return reflect;
}
```

Σχήμα 2.26: Συνεχίζεται

```
/*
 * Compute transmitted direction vector, given incident, normal vectors and IRs.
 * assume incident, normal normalized
 *
 * return 0 if total internal reflection
 */
int TransmitRay(incident, normal, index1, index2, transmit)
Vector incident, normal, *transmit; /* assume unit length */
double index1, index2;
/* index of refraction on both sides of boundary*/
{
double dotIN, index, dotTN;
Vector vtmp;

dotIN= DOT(incident, normal);
TIMES(vtmp, normal, dotIN);
MINUS(vtmp, incident, vtmp);
index= index1/index2;
TIMES(vtmp, vtmp, index);
dotTN= 1.0 - DOT(vtmp, vtmp);
if(dotTN <= 0.0)
return 0; /* total internal reflection */

dotTN= sqrt(dotTN);
TIMES(*transmit, normal, dotTN);
if(dotIN < 0.0)
MINUS(*transmit, vtmp, *transmit)
else
PLUS(*transmit, vtmp, *transmit)
return 1;
}
```

Σχήμα 2.26 Αρχείο rt_Misc.c: Κώδικας για διάφορες βοηθητικές συναρτήσεις

```
#include "rtInternal.h"
extern double sqrt();
/*
 * compute intersection between ray and unit (radius) sphere at origin.
 * Returns MISS if no intersection; otherwise, it returns HIT and
 * sets t to the the distance  to the intersection point and sets the
 * normal vector to the outward facing normal (unit length) at the
 * intersection point.
 */
int IntersectSphere(ray, t, normal)
Ray *ray;
double *t;
Vector *normal;
{
double a, b, c;
double d;
double r1, r2;
Point hit;

a= DOT(ray->direction, ray->direction);
b= 2.0*DOT(ray->direction, ray->origin);
c= DOT(ray->origin, ray->origin) -1.0;

d= b*b-4.0*a*c;
if(d <= 0.0)
return MISS;
if(b < 0.0) {
r2= (-b+sqrt(d))/(2.0*a);
r1= c/(a*r2);
} else {
r1= (-b-sqrt(d))/(2.0*a);
r2= c/(a*r1);
}
if(r1 < EPSILON)
if(r2 < EPSILON)
return MISS;
else *t= r2;
else *t= r1;
TIMES(hit,  ray->direction, *t);
PLUS(hit, ray->origin, hit);
*normal= hit;
return HIT;
}
```

Σχήμα 2.27 Αρχείο sphere.c: Υπολογισμός της τομής ακτίνας με μοναδιαία σφαίρα

```
#include <stdio.h>
#include "rtInternal.h"

extern double Normalize();
extern Point InvTransPoint();
extern Vector InvTransVector();
extern Vector TransNormal();
extern Vector Cross();

#define BACKGROUND 0.0

Affine sphereAffine;
Vector toLight;
static Point camera;
static Vector vx, vy, vz;
static double scaleX, scaleY;

typedef struct {
double Ka, Kd;
} CheapSurface;

CheapSurface surface;


main()
{
InitScene();
Trace(256, 256, "pic");
}

char *
Camera(lookFrom, lookAt, lookUp)
Point lookFrom, lookAt;
Vector lookUp;
{
Vector x, z;

camera= lookFrom;
MINUS(z, lookAt, lookFrom);
if(Normalize(&z) == 0.0)
return "Camera: domain error";
```

Σχήμα 2.28: Συνεχίζεται

```
x= Cross(z, lookUp);
if(Normalize(&x) == 0.0)
return "Camera: domain error";
vx= x;
vy= Cross(x, z);
vz= z;
return NULL;
}

char *
Perspective(distance, width, height)
double distance, width, height;
{
if(distance <= 0.0 || width <= 0.0 || height <= 0.0)
return "Perspective: domain error";
scaleX= width/distance;
scaleY= height/distance;
return NULL;
}

InitScene()
{
int i, j;
Point lookFrom, lookAt;
Vector lookUp;

/*
Vvery simple scene for now; just one ellipsoid and one
light src.  Hack computation of affine for ellipsoid
(scale by two in x and y, not in z and apply to sphere)

*/
for(i=0; i < 4; i++)
for(j=0; j < 4; j++) {
sphereAffine.TM.m[i][j] = 0.0;
sphereAffine.inverseTM.m[i][j] = 0.0;
}
for(i=0; i < 2; i++) {
sphereAffine.TM.m[i][i] = 2.0;
sphereAffine.inverseTM.m[i][i] = 0.5;
}
```

Σχήμα 2.28: Συνεχίζεται

