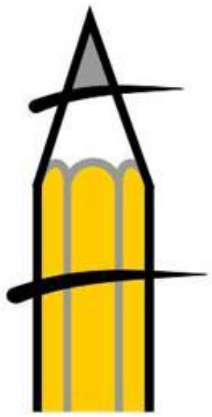


*ovde ide logo škole * ako postoji **

MATURSKI RAD

PREDMET:

Informatika

TEMA:

Osnovi raccunarske grafike

učenik:
Ime i Prezime
Odeljenje :

predmetni nastavnik:
prof. Ime i Prezime

KUĆA:
Radojke Lakić 15 /][
11050 Beograd
Srbija, YUGOSLAVIA

ŠKOLA:
Narodnog Fronta 37
11000 Beograd
Srbija, YUGOSLAVIA

INTERNET:
mailto:brcha@geocities.com
http: / / www.geocities.com / SiliconValley / Campus / 3732 / Grafika.html

Sadržaj

Slike

Glava 1

Uvod

Kompjuterska grafika je relativno nova grana računarstva koja je u poslednje vreme u hiperekspanziji. Od pojave Apple®-ovog Macintosh®-a i MIT-ovog X Window System®-a početkom osamdesetih, kompjuterska grafika se ubrzano razvija i nadograđuje. Sada se već kompjuter skoro i ne može zamisliti bez dobre video i muzičke kartice, velikog monitora, jakih zvučnika, modema i nekog grafičkog operativnog sistema (nažalost uglavnom Microsoft® Windows®-a 95/98), a ni fabrika koja ne nadgleda rad mašina vizualizujući ih na kompjuterima. Shodno tome, kompjuterska grafika bi mogla da se definiše kao igračka ili ukras bez koga se više ne može.

Iako je kompjuterska grafika hyper-moderna „nauka", njeni koreni leže duboko u prošlosti, među prvim čovekovim pokušajima da nešto razume, da nešto otkrije. Logično, mislim na geometriju i matematiku, jer kompjuterska grafika nisu samo lepe slike, animacije, kompjuterske igre i sl. koje vidimo na ekranima (ili VR-Šlemovima). Naprotiv, kompjuterska grafika je vrlo složen matematički aparat koji koristi znanja iz geometrije, i sintetičke i analitičke, linearne algebre, trigonometrije, pa čak i iz estetike.

Glava 2

Osnovni pojmovi

2.1 Osnovne operacije

Dva osnovna pristupa kompjuterskoj grafici su vektorski i rasterski pristup. Vektorski pristup je u neku ruku na višem nivou, i zasnovan je na definisanju objekata pomoću vektora. Moglo bi se reći da su dve osnovne operacije koje izvršava grafički uređaj u vektorskom pristupu komjuterskoj grafici:

1. *moveto(x,y)* pomera kurzor na poziciju (x,y)[1]
2. *lineto(x,y)* crta liniju do pozicije (x,y)[2]

Rasterski pristup grafici je zasnovan na definisanju objekata pomoću matrice PIXELa (tačaka). Svaki PIXEL ima celobrojne koordinate i svoju boju. Zbog toga se rasterski pristup naziva i BITMAP pristup (MAPa BITova). Boja se uglavnom prikazuje kao uređena trojka brojeva koji predstavljaju intenzitet crvene, zelene i plave boje u RGB(RED-GREEN-BLUE) sistemu[3].

Zbog činjenice da se vektorski pristup može lako implementirati unitar rasterskog, i činjenice da je 90% grafičkih uređaja zasnovano na rasterskom pristupu, ja ću se samo na njega i ograničiti. PIXEL ću da modeliram kao uređeni par celobrojnih koordinata (x, y) kome ću da dodeljujem broj koji predstavlja indeks boje. Boje ću da prikazujem kao uređenu trojku brojeva iz skupa $[0, 1](r, g, b)$ kome ću da dodeljujem indeks unutar palete boja (vidi algoritam 2.1.1).

2.1.1 Definicije primitivnih funkcija u C-u

Zbog raznovrsnosti grafičkih uređaja, smatram da je korisno uvesti konvenciju o nazivu osnovnih funkcija raster DISPLAYa (DISPLAY je uređaj na kome se prikazuju grafički objekti), pa ću prvo da napravim HEADER FILE u programskom jeziku C u koji može da se uklopi većina biblioteka za grafiku:

```
/* Fajl: primitives.h */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
```

[1] u ADOBE® POSTSCRIPT®-u: *x y moveto*

[2] u ADOBE® POSTSCRIPT®-u: *x y moveto*

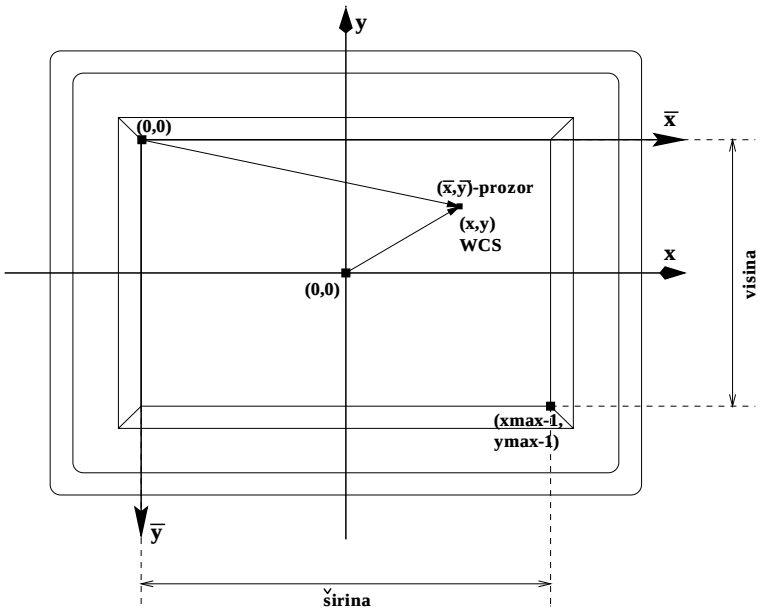
[3] Postoji još i CMY(CYAN-MAGENTA-YELLOW) sistem, HSV(HUE-SATURATION-VALUE, ponekad nazivan i HSB, B=BRIGHTNES) sistem, HLS(HUE-LIGHTNESS-SATURATION)

```
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#ifndef __PRIMITIVES_H
#define __PRIMITIVES_H

/* ISO C Standatd: 4.1.3 Errors <errno.h> */
#include <errno.h>
/* ISO C Standard: 4.2 DIAGNOSTICS <assert.h> */
#include <assert.h>
/* ISO C Standard: 4.3 CHARACTER HANDLING <ctype.h> */
#include <ctype.h>
/* ISO C Standard: 4.5 MATHEMATICS <math.h> */
#include <math.h>
/* ISO C Standard: 4.7 SIGNAL HANDLING <signal.h> */
#include <signal.h>
/* ISO C Standard: 4.9 INPUT/OUTPUT <stdio.h> */
#include <stdio.h>
/* ISO C Standard: 4.11 STRING HANDLING <string.h> */
#include <string.h>
/* POSIX Standard: 2.10 Symbolic Constants <unistd.h> */
#include <unistd.h>

/* memory allocation utilities (Fusion Systems inc.) */
#include <allocUtil.h>
/* stack of any type (Fusion Systems inc.) */
#include <stack.h>
/* boolean type definition (Fusion Systems inc.) */
#include <bool.h>

/* Maksimalna veličina poligona */
#define maxPoly 100

char *programName; /* Ime programa (prozora) */
int numColors; /* Broj boja <- beginGraphics() */
int nXpixels,nYpixels; /* Veličina ekrana/viewport-a <- beginGraphics() */

/* Struktura pixel-a = uređeni par (x,y) */
struct pixelVector{ int x,y; };

int currentColor; /* Tekuća boja */
float *red,*green,*blue; /* Tabela boja, nizovi od <numColors>
                         * elemenata <- beginGraphics
                         */
struct pixelVector currentPixel; /* Tekuća pozicija na ekranu */

/* Osnovne funkcije(primitive) */
void beginGraphics(void); /* Inicijalizacija grafičkog moda */
void updateGraphics(void);
void endGraphics(void);    /* Kraj rada sa grafikom */
void clearDisplay(void);   /* Brisanje ekrana u tekuću boju */
void setColor(int color); /* Namesti tekuću boju */
void setPixel(struct pixelVector pixel);   /* Idi na poziciju pixel i
                                    * oboj je u tekuću boju */
void movePixel(struct pixelVector pixel); /* Idi na poziciju pixel */
```

```
void linePixel(struct pixelVector pixel); /* Nacrtaj liniju od tekuće
                                          * pozicije do „pixel”-a */
void polyPixel(int n,struct pixelVector poly[]); /* Nacrtaj poligon =
                                                  * mnogougao (n-tougao) */
void setrgb(int i,float r,float g,float b); /* Definiši i-tu boju
                                             * kao (r,g,b) */
void flip(void);       /* Iz grafičkog u tekstualni mod i obrnuto
                        * bez brisanja sadržaja ekrana */
/* Stil linije:
 * width - širina
 * lineStyle - LineSolid, LineOnOffDash, LineDoubleDash
 * capStyle - kraj linije: CapNotLast, CapButt, CapRound, CapProjecting
 * joinStyle - način povezivanja: JoinMitter, JoinRound, JoinBevel
 * Za detalje pogledati literaturu za Xlib
 */
void setLineStyle(unsigned int width,int lineStyle,
                  int capStyle,int joinStyle);
#define SET 0
#define OR  1
#define AND 2
#define XOR 3
void setLineType(int type);

bool keyPressed(void); /* Da li je pritisnut taster */
bool displayExposed(void); /* Da li je pokvaren prikaz (interno) */
int getChar(void); /* Pročitaj karakter ili -1, ako ništa nije pritisnuto */

#endif /* __PRIMITIVES_H */
/* Kraj fajla primitives.h */
```

2.2 Proširenje ekrana

Velika mana PIXELa su njihove celobrojne koordinate. U 2D prostoru sa nima još i može da se preživi, ali pri samoj pomisli na tri dimenzije od njih se mora odustati. Zbog toga treba uvesti koordinate kao realne brojeve i transformacije tih koordinata u koordinate DISPLAYa. Treba zamisliti objekte koji se modeliraju (linije, poligone, lopte, slova, ljude, fabrike, mostove, PVO, avione, ...) kao objekte u nekakvom virtualnom grafičkom svetu (vidi sliku 2.1). Kao što u ovom našem svetu ima ∞ mnogo koordinatnih sistema (u daljem tekstu samo KS ili CS od COORDINATE SYSTEM), i u tom virtualnom svetu može da se definiše ∞ mnogo KS. Ali, u virtualnom, za razliku od našeg, mora da postoji i apsolutni KS (Absolute CS/ACS/ ili World CS/WCS/), da bi se u svakom trenutku znao međusobni položaj objekata bez obzira u kom KS su definisani. Pošto ćemo taj svet gledati iz svih uglova i iz svih pozicija, korisno je definisati i vezu između našeg i kompjuterskog sveta. Naš kraj te veze je ekran ili pogled na tom ekranu (pogled ili VIEWPORT je deo ekrana na kome aktivna aplikacija može da pristupi, npr: prozor u X WINDOW SYSTEM®u ili MICROSOFT® WINDOWS®u). Na drugi kraj vezujemo prozor (WINDOW) definisan u WCSu svojim koordinatnim početkom, širinom i visinom. Rekoh da je prozor definisam svojim koordinatnim početkom. To je u stvari koordinatni početak lokalnog KS prozora, definisan npr. u centru prozora. Sada se na objekat iz virtualnog prostora primenjuje niz transformacija da bi on dospeo na ekran. Prvo se objekat osiromašuje na 2D projekcijom u ravan prozora. Onda se odseca sve što ne može da stane u prozor i na kraju se sve iz prozora skalira po X osi (prilagođavanje razmere tako da se cela širina prozora uklopi) na pogled i tek onda se pomoću jediničnih operacija raster DISPLAYa upisuju PIXELi na DISPLAY. Ako je data širina prozora, njegova visina je $širina * \frac{ymax}{xmax}$. Pošto su koordinate u KS prozora, kao i u WCS, u metrima, milimetrima, jardima, inčima, kilo-



Slika 2.1: Apsolutni i koordinatni sistem ekrana (prozora)

gramima, džulima, njutnima, veberima, teslama, svetlosnim godinama, ili u bilo kojoj drugoj jedinici, moramo da definišemo koeficijent transformacije iz KS prozora u KS pogleda. Zato se definiše $xyscale$ kao $\frac{xmax-1}{širina}$(da bi prozor stao u pogled ceo po širini). Sada dakle možemo definisati funkciju transformacije iz KS prozora u KS pogleda kao:

$$\begin{aligned} \overline{x} &= f_x(x) = \left\lfloor x * xyscale + \tfrac{xmax}{2} \right\rfloor \\ \overline{y} &= f_y(y) = \left\lfloor -y * xyscale + \tfrac{ymax}{2} \right\rfloor \end{aligned} \tag{2.1}$$

Za detalje implementacije, pogledajte programe 2.2.1 i 2.2.2.

2.2.1 Definicija funkcija za realne koordinate

WINDOW2VPORT.H

```
/* Fajl: <window2vport.h> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#ifndef __WINDOW2VPORT_H
#define __WINDOW2VPORT_H

#include "primitives.h"

/* Broj π */
#define pi 3.1415926535
/* ε je najveća moguća preciznost */
#define epsilon 0.000001
```

```
struct vector2D{ float x,y; };    /* 2D i 3D vektori sa    */
struct vector3D{ float x,y,z; }; /* realnim koordinatama */

float horiz,vert,xyscale; /* Veličina prozora i odnos
                          * maksimuma x koordinata u
                          * prrozoru i u viewportu */

void startGraphics(float aHoriz); /* beginGraphics na višem nivou */
void stopGraphics(void);              /* = endGraphics */
int fx(float x); /* x̄ = f_x(x) */
int fy(float y); /* ȳ = f_y(y) */
void moveTo(struct vector2D pt); /* ≈ movePixel */
void lineTo(struct vector2D pt); /* ≈ linePixel */
void polyFill(int n,struct vector2D polygon[]); /* ≈ polyPixel */

/* Glavni deo programa - umesto main */
int graphMain(int argc,char *argv[]);

#endif /* __WINDOW2VPORT_H */
/* Kraj fajla <window2vport.h> */
```

2.2.2 Implementacija finkcija za realne koordinate

WINDOW2VPORT.C

```
/* Fajl: <window2vport.c> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#include "window2vport.h"

void startGraphics(float aHoriz){
    beginGraphics();
    horiz = aHoriz;
    vert = horiz*nYpixels/nXpixels;
    xyscale = (nXpixels-1)/horiz;
}

void stopGraphics(void){
    endGraphics();
}

int fx(float x){
    return (int)(x*xyscale+nXpixels*0.5-0.5);
}

int fy(float y){
    return (int)(-y*xyscale+nYpixels*0.5-0.5);
}
```

```
void moveTo(struct vector2D pt){
    struct pixelVector pixel;
    pixel.x = fx(pt.x);
    pixel.y = fy(pt.y);
    movePixel(pixel);
}

void lineTo(struct vector2D pt){
    struct pixelVector pixel;
    pixel.x = fx(pt.x);
    pixel.y = fy(pt.y);
    linePixel(pixel);
}

void polyFill(int n,struct vector2D polygon[]){
    int i;
    struct pixelVector *pixelPolygon;

    if(n>2){
        pixelPolygon = (struct pixelVector*)
       safeCalloc((unsigned)n,sizeof(struct pixelVector));
        for(i=0;i<n;i++){
        pixelPolygon[i].x = fx(polygon[i].x);
        pixelPolygon[i].y = fy(polygon[i].y);
        }
        polyPixel(n,pixelPolygon);
        safeFree(pixelPolygon);
    }
}

int main(int argc,char *argv[]){
    float horizontalSize;
    int programResult;

    printf("\nUnesite sirinu prozora: ");
    scanf("%f",&horizontalSize);
    startGraphics(horizontalSize);
    programResult = graphMain(argc,argv);
    stopGraphics();
    return programResult;
}
/* Kraj fajla <window2vport.c> */
```