```
sphereAffine.TM.m[2][2] = sphereAffine.inverseTM.m[2][2] = 1.0;
sphereAffine.TM.m[3][3] = sphereAffine.inverseTM.m[3][3] = 1.0;

toLight.v[0]= 0.0;
toLight.v[1]= 0.0;
toLight.v[2]= 1.0;

surface.Ka= 0.2;
surface.Kd= 0.8;

Perspective(1.0, 1.0, 1.0);
printf("camera? ");
scanf("%lf %lf %lf   %lf %lf %lf   %lf %lf %lf",
&lookFrom.v[0], &lookFrom.v[1], &lookFrom.v[2],
&lookAt.v[0], &lookAt.v[1], &lookAt.v[2],
&lookUp.v[0], &lookUp.v[1], &lookUp.v[2]);
Camera(lookFrom, lookAt, lookUp);
/* try 0 -5 0    0 0 0    0 0 1 */
}

int IntersectScene(ray, t, normal)
Ray *ray;
double *t;
Vector *normal;
{
Ray transRay;
Vector normal2;

/* assuming only one sphere in scene */
transRay.origin= InvTransPoint(ray->origin, &sphereAffine);
transRay.direction= InvTransVector(ray->direction, &sphereAffine);
if(IntersectSphere(&transRay, t, &normal2) == HIT) {
*normal= TransNormal(normal2, &sphereAffine);
(void) Normalize(normal);
return HIT;
}
else return MISS;
}
```

Σχήμα 2.28: Συνεχίζεται

```
double Shade(ray, t, normal, surface)
Ray *ray;
double t;
Vector *normal;
CheapSurface *surface;
{
double diffuse, intens;

diffuse= DOT(*normal, toLight);
if(diffuse < 0.0)
diffuse= 0.0;
intens= surface->Kd*diffuse+surface->Ka;

return intens;
}

double GetLight(ray)
Ray *ray;
{
double t, diffuse, intens;
Point hit;
Vector normal;

if(IntersectScene(ray, &t, &normal) == HIT)
intens= Shade(ray, t, &normal, &surface);
else
intens= BACKGROUND;

return intens;

}
```

Σχήμα 2.28: Συνεχίζεται

```
/*
 * given sample point on unit viewing square (0.0 <= u, v <= 1.0)
 * return ray that travels in that direction from the eye
 */

static Ray
ShootRay(u, v)
double u, v;
{
Ray ray;
Vector tmp;

ray.origin= camera;
ray.direction= vz;

if(u < 0.0 || u > 1.0 || v < 0.0 || v > 1.0)
(void) fprintf(stderr, "ShootRay: domain error\n");
else {
TIMES(tmp, vx, (u-0.5)*scaleX);
PLUS(ray.direction, tmp, ray.direction);
TIMES(tmp, vy, (v-0.5)*scaleY);
PLUS(ray.direction, tmp, ray.direction);
(void) Normalize(&ray.direction);
}

return ray;
}

Trace(xRes, yRes, filename)
int xRes, yRes;
char *filename;
{
FILE *fp;
int x, y;
double u, v;
Ray ray;
int pix;

if((fp= fopen(filename, "w")) == 0) {
fprintf(stderr, "sorry, couldn't open '%s'\n", filename);
exit(1);
}
```

Σχήμα 2.28: Συνεχίζεται

```
/* write pgm (portable greymap) header */
(void) fprintf(fp, "P5 %d %d 255\n", xRes, yRes);

for(y=0; y < yRes; y++) {
for(x= 0; x < xRes; x++) {
u= ((double) x) /xRes;
v= 1.0-((double) y) /yRes;
ray= ShootRay(u, v);
pix= GetLight(&ray)*255.0;
if(pix > 255)
pix= 255;
fputc(pix, fp);
}
}
fclose(fp);
}
```

Σχήμα 2.28 Αρχείο example_rt.c: Ενας πολύ απλός φωταγωγός