2.2.3 Krive

Sada verovatno mislite: „Dobro, imamo tačke, linije, poligone, prozore i poglede, imamo virtualni prostor, ali šta se dogodilo sa krivim linijama ?!”. E, pa, postoji više načina da se i one prikažu. Mogu da se crtaju PIXEL po PIXEL, što najbolje aproksimira tu krivu na ekranu, a mogu i da se interpretiraju kao n-tougao ili n povezanih linija, gde je n veliki broj. Ovaj drugi način je lakši za implementaciju, zgodniji za geometrijske transformacije, a koriste ga mnogi eminentni programi za grafiku (kao što je AUTODESK® AUTOCAD®), pa ću se opredeliti za njega (primer: program 2.2.3.1 koristeći taj princip crta krug kao 100-ugao).

2.2.3.1 Program za crtanje kruga kao 100-ugla

CIRCLE.C

```
#include "window2vport.h"

void circle(float r){
    float theta,thinc;
    int i;
    struct vector2D pt;

    theta = 0;
    thinc = 2*pi/100;
    /* Idi na početnu tačku */
    pt.x = r;
    pt.y = 0.0;
    moveTo(pt);
    /* Nacrtaj stranice 100-ugla */
    for(i=0;i<100;i++){
        theta += thinc;
        pt.x = r*cos(theta);
        pt.y = r*sin(theta);
        lineTo(pt);
    }
}

int graphMain(int argc,char *argv[]){
    float r;
    struct vector2D pt;

    printf("\nUnesite poluprecnik kruga: ");
    scanf("%f",&r);
    pt.x = 0; pt.y = 0;
    moveTo(pt);
    pt.x = 1; pt.y = 1;
    lineTo(pt);
    circle(r);
    while(getChar() != 'q');
    return 0;
}
```

Glava 3

2D Prostor

3.1 Geometrija 2D prostora

3.1.1 Prave

Kao za jedan od osnovnih elemenata u grafici, za pravu, mora se naći pogodna analitička reprezentacija. Uobičajeni oblik $y = kx + n$ (ili $ax + by + c = 0$) ovde nije najzgodniji. Mnogo je lakše prikazati pravu kao linearnu kombinaciju vektora položaja dveju poznatih tačaka te prave ($\overrightarrow{p_1}$ i $\overrightarrow{p_2}$):

$$\overrightarrow{p}(\mu) = (1-\mu)\overrightarrow{p_1} + \mu\overrightarrow{p_2},\ \mu \in R \tag{3.1}$$

što je ekvivalentno sa

$$\overrightarrow{p}(\mu) = \overrightarrow{p_1} + \mu(\overrightarrow{p_2} - \overrightarrow{p_1}). \tag{3.2}$$

Zamenom $\overrightarrow{p_2} - \overrightarrow{p_1}$ sa $\overrightarrow{q}$ dobija se sistem:

$$\begin{cases} \overrightarrow{p}(\mu) = \overrightarrow{p_1} + \mu\overrightarrow{q} \\ \overrightarrow{q} = \overrightarrow{p_2} - \overrightarrow{p_1} \end{cases} \tag{3.3}$$

gde je $\mu = \frac{razdaljina\ od\ \overrightarrow{p_1}\ do\ \overrightarrow{p}(\mu)}{razdaljina\ od\ \overrightarrow{p_1}\ do\ \overrightarrow{p_2}}$, a $\overrightarrow{p}(\mu)$ proizvoljna tačka na pravoj. Vektor $\overrightarrow{q}$ (3.3) predstavlja vektor pravca (DIRECTION VECTOR), a vektor $\overrightarrow{p_1}$ bazni vektor (BASE VECTOR).

Sada možemo da rešimo problem preseka dve prave, $\overrightarrow{\wp_1}(\mu) = \overrightarrow{p} + \mu\overrightarrow{q}$ i $\overrightarrow{\wp_2}(\lambda) = \overrightarrow{r} + \lambda\overrightarrow{s}$ koji se svodi na

$$\overrightarrow{\wp_1}(\mu) = \overrightarrow{\wp_2}(\lambda) \tag{3.4}$$

odnosno na

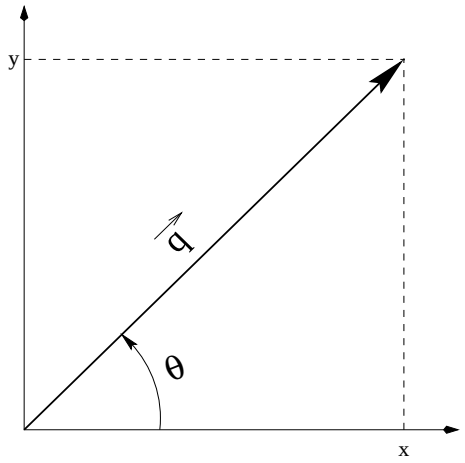
$$\overrightarrow{p} + \mu\overrightarrow{q} = \overrightarrow{r} + \lambda\overrightarrow{s}. \tag{3.5}$$

Rastavljanjem (3.5) na x i y ose dobija se:

$$\begin{cases} x_p + \mu x_q = x_r + \lambda x_s \\ y_p + \mu y_q = y_r + \lambda y_s \end{cases} \tag{3.6}$$

što je ekvivalentno sa

$$\begin{cases} \mu x_q - \lambda x_s = x_r - x_p \\ \mu y_q - \lambda y_s = y_r - y_p \end{cases}. \tag{3.7}$$



Slika 3.1: Vektor pravca

Množenjem prve jednačine sistema (3.7) sa y_s, a druge sa $-x_s$ i njihovom kombinacijom dobija se

$$\mu(x_q y_s - y_q x_s) = (x_r - x_p)y_s - (y_r - y_p)x_s \tag{3.8}$$

odakle je

$$\mu = \frac{(x_r - x_p)y_s - (y_r - y_p)x_s}{x_q y_s - y_q x_s}. \tag{3.9}$$

Presek pravih je

$$\overrightarrow{p} + \mu\overrightarrow{q}. \tag{3.10}$$

Funkcija koja računa presek dve prave u dvodimenzionalnom prostoru je LINEINTERSECT2D() iz fajla GRAPHUTILITIES.H / .C (dodatak A na strani 45). Argumenti funkcije su vektori $\overrightarrow{p}$, $\overrightarrow{q}$, $\overrightarrow{r}$ i $\overrightarrow{s}$ i pokazivač na vektor preseka tih pravih (jednačine (3.10) i (3.9)).