```
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "rtInternal.h"

extern Vector Cross();
extern double Normalize();
extern int IntersectSphere();

static Matrix identity= {       1.0, 0.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 1.0};
static Affine CTM;
static StackNode *CTMstack;
static ListNode *scene;
static Vector lightSourceDirection;

static Point camera;
static Vector vx, vy, vz;
static double scaleX, scaleY;


char *
rt_Start()
{
CTM.TM= identity;
CTM.inverseTM= identity;
CTMstack= NULL;

return NULL;
}

char *
rt_End()
{
return NULL;
}
```

Σχήμα 2.29: Συνεχίζεται

```
char *
rt_InitTM()
{
CTM.TM= identity;
CTM.inverseTM= identity;
return NULL;
}

char *
rt_PushTM()
{
        StackNode *sp;

sp= (StackNode *) malloc(sizeof(StackNode));
sp->CTM= CTM;
sp->next= CTMstack;
CTMstack= sp;
return NULL;
}

char *
rt_PopTM()
{
        StackNode *sp;

        if(CTMstack != NULL) {
                CTM= CTMstack->CTM;
                sp= CTMstack;
                CTMstack= CTMstack->next;
                free(sp);
return NULL;
        }
        else return "stack underflow";
}

char *
rt_Camera(lookFrom, lookAt, lookUp)
Point lookFrom, lookAt;
Vector lookUp;
{
Vector x, z;
```

Σχήμα 2.29: Συνεχίζεται

```
camera= lookFrom;
MINUS(z, lookAt, lookFrom);
if(Normalize(&z) == 0.0)
return "rt_Camera: domain error";
x= Cross(z, lookUp);
if(Normalize(&x) == 0.0)
return "rt_Camera: domain error";
vx= x;
vy= Cross(x, z);
vz= z;
return NULL;
}


char *
rt_Perspective(distance, width, height)
double distance, width, height;
{
if(distance <= 0.0 || width <= 0.0 || height <= 0.0)
return "rt_Perspective: domain error";
scaleX= width/distance;
scaleY= height/distance;
return NULL;
}

/*
 * given sample point on unit viewing square (0.0 <= u, v <= 1.0)
 * return ray that travels in that direction from the eye
 */

static Ray
ShootRay(u, v)
double u, v;
{
Ray ray;
Vector tmp;

ray.origin= camera;
ray.direction= vz;

if(u < 0.0 || u > 1.0 || v < 0.0 || v > 1.0)
(void) fprintf(stderr, "ShootRay: domain error\n");
```

Σχήμα 2.29: Συνεχίζεται

```
else {
TIMES(tmp, vx, (u-0.5)*scaleX);
PLUS(ray.direction, tmp, ray.direction);
TIMES(tmp, vy, (v-0.5)*scaleY);
PLUS(ray.direction, tmp, ray.direction);
(void) Normalize(&ray.direction);
}

return ray;
}


char *
rt_Scale(sx, sy, sz)
double sx, sy, sz;
{
/* replace the following with your code */
(void) printf("scale %g %g %g\n", sx, sy, sz);
return NULL;
}


char *
rt_Rotate(axis, angle)
int axis;
double angle;
{
/* replace the following with your code */
(void) printf("rotate %d %g\n", axis, angle);
return NULL;
}

char *
rt_Translate(tx, ty, tz)
double tx, ty, tz;
{
/* replace the following with your code */
(void) printf("translate %g %g %g\n", tx, ty, tz);
return NULL;
}
```

Σχήμα 2.29: Συνεχίζεται

```
char *
rt_Shear(axis1, axis2, shear)
int axis1, axis2;
double shear;
{
/* replace the following with your code */
(void) printf("shear %d %d %g\n", axis1, axis2, shear);
return NULL;
}


char *
rt_Sphere()
{
/* replace the following with your code */
        (void) printf("sphere\n");
return NULL;
}


char *
rt_Plane()
{
/* replace the following with your code */
        (void) printf("plane\n");
return NULL;
}


char *
rt_Cube()
{
/* replace the following with your code */
        (void) printf("cube\n");
return NULL;
}
```

Σχήμα 2.29: Συνεχίζεται

```
char *
rt_Light(x, y, z)
double x, y, z;
{
lightSourceDirection.v[0]= x;
lightSourceDirection.v[1]= y;
lightSourceDirection.v[2]= z;
if (Normalize(&lightSourceDirection) == 0.0)
return "rt_Light: domain error";
return NULL;
}


char *
rt_Surface(Ka, Kd, Ks, n)
double Ka, Kd, Ks;
int n;
{
/* replace the following with your code */
        (void) printf("surface %lg %lg %lg %d\n", Ka, Kd, Ks, n);
return NULL;
}


char *
rt_InitObjects()
{
/* replace the following with your code */
return NULL;
}


char *
rt_Trace(xRes, yRes, filename)
int xRes, yRes;
char *filename;
{
/* replace the following with your code */
        (void) printf("trace %d %d %s\n", xRes, yRes, filename);
return NULL;
}
```

Σχήμα 2.29 Αρχείο rt.c: Κώδικας που ζητείται να συμπληρωθεί

3

Βιβλιογραφικές Εργασίες

3.1 Συλλογή και αποτίμηση πληροφοριών για hardware Γραφικών Υπολογιστή

Σκοπός:

Η έρευνα, συλλογή και αποτίμηση πληροφοριών που αφορούν το υλικό (hardware) για εξοπλισμό θέσεων εργασίας με υψηλές δυνατότητες γραφικών.

Πηγές:

Οι σχετικές πληροφορίες θα αντληθούν στα τεύχη του περιοδικού *Computer Graphics World* και από τις σελίδες του web.

Ομάδες: Τέσσερις (4) των δύο (2) ατόμων.

Αντικείμενα έρευνας:

1. Workstations & PCs: Διαβάθμιση με βάση τα τεχνικά χαρακτηριστικά (Ομάδα CGW1).
2. Peripherals I : Display devices; printers; hardcopy devices (Ομάδα CGW2).
3. Peripherals II : Cards, mass storage devices (Ομάδα CGW3).
4. Software Packages, π.χ. PHIGS, PEX, GL, RenderMan κ.ο.κ. (Ομάδα CGW4).

Διάρκεια Ερευνας: Δέκα (10) εβδομάδες.

Παραδοτέα:

Εβδομάδα	1η Δέσμη	2η Δέσμη	3η Δέσμη	4η Δέσμη
1	CGW1	CGW2	CGW3	CGW4
2	CGW4	CGW1	CGW2	CGW3
3	CGW3	CGW4	CGW1	CGW2
4	CGW2	CGW3	CGW4	CGW1
5	Παράδοση περιοδικών			
6-10	Συγγραφή εργασίας			

Πίνακας 3.1 Χρονοδιάγραμμα υλοποίησης της εργασίας

Φάκελλος που θα περιέχει το κείμενο της εργασίας (όροι σύγκρισης, υποθέσεις, συγκριτική μελέτη, πίνακες κ.ο.κ.) γραμμένο σε LaTeX και θα παραπέμπει σε παραρτήματα με φωτοτυπίες των σχετικών πληροφοριών που ευρέθησαν στις σελίδες των τευχών του περιοδικού και σχετική δισκέτα. Η εργασία (χωρίς τα παραρτήματα) να μην υπερβαίνει τις τριάντα (30) τυπωμένες σελίδες. Η εργασία θα αποτελείται από πρόλογο, θα διαρθρώνεται σε κεφάλαια και θα καταλήγει με την παράθεση της σχετικής βιβλιογραφίας.

Μεθοδολογία εργασίας - Χρονοδιάγραμμα:

1. Τα σχετικά τεύχη του περιοδικού χωρίστηκαν σε τέσσερις δέσμες. (Σημείωση: Λείπουν τα τεύχη Sep, Oct, Noe 92.)
2. Κάθε ομάδα θα χρεωθεί από το Εργαστήριο Τεχνητής Νοημοσύνης & Επεξεργασίας Πληροφοριών (Ομάδα Ψηφιακής Επεξεργασίας Εικόνας & Τεχνητής Ορασης) μια δέσμη περιοδικών, όπως παρακάτω:
   - η ομάδα CGW1 θα χρεωθεί τα τεύχη Mar 90 – Mar 91 (Τεύχη: 13),
   - η ομάδα CGW2 θα χρεωθεί τα τεύχη Apr 91 – Apr 92 (Τεύχη: 13),
   - η ομάδα CGW3 θα χρεωθεί τα τεύχη May 92 – Jul 93 (Τεύχη: 13),