3.1.2 Vektori pravca

Ako pogledamo još jednom jednačinu prave (3.3), primetićemo da $\overrightarrow{q}$ predstavlja, kao što sam već rekao, vektor pravca prave $\overrightarrow{p}(\mu)$ (vidi sliku 3.1). Ako je θ ugao izmedju x-ose i vektora $\overrightarrow{q}$, koordinate vektora $\overrightarrow{q}$ su $|\overrightarrow{q}|\cos\theta$ i $|\overrightarrow{q}|\sin\theta$, pa ugao θ možemo da izračunamo iz

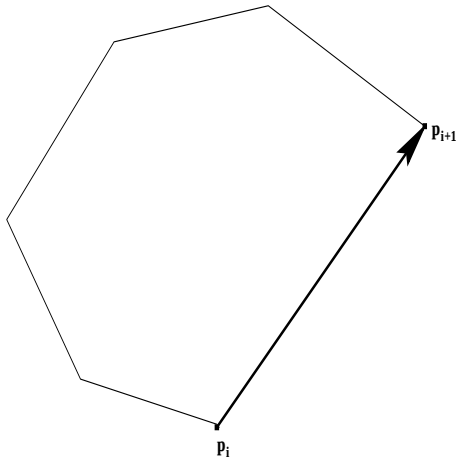
$$\frac{q_y}{q_x} = \tan\theta. \tag{3.11}$$

To radi funkcija ANGLE() iz fajla GRAPHUTILITIES.H / .C (dodatak A na strani 45), kojoj se kao argumenti daju koordinate vektora $\overrightarrow{q}$, a vraća ugao θ (3.11).

3.1.3 Šta je unutra, a šta napolju?

Vratimo se za trenutak na analitički oblik prave

$$f(x, y) \equiv ax + by + c. \tag{3.12}$$



Slika 3.2: Konveksan n-tougao

Funkcija prave koja prolazi kroz tačke $\overrightarrow{p_1}(x_1, y_1)$ i $\overrightarrow{p_2}(x_2, y_2)$ sada glasi:

$$\begin{cases} (x, y) = \overrightarrow{p_1} + \mu\overrightarrow{q} \\ \overrightarrow{q} = \overrightarrow{p_2} - \overrightarrow{p_1} = (x_q = x_2 - x_1, y_q = y_2 - y_1) \end{cases} . \tag{3.13}$$

Rastavljanjem sistema (3.13) na x i y osu dobija se:

$$\begin{cases} x = x_1 + \mu x_q \\ y = y_1 + \mu y_q \end{cases} . \tag{3.14}$$

Kombinacijom jednačina iz sistema (3.14) sa koeficijentima $-y_q$ i x_q dobija se da je

$$yx_q - xy_q = y_1x_q + \mu y_q x_q - x_1y_q - \mu x_q y_q \tag{3.15}$$

što je ekvivalentno sa

$$x_q y - y_q x - (y_1x_q - x_1y_q) = 0 \tag{3.16}$$

odnosno

$$x_q(y - y_1) - y_q(x - x_1) = 0. \tag{3.17}$$

Zamenom $x_q = x_2 - x_1$ i $y_q = y_2 - y_1$ dobija se

$$(x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1) = 0, \tag{3.18}$$

pa je dakle

$$f(x, y) \equiv (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1) \tag{3.19}$$

analitički oblik funkcije prave kroz tačke $\overrightarrow{p_1}$ i $\overrightarrow{p_2}$. Tačke za koje je $f(x, y) = 0$ pripadaju toj pravoj, a dve tačke $(\overline{x_1}, \overline{y_1})$ i $(\overline{x_2}, \overline{y_2})$ su sa iste strane akko su $f(\overline{x_1}, \overline{y_1})$ i $f(\overline{x_2}, \overline{y_2})$ istog znaka[1].

Ako imamo konveksan n-tougao zadat svojim uzastopnim temenima (vidi sliku 3.2):

$$\overrightarrow{p_i} \equiv (x_i, y_i),\ i = \overrightarrow{o, n-1}, \tag{3.20}$$

[1]isto važi i za konveksne krive, samo jje njima drugačija funkcija, npr: $f(x, y) \equiv ax^2 + by^2 + cxy + dx + ey + f$ za krive drugog reda

možemo da odredima šta je unutar n-tougla, a šta van njega. Funkcija koja odgovara i-toj stranici n-tougla glasi

$$f_i(x,y) \equiv (x_{i+1} - x_i)(y - y_i) - (y_{i+1} - y_i),\ i = \overrightarrow{o, n-1} \tag{3.21}$$

uz napomenu da je $\overrightarrow{p_n} = \overrightarrow{p_0}$. Tačka (x, y) je unutar n-tougla akko je $f_i(x, y) > 0,\ i = \overrightarrow{0, n-1}$ ako su temena zadata u pozitivnom matematičkom smeru (suprotno od kazaljke na satu), a ako su zadata u suprotnom smeru (u smeru kazaljke na satu), akko je $f_i(x, y) < 0$. To će kasnije biti veoma značajno za određivanje šta je u pozadini u trodimenzionalnom koordinatnom sistemu.

3.2 2D transformacije

Kad sam spominjao Apsolutni KS i KS prozora, rekao sam da je prozor (i njegov KS) definisan u odnosu na Apsolutni KS. Ali zašto bismo se ograničili na samo jedan položaj prozora. Bolje bi bilo kad bismo mogli da „šetamo" taj prozor kroz naš virtuelni prostor i razgledamo stvari iz svih uglova. Za to će biti neophodne razno ravanske (za sada) transformacije, koje se svode na kombinacije translacije, rotacije i skaliranja (menjanja razmere). Definišimo ih:

- Translacija tačke $\overrightarrow{P}(x, y)$ za vektor $\overrightarrow{T}(d_x, d_y)$ u tačku $\overrightarrow{P'}(x', y')$ se definiše ovako:

$$\begin{cases} x' = x + d_x \\ y' = y + d_y \end{cases}, \tag{3.22}$$

  ili vektorski

$$\overrightarrow{P'} = \overrightarrow{P} + \overrightarrow{T}, \tag{3.23}$$

  ili matrično

$$\begin{Bmatrix} x' \\ y' \end{Bmatrix} = \begin{Bmatrix} x \\ y \end{Bmatrix} + \begin{Bmatrix} d_x \\ d_y \end{Bmatrix}. \tag{3.24}$$

- Skaliranje s_x puta po x-osi i s_y puta po y-osi je ustvari množenje x i y sa s_x i s_y, redom:

$$\begin{cases} x' = s_x x \\ y' = s_y y \end{cases}, \tag{3.25}$$

  ili matrično

$$\begin{Bmatrix} x' \\ y' \end{Bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} \tag{3.26}$$

  ili

$$\overrightarrow{P} = S(s_x, s_y)\overrightarrow{P} \tag{3.27}$$

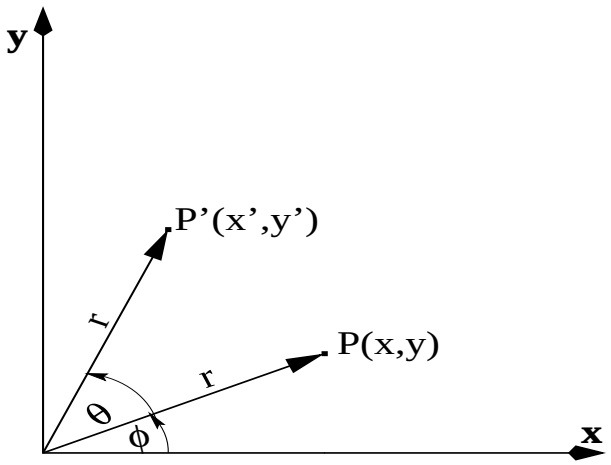
  gde je $S(s_x, s_y)$ matrica iz jednačine (3.26).

- Rotacija je već malo komplikovanija. Sa slike 3.3se vidi da je

$$\begin{cases} x = r\cos\phi \\ y = r\sin\phi \end{cases}, \tag{3.28}$$

  a takođe i da je

$$\begin{cases} x' = r\cos(\theta + \phi) = r\cos\theta\cos\phi - r\sin\theta\sin\phi \\ y' = r\sin(\theta + \phi) = r\sin\theta\cos\phi + r\cos\theta\sin\phi \end{cases}. \tag{3.29}$$



Slika 3.3: Rotacija tačke

Ako zamenimo (3.28) u (3.29) dobijamo:

$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{cases} . \tag{3.30}$$

U matričnoj formi taj sistem dobija oblik:

$$\begin{Bmatrix} x' \\ y' \end{Bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} \tag{3.31}$$

ili

$$\overrightarrow{P'} = R(\theta)\overrightarrow{P} \tag{3.32}$$

gde je $R(\theta)$ matrica iz jednačine (3.31).

Da bi se transformacije mogle lakše kombinovati, potrebno je sve prikazati preko proizvoda ili preko zbira. Pošto se skaliranje i rotacija ne mogu (ili vrlo teško mogu?) prikazati pomoću zbira nekakvih matrica ili vektora, pokušajmo da translaciju prikažemo kao proizvod. Nameće se da vektor tačke proširimo još jednim, trećim poljem sa vrednošću 1, a matrice transformacija da proširimo na 3x3. Tada dobijamo *homogene koordinate tačaka*:

$$P_H = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.33}$$

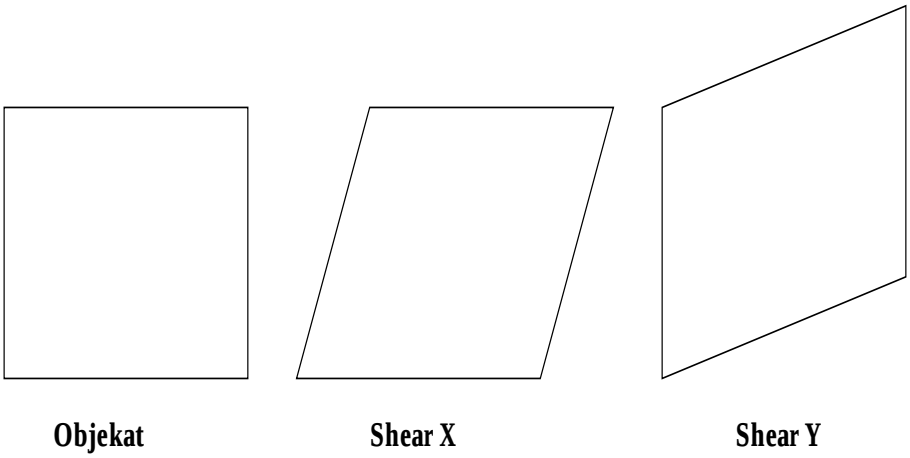
i unifikovane transformacije:

- Translacija:

$$T_H(d_x, d_y) = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.34}$$

- Skaliranje:

$$S_H(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.35}$$



Slika 3.4: Transformacija smicanja (SHEAR)

- Rotacija:

$$R_H(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.36}$$

Očigledno je da ovime nisu iscrpljene sve moguće transformacije. Postoji još i transformacija smicanja (SHEAR) po x i po y osi (vidi sliku 3.4):

$$SH_x(a) = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.37}$$

i

$$SH_y(b) = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.38}$$

3.2.1 Kompozicije 2D Transformacija

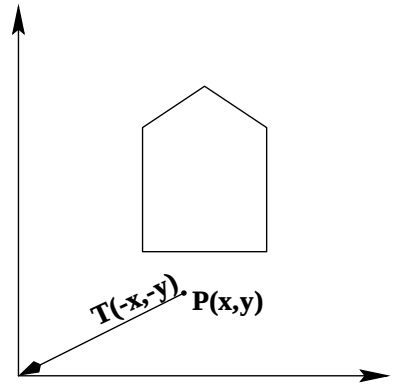
Da bi se uštedelo vreme, umesto da se na objekat (tačku) primenjuje više matrica transformacija za redom, bolje je izračunati rezultujuću matricu i samo nju i primeniti. Na primer, problem rotiranja objekta oko neke tačke P nije elementaran (već predviđen matricama $T(d_x, d_y)$, $S(s_x, s_y)$, $R(\theta)$, $SH_x(a)$ i $SH_y(b)$), već se može prikazati kao kombinacija tri elementarne transformacije:

1. Translacija za vektor $-\overrightarrow{P}$ - tačka P ide u koordinatni početak (vidi sliku 3.5 za početni izgled i sliku 3.6 za izgled posle translacije)
2. Rotacija za neki ugao θ (vidi sliku 3.7)
3. Translacija za vektor $\overrightarrow{P}$ - tačka P se vraća iz koordinatnog početka na svoj prvobitni položaj (vidi sliku 3.8)

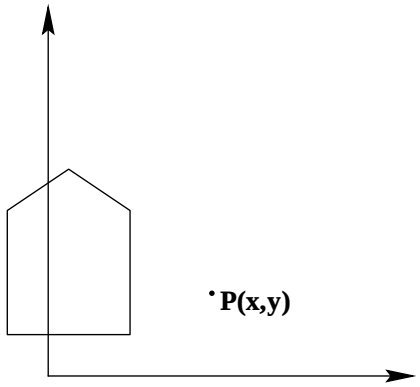
Kompozicija ovih matrica se vrši u obrnutom poretku od izvršavanja transformacija (jer se novi položaj tačke računa kao $\overrightarrow{P'} = M_{3\times 3} \times \overrightarrow{P}$, a operacija množenja matrica nije komutativna), pa je tako rezultujuća matrica

$$R = T(x, y) \times R(\theta) \times T(-x, -y) \tag{3.39}$$

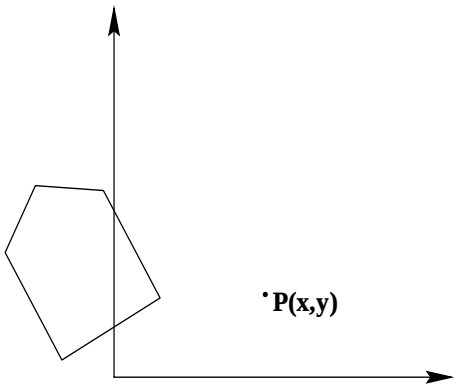
Za detalje implementacije pogledajte programe 3.2.3 na strani 24 i 3.2.4 na strani 25.



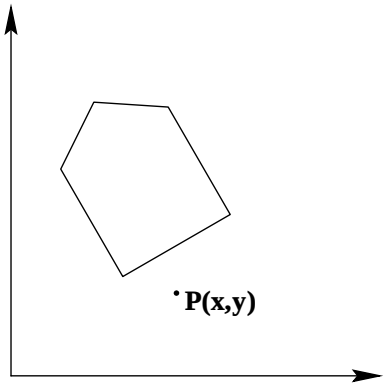
Slika 3.5: Originalna pozicija objekta



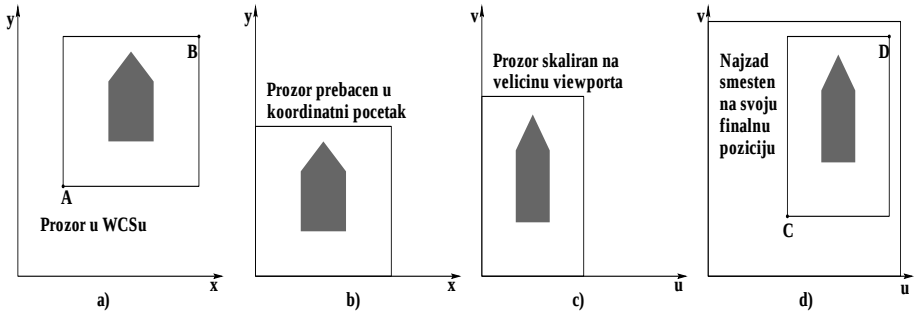
Slika 3.6: Transliran objekat za vektor T sa slike 3.5 - $T(-x, -y)$