   - η ομάδα CGW1 θα χρεωθεί τα τεύχη Sep 93 – Aug 94 (Τεύχη: 12).
3. Καθεμιά ομάδα θα έχει στη διάθεσή της μια δέσμη περιοδικών για μία (1) εβδομάδα. Μετά μία εβδομάδα οι ομάδες θα ανταλλάξουν μεταξύ τους τις δέσμες περιοδικών σύμφωνα με τον Πίνακα 3.1.
4. Την πέμπτη εβδομάδα, οι ομάδες που χρεώθηκαν την πρώτη φορά τα περιοδικά, θα τα συγκεντρώσουν και θα τα παραδώσουν στο Εργαστήριο. **Προσοχή:** Υπεύθυνοι και υπόλογοι για οποιαδήποτε απώλεια ή καταστροφή των περιοδικών καθίστανται οι συμμετέχοντες στις ομάδες αυτές.

3.2 Εξελίξεις στη Θεωρία και Εφαρμογές των Γραφικών

Σκοπός:

Η ενημέρωση των φοιτητών για την τρέχουσα εξέλιξη της θεωρίας και των εφαρμογών των Γραφικών Υπολογιστή.

Πηγές:

Οι σχετικές πληροφορίες θα αντληθούν στα τεύχη των περιοδικών *IEEE Computer Graphics & Applications* (CGA) και *IEEE Transactions on Visualization and Computer Graphics.*

Ομάδες: Επτά (7) των δύο (2) ατόμων.

Αντικείμενα έρευνας:

1. Αλγόριθμοι σάρωσης & μετατροπής καμπυλών (Ομάδα CGA1).
2. Αλγόριθμοι ray tracing/casting (Ομάδα CGA2).
3. Χειρουργική υποβοηθούμενη από υπολογιστή -Computer Aided Surgery. Εφαρμογές των γραφικών στη ιατρική με έμφαση τις τρισδιάστατες εικόνες υπερήχων, μαγνητικής και αξονικής τομογραφίας (Ομάδα CGA3).
4. Επιστημονική απεικόνιση - Scientific Visualization (Ομάδα CGA4).
5. Απόδοση όγκων - Volume rendering (Ομάδα CGA5).
6. NURBS - Ενα πακέτο γραφικών για περιγραφή και απόδοση επιφανειών (Ομάδα CGA6).
7. Τριγωνοποίηση Delauney και διαμέριση Voronoi στον τρισδιάστατο χώρο.
8. (Ομάδα CGA7).

Διάρκεια Ερευνας: Δέκα (10) εβδομάδες.

Παραδοτέα:

Κείμενο της εργασίας γραμμένο σε LaTeX (και σχετική δισκέτα) που δεν θα υπερβαίνει τις τριάντα (30) σελίδες μαζί με σχήματα (σε φωτοτυπίες από τα πρωτότυπα και σε εύλογη σμίκρυνση) και κατάλογο αναφορών. Η εργασία θα αποτελείται από πρόλογο, θα διαρθρώνεται σε κεφάλαια και θα καταλήγει με την παράθεση της σχετικής βιβλιογραφίας.

Μεθοδολογία εργασίας - Χρονοδιάγραμμα:

1. Τα σχετικά τεύχη του περιοδικού χωρίστηκαν σε επτά δέσμες.
2. Κάθε ομάδα θα χρεωθεί από το Εργαστήριο Τεχνητής Νοημοσύνης & Επεξεργασίας Πληροφοριών (Ομάδα Ψηφιακής Επεξεργασίας Εικόνας & Τεχνητής Ορασης) μια δέσμη περιοδικών, όπως παρακάτω:

Εβδ.	1η Δέσμη	2η Δέσμη	3η Δέσμη	4η Δέσμη	5η Δέσμη	6η Δέσμη	7η Δέσμη
1	CGA1	CGA2	CGA3	CGA4	CGA5	CGA 6	CGA 7
2	CGA7	CGA1	CGA2	CGA3	CGA4	CGA 5	CGA 6
3	CGA6	CGA7	CGA1	CGA2	CGA3	CGA 4	CGA 5
4	CGA5	CGA6	CGA7	CGA1	CGA2	CGA 3	CGA 4
5	CGA4	CGA5	CGA6	CGA7	CGA1	CGA 2	CGA 3
6	CGA3	CGA4	CGA5	CGA6	CGA7	CGA 1	CGA 2
7	CGA2	CGA3	CGA4	CGA5	CGA6	CGA 7	CGA 1
8	Παράδοση περιοδικών						
9-10	Συγγραφή εργασίας						

Πίνακας 3.2 Χρονοδιάγραμμα υλοποίησης της εργασίας

- η ομάδα CGA1 θα χρεωθεί τα τεύχη Mar 89 – May 90 (Τεύχη: 8),
- η ομάδα CGA2 θα χρεωθεί τα τεύχη Jul 90 – Sep 91 (Τεύχη: 8),
- η ομάδα CGA3 θα χρεωθεί τα τεύχη Nov 91 – Jan 93 (Τεύχη: 8),