Slika 3.7: Objekat rotiran za 30° - $R(\theta = 30^\circ)$



Slika 3.8: Objekat vraćen na mesto - $T(x, y)$



Slika 3.9: Transformacija prozor -> VIEWPORT

3.2.2 Transformacija Prozor->VIEWPORT

Evo ga jedan lep primer primene matričnih transformacija i njihove kompozicije. Kako prozor i VIEWPORT nisu obavezno istih dimenzija, nemaju obavezno iste koordinate, itd., potrebno je pre prikazivanja PIXELa prozor skalirati na veličinu VIEWPORTa i dovesti ga na mesto VIEWPORTa (vidi sliku 3.9).Ako donji levi ugao prozora (tačka A na slici 3.9 pod a)) ima koordinate (x_{min}, y_{min}), a gornji desni ugao (tačka B na istoj slici) koordinate (x_{max}, y_{max}), a VIEWPORT ima uglove (u_{min}, v_{min}) i (u_{max}, v_{max}) (tačke C i D na slici 3.9 pod d)), onda se, dakle, primenjuju sledeće transformacije:

1. $T(-x_{min}, y_{min})$ - slika 3.9 pod b)
2. $S\left(\frac{u_{max}-u_{min}}{x_{max}-x_{min}}, \frac{v_{max}-v_{min}}{y_{max}-y_{min}}\right)$ - slika 3.9 pod c)
3. $T(u_{min}, v_{min})$ - slika 3.9 pod d)

pa je rezultujuća matrica

$$M_{W2V} = T(u_{min}, v_{min}) \times S\left(\frac{u_{max} - u_{min}}{x_{max} - x_{min}}, \frac{v_{max} - v_{min}}{y_{max} - y_{min}}\right) \times T(-x_{min}, -y_{min}). \qquad (3.40)$$

3.2.3 Definicija funkcija za matrične transformacije dvodimenzionalnog prostora

MATRIX2D.H

```
/* Fajl: <matrix2D> */
/*
```

```
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#ifndef __MATRIX2D_H
#define __MATRIX2D_H

#include "window2vport.h"

/* tip trans2D i njemu odgovarajući "malloc()" i "free()" */
typedef double** trans2D;

trans2D initTrans2D();
void finishTrans2D(trans2D);

/* 2D matrične transformacije koordinatnog početka: */
void translate2D(float dx,float dy,trans2D T); /* Translacija za (dx,dy)
*/
void scale2D(float sx,float sy,trans2D T); /* (1,1) ↦ (sx,sy) */
void rotate2D(float theta,trans2D T); /* Koordinatne ose se rotiraju za ∠θ */
void compose2D(trans2D A,trans2D B,trans2D C); /* C = A × B */

#endif /* __MATRIX2D_H */
/* Kraj fajla <matrix2D> */
```

3.2.4 Implementacija funkcija za matrične transformacije dvodimenzionalnog prostora

MATRIX2D.C

```
/* Fajl: <matrix2D.c> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#include "matrix2D.h"

/* Alociranje memorije za trans2D */
trans2D initTrans2D(){
    return dmatrix(1,3,1,3); /* Iz "liballocUtil.so" by Fusion Systems */
}

void finishTrans2D(trans2D T){
    assert(T!=NULL);
    return free_dmatrix(T,1,3,1,3); /* Iz "liballocUtil.so" by Fusion Systems */
}
```

```
/* 2D matrične transformacije koordinatnog početka: */

/* Translacija za (dx,dy) */
void translate2D(float dx,float dy,trans2D T){
    int i,j;
    assert(T!=NULL);
    for(i=1;i<=3;i++){
        for(j=1;j<=3;j++)
       T[i][j] = 0.0;
        T[i][i] = 1.0;
    }
    T[1][3] = -dx;
    T[2][3] = -dy;
}

/* (1,1) ↦ (sx,sy) */
void scale2D(float sx,float sy,trans2D T){
    int i,j;
    assert(T!=NULL);
    for(i=1;i<=3;i++)
       for(j=1;j<=3;j++)
          T[i][j] = 0.0;
    T[1][1] = sx;
    T[2][2] = sy;
    T[3][3] = 1.0;
}

/* Koordinatne ose se rotiraju za ∠θ */
void rotate2D(float theta,trans2D T){
    int i;
    float c,s;
    assert(T!=NULL);
    for(i=1;i<=3;i++){
        T[i][3] = 0.0;
        T[3][i] = 0.0;
    }
    T[3][3] = 1.0;
    c = cos(theta);
    s = sin(theta);
    T[1][1] = c;
    T[2][2] = c;
    T[1][2] = s;
    T[2][1] = -s;
}

/* C = A × B */
void compose2D(trans2D A,trans2D B,trans2D C){
    int i,j,k;
    float ab;
    assert(A!=NULL);
    assert(B!=NULL);
    assert(C!=NULL);
    for(i=1;i<=3;i++)
       for(j=1;j<=3;j++){
```

```
       ab = 0;
       for(k=1;k<=3;k++)
          ab+=A[i][k]*B[k][j];
       C[i][j] = ab;
       }
}

/* Kraj fajla <matrix2D.c> */
```

3.3 Objekti u 2D grafičkom prostoru

Da bi se u svakom trenutku znalo šta se gde nalazi, još ranije je uveden apsolutni KS (WCS - vidi poglavlje 2.2 „Proširenje ekrana" na strani 11), a sada se uvodi i baza podataka koja sadrži informacije o pozicijjama tih objekata u WCSu. Za osnovni element uzima se tačka i za svaku se pamte njene koordinate i dodeljuje joj se jedinstveni broj (RECORD ID u jeziku baza podataka). Za linije se pamte indeksi krajnjih tačaka. Poligoni, ovde nazvani FACETi (FACET∼stranica poliedra), se čuvaju kao liste tačaka, a svakom je dodeljena i boja (strukture i funkcije za modeliranje objekata date su u listinzima 3.3.1 na strani 27 i 3.3.2 na strani 28).

3.3.1 Definicija struktura i funkcija za modeliranje objekata

MODEL2D.H

```
/* Fajl: <model2D.h> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#ifndef __MODEL2D_H
#define __MODEL2D_H

#include "matrix2D.h"

/* Maksimalan broj tačaka, linija, FACETa i elemenata liste */
#define maxVertices 400
#define maxLines 400
#define maxFacets 400
#define maxList 2000

struct linePoints{
    int front,back; /* Početna i krajnja tačka linije */
};

/* Prvi slobodan element liste FACETa */
int firstFreeListElement;
/* Trenutan broj tačaka, linija i FACETa */
int nVertices,nLines,nFacets;
/* Tačke u pozicijama ACTUAL, OBSERVER i SETUP */
struct vector2D actual[maxVertices],observer[maxVertices],setup[maxVertices];
/* Linije van FACETa */
```

```
struct linePoints line[maxLines];
/* i-ti FACET:
 * boja: facetColor[i]
 * tačke: facetList[facetFirst[i]+j], j = 0, (facetSize[i] - 1)
*/
int facetColor[maxFacets]; /* Boje FACETa */
int facetFirst[maxFacets]; /* Početci FACETa */
int facetList[maxList]; /* Lista tačaka u FACETima */
int facetSize[maxFacets]; /* Veličine FACETa */
/* KS gledaoca - OBSERVER CS (OCS) */
float alpha; /* Ugao, u odnosu na x-osu, pod kojim se gleda */
struct vector2D eye; /* Pozicija gledaoca (oka) */
struct vector2D eyeScale; /* Odnos x, odn. y koordinata sistema */
trans2D toObserver; /* Transformacija iz ACTUAL u OBSERVER KS */

/* Transformacija v u (*w) preko transformacije A */
void transform(struct vector2D v,trans2D A,struct vector2D *w);
/* Još jedna zamena za main() */
int sceneMain(int argc,char *argv[]);

#endif /* __MODEL2D_H */
/* Kraj fajla <model2D.h> */
```

3.3.2 Implementacija funkcija za modeliranje objekata

MODEL2D.C

```
/* Fajl: <model2D.c> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#include "model2D.h"

/* Transformacija v u (*w) preko transformacije A
 * ulaz: v,A
 * izlaz: w
 */
void transform(struct vector2D v,trans2D A,struct vector2D *w){
    w->x = A[1][1]*v.x+A[1][2]*v.y+A[1][3];
    w->y = A[2][1]*v.x+A[2][2]*v.y+A[2][3];
}

int graphMain(int argc,char *argv[]){
    int result;
    firstFreeListElement = 0;
    nVertices = nLines = nFacets = 0;
    toObserver = initTrans2D();
    result = sceneMain(argc,argv); /* Nova zamena za main() */
    finishTrans2D(toObserver);
    return result;
```

```
};

/* Kraj fajla <model2D.c> */
```

3.3.3 Koordinatni sistem gledaoca

Sada može da se definiše i koordinatni sistem gledaoca (OBSERVER CS) uglom nagiba glave, koordinatama oka i odnosom jedinica x i y koordinata apsolutnog KS i KS gledaoca. To je u stvari koordinatni sistem koji omogućuje „šetanje" kroz grafički prostor. Pošto se tedino tačkama pamte koordinate, samo za njih i treba uvesti OBSERVER CS. Zbog toga svaka tačka ima svoj „stvarni" (ACTUAL) položaj u WCSu i svoj relativni položaj u odnosu na gledaoca, u OCSu.

Ponekad je lakše objekat prvo definisati u nekom pogodnijem položaju (npr. u okolini koordinatnog početka), pa tek onda ga transformacijama smestiti na pravi položaj i promeniti mu širinu i visinu. Ta pogodnija, početna pozicija zove se SETUP pozicija. SETUP pozicije tačaka nekog objekta mogu se i zapamtiti i iskoristiti za stvaranje više instanci istog objekta sa različitom veličinom i drugim položajem. Konstrukcija scene u ovakvom sistemu sastoji se iz tri glavne faze:

1. Faza definisanja tačaka u SETUP pozicijama i definisanja linija i FACETa;
2. Faza postavljanja tačaka iz SETUP u ACTUAL pozicije;
3. Faza postavljanja gledaoca na svoje mesto (korišćenjem funkcija FINDA2O() i LOOK2D() iz fajlova DISPLAY2D.H / .C - program 3.3.3.1 na strani 29 i 3.3.3.2 na strani 30).

3.3.3.1 Definicija funkcija za manipulisanje OBSERVER CSom

DISPLAY2D.H

```
/* Fajl: <display2D.h> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#ifndef __DISPLAY2D_H
#define __DISPLAY2D_H

#include "model2D.h"

/* Nalaženje transformacije iz ACTUAL u OBSERVER KS */
void findA2O(void);
/* Definicija OBSERVER KSa */
void look2D(void);
/* Transformacija tačaka iz ACTUAL u OBSERVER KS */
void observe(void);
/* Crtanje linija i FACETa */
void drawIt(void);

void redraw(void);

#endif /* __DISPLAY2D_H */
/* Kraj fajla <display2D.h> */
```

3.3.3.2 Implementacija funkcija za manipulisanje OBSERVER CSom

DISPLAY2D.C

```
/* Fajl: <display2D.c> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#include "display2D.h"

/* Nalaženje transformacije iz ACTIVE u OBSERVER KS */
void findA2O(void){
    trans2D A,B,C;

    A = initTrans2D();
    B = initTrans2D();
    C = initTrans2D();
    if(!toObserver)
       toObserver = initTrans2D();
    translate2D(eye.x,eye.y,A);
    rotate2D(alpha,B);
    compose2D(B,A,C);
    scale2D(eyeScale.x,eyeScale.y,A);
    compose2D(C,A,toObserver);
    finishTrans2D(C);
    finishTrans2D(B);
    finishTrans2D(A);
}

/* Definicija OBSERVER KSa */
void look2D(void){
    printf("\nInicijalizacija observer sistema");
    printf("\nUnesite x i y koordinate oka i nagib glave[deg]: ");
    scanf("%f %f %f",&eye.x,&eye.y,&alpha);
    printf("\nUnesite odnos x i odnos y koordinata sistema: ");
    scanf("%f %f",&eyeScale.x,&eyeScale.y);
    alpha *= pi/180;
    findA2O();
}

/* Transformacija ta\v caka iz ACTIVE u OBSERVER KS */
void observe(void){
    int i;
    for(i=0;i<nVertices;i++)
       transform(actual[i],toObserver,&observer[i]);
}

/* Crtanje linija i FACETa */
void drawIt(void){
    int i,j,vertex;
```

```
    struct vector2D polygon[maxPoly];

    for(i=0;i<nFacets;i++){
        for(j=0;j<facetSize[i];j++){
        vertex = facetList[facetFirst[i]+j];
        polygon[j].x = observer[vertex].x;
        polygon[j].y = observer[vertex].y;
        }
        setColor(facetColor[i]);
        polyFill(facetSize[i],polygon);
    }
    setColor(4);
    for(i=0;i<nLines;i++){
        moveTo(observer[line[i].front]);
        lineTo(observer[line[i].back]);
    }
}


void redraw(void){
    if(displayExposed()){
        updateGraphics();
        vert=horiz*nYpixels/nXpixels;
        xyscale=(nXpixels-1)/horiz;

        findA2O();
        observe();
        clearDisplay();
        drawIt();
    }
}

/* Kraj fajla <display2D.c> */
```

3.3.4 Konstrukcija objekata

Te faze mogu da se izvrše „ručnim" popunjavanjem odgovarajućih struktura ili korišćenjem pomoćnih funkcija iz fajlova CONSTRUCT2D.H / .C - poglavlje **??** na strani **??** (pravljene po ugledu na SGI OpenGL^{tm}). Struktura tih funkcija je otprilike ovakva:

- BEGINOBJECT() - početak definicije objekta, mora da se pozove kao prva funkcija pri definiciji novog objekta
  - DEFINEPOINT(X,Y) - postavljanje tačke na SETUP poziciju. Prvi poziv definiše tačku pod brojem 0, drugi tačku broj 1, treći tačku broj 2, ... ,n-ti poziv tačku $n-1$
  - ADDLINE(P1,P2) - definisanje linije od tačke sa indeksom P1 do tačke sa indeksom P2
  - BEGINFACET() - početak definicije FACETa
    - ADDPOINT(P) - dodavanje tačke sa indeksom P u listu temena tekućeg FACETa. Logično, mora biti pozvana posle BEGINFACET(), a pre ENDFACET()
    - ADDCOLOR(C) - dodeljivanje boje FACETu. Kao i ADDPOINT(P), mora biti pozvana između BEGIN i ENDFACET()
  - ENDFACET() - kraj definicije FACETa (unutar objekta može da bude definisano više FACETa)

- ENDOBJECT(S2A) - kraj definicije objekta i njegovo postavljanje u ACTUAL poziciju pomoću transformacije S2A.

Detalji implementacije i zavisnosti funkcija mogu da se vide u programima 3.3.4.1 na strani 32 i 3.3.4.2 na strani 32.