- η ομάδα CGA4 θα χρεωθεί τα τεύχη Mar 93 – May 94 (Τεύχη: 8),
- η ομάδα CGA5 θα χρεωθεί τα τεύχη Jul 94 – Sep 95 (Τεύχη: 8).
- η ομάδα CGA6 θα χρεωθεί τα τεύχη Nov 95 – Nov 96 (Τεύχη: 7).
- η ομάδα CGA7 θα χρεωθεί τα πέντε (5) τεύχη του περιοδικού *IEEE Trans. on Visualization and Computer Graphics.*

3. Καθεμιά ομάδα θα έχει στη διάθεσή της μια δέσμη περιοδικών για μία (1) εβδομάδα. Μετά μία εβδομάδα οι ομάδες θα ανταλλάξουν μεταξύ τους τις δέσμες περιοδικών σύμφωνα με τον Πίνακα 3.2.
4. Την όγδοη εβδομάδα, οι ομάδες που χρεώθηκαν την πρώτη φορά τα περιοδικά, θα τα συγκεντρώσουν και θα τα παραδώσουν στο Εργαστήριο. **Προσοχή:** Υπεύθυνοι και υπόλογοι για οποιαδήποτε απώλεια ή καταστροφή των περιοδικών καθίστανται οι συμμετέχοντες στις ομάδες αυτές.

3.3 Διδασκαλία του μαθήματος των Γραφικών μέσα από τις σελίδες του web

Σκοπός:

Η συλλογή πληροφοριών μέσα από το δίκτυο των υπολογιστών για τη διδασκαλία των Γραφικών Υπολογιστή ως προπτυχιακού μαθήματος στα πανεπιστήμια ανά τον κόσμο.

Πηγές:

Οι σχετικές πληροφορίες θα αντληθούν από το URL:

`http://www.education.siggraph.org/`

Ομάδες: Μία (1) των δύο (2) ατόμων.

Διάρκεια Ερευνας: Δέκα (10) εβδομάδες.

Παραδοτέα:

Φάκελλος που θα περιέχει το κείμενο της εργασίας γραμμένο σε LaTeXπου θα συγκεντρώνει, συνοψίζει και αποτιμά τις μεθόδους διδασκαλίας του μαθήματος και θα παραπέμπει σε παραρτήματα με τις εκτυπώσεις των σχετικών σελίδων hypertext και σχετική δισκέτα. Το κείμενο της εργασίας να μην υπερβαίνει τις τριάντα (30) σελίδες (χωρίς τα παραρτήματα). Η εργασία θα αποτελείται από πρόλογο, θα διαρθρώνεται σε κεφάλαια και θα καταλήγει με την παράθεση της σχετικής βιβλιογραφίας.

Βιβλιογραφία

[1] J.D. Foley, A. VanDam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice.* Reading, MA: Addison-Wesley, 2/e 1990; 3/e 1994.

[2] S. Harrington, *Computer Graphics. A Programming Approach.* N.Y.: McGraw Hill, 1987.

[3] D. Hearn and P.M. Baker, *Computer Graphics*, 2/e. Englewood Cliffs, N.J.: Prentice Hall, 1994.

[4] W.M. Newman and R.F. Sproull, *Principles of Interactive Computer Graphics*, 2/e. N.Y.: McGraw-Hill, 1979.

[5] Plastock and Kalley, *Schaum's Outline: Computer Graphics.* N.Y.: McGraw-Hill, 1986.

[6] Rogers and Adams, *Mathematical Elements for Computer Graphics*, 2nd Ed. N.Y.: McGraw-Hill, 1990.

[7] Rogers, *Procedural Elements for Computer Graphics.* N.Y.: McGraw-Hill, 1990.

[8] A. Watt, *Fundamentals of Three-Dimensional Computer Graphics.* Reading, MA: Addison-Wesley, 1989.

[9] A. Watt, *Advanced Animation and Rendering Techniques.* Reading, MA: Addison-Wesley, 1992.

[10] J. Amanatides, *Lecture Notes in Computer Graphics Course at York University*, Computer Science Dept. York University, Canada, 1995.

[11] A. Nye, Ed., *Xlib Reference Manual*, vol. II. O'Reilly & Associates, 1992.