3.3.4.1 Definicija funkcija za definiciju objekata

CONSTRUCTION2D.H

```
/* Fajl: <construction2D.h> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#ifndef __CONSTRUCTION2D_H
#define __CONSTRUCTION2D_H

#include "display2D.h"

bool beginObject(void); /* Početak definicije objekta ---------+ */
bool definePoint(float x,float y); /* Definicija tačke                  | */
bool addLine(int front,int back); /* Dodavanje linije u listu linija     | */
bool beginFacet(void); /* Početak definicije FACETa ---+ | */
bool addPoint(int i); /* Dodavanje tačke u listu temena poligona   | | */
bool setFacetColor(int c); /* Definicija boje poligona                    | | */
bool endFacet(void); /* Kraj definicije FACETa ---------+ | */
bool endObject(trans2D toActive); /* Kraj definicije objekta -------+ */

#endif /* __CONSTRUCTION2D_H */
/* Kraj fajla <construction2D.h> */
```

3.3.4.2 Implementacija funkcija za definiciju objekata

CONSTRUCITION2D.C

```
/* Fajl: <construction2D.c> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#include "construction2D.h"

static bool defineObject = 0; /* Da li se trenutno definiše objekat */
static bool defineFacet = 0; /* Da li se trenutno definiše FACET */
static int setupPointCount; /* Trenutna setup tačka */

/* Početak definicije objekta */
```

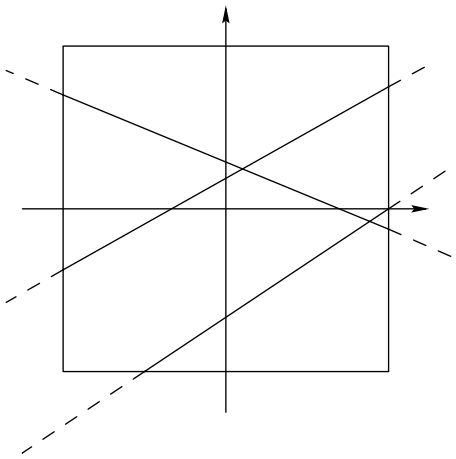
```
bool beginObject(void){
    if(defineObject==true) return false;
    defineObject = true;
    setupPointCount = 0;
    return true;
}

/* Definicija tačke */
/* Prvi poziv: tačka 0,
 * Drugi poziv: tačka 1,
 * ...
 * n-ti poziv: tačka n-1.
 */
bool definePoint(float x,float y){
    if((defineObject==false)||
        ((setupPointCount+nVertices)>=maxVertices)) /* Nema više mesta */
       return false;
    setup[setupPointCount].x = x;
    setup[setupPointCount].y = y;
    setupPointCount++;
    return true;
}

/* Dodavanje linije u listu linija */
bool addLine(int front,int back){
    if((defineObject==false)||
        (nLines>=maxLines)|| /* Nema mesta za liniju */
        ((front+nVertices)>= maxVertices)|| /* Nema mesta */
        ((back+nVertices)>= maxVertices)||  /* za tačke */
        (front>=setupPointCount)|| /* Ne postoje */
        (back>=setupPointCount))    /* tačke */
       return false;
    line[nLines].front = front+nVertices;
    line[nLines].back = back+nVertices;
    nLines++;
    return true;
}

/* Početak definicije FACETa */
bool beginFacet(void){
    if((defineObject==false)||
        (defineFacet==true)||
        (nFacets>=maxFacets)) /* Nema mesta u listi */
       return false;
    defineFacet = true;
    facetFirst[nFacets] = firstFreeListElement;
    facetColor[nFacets] = 1; /* Ako se ne definiše boja, biće 1 */
    facetSize[nFacets] = 0; /* Nema tačaka */
    return true;
}

/* Dodavanje tačke u listu temena poligona */
bool addPoint(int i){
    if((defineFacet==false)||
```

```
        (nFacets>=maxFacets)|| /* Nema više FACETa */
        (firstFreeListElement>=maxList)|| /* Nema mesta u listi */
        ((i+nVertices)>=maxVertices)|| /* Nemam gde da je upišem */
        (i>=setupPointCount)) /* Ne postoji ta tačka */
      return false;
    facetList[firstFreeListElement++] = i+nVertices;
    facetSize[nFacets]++;
    return true;
}

/* Definicija boje poligona */
bool setFacetColor(int c){
    if((defineFacet==false)||
        (nFacets>=maxFacets)) /* Nema više mesta */
      return false;
    facetColor[nFacets] = c;
    return true;
}

/* Kraj definicije FACETa */
bool endFacet(void){
    int i;

    if((defineFacet==false)||
        (nFacets>=maxFacets))
      return false;
    nFacets++;
    defineFacet = false;
    return true;
}

/* Kraj definicije objekta */
bool endObject(trans2D toActual){
    int i;

    if(defineObject==false) return false;
    for(i=0;i<setupPointCount;i++)
      transform(setup[i],toActual,&actual[nVertices++]);
    defineObject = false;
    return true;
}

/* Kraj fajla <construction2D.c> */
```

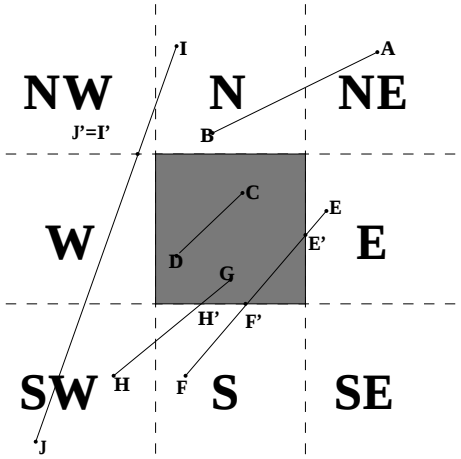
3.4 Još neke korisne stvari u 2D prostoru

3.4.1 LINE CLIPPING („granice prikaza linija”)

Obzirom da ekran, odn. prozor nekog WMa (WINDOW MANAGER) nema beskonačne dimenzije, negde moramo da stavimo granicu i da „isečemo” (CLIP) sve van njih. Za početak ograničimo se na linije. Pošto je centar OBSERVER CSa u centru prozora i pošto je prozor pravougaonog oblika (ili nije ?!), naša granica bi'ce pravougaonik sa temenima $(\pm clipedX, \pm clipedY)$ gde su $clipedX$ i $clipedY$ polovine širine (po x-osi) i visine (po y-osi) prozora (slika 3.10).Ako se stranice



Slika 3.10: Prikaz „sečenja” linija



Slika 3.11: Podela ravni produženjem stranica pravougaonika

pravougaonika produže beskonačno, cela ravan se deli na 9 delova (slika 3.11).Ako te delove prikažemo kao uređene parove (ix, iy) gde $ix, iy \in \{-1, 0, 1\}$ (tabela 3.1), vrlo lako se određuju parametri ix i iy za proizvoljnu tačku, a time i njen položaj u odnosu na granični pravougaonik (CLIP RECTANGLE). Za x koordinatu dobija se vrednost ix pomoću sledećeg algoritma:

```
if x < -clipedX
then return -1
elif clipedX > x
then return 1
else return 0
fi
```

$iy \setminus ix$	-1	0	+1
+1	NW	N	NE
0	W	+	E
-1	SW	S	SE

Tabela 3.1: Uređeni parovi i delovi koordinatnog sistema

Istim algoritmom dobija se i iy.

Kako sad pomoću toga odraditi 2D LINE CLIPPING? Prvo da vidimo šta se dešava sa linijjama. Ako su krrajevi neke linije tačke p_1 i p_2 sa njima odgovarajućim parametrima (ix_1, iy_1) i (ix_2, iy_2), postoje tri različita slučaja:

1. Ako je $ix_1 = ix_2 \neq 0$ ili $iy_1 = iy_2 \neq 0$, onda je cela linija sigurno van pravougaonika i može se ignorisati (linija $\overline{AB}$ na slici 3.11).

2. Ako je $ix_1 = iy_1 = ix_2 = iy_2 = 0$, onda je cela linija sigurno unutar pravougaonika, pa cela mora biti nacrtana (linija $\overline{CD}$ na slici 3.11).

3. Ako nije zadovoljen ni prvi, ni drugi slučaj, dolazi do usložnjavanja problema. Ako je $ix_1 \neq 0$ i/ili $iy_1 \neq 0$, onda je tačka $p_1 \equiv (x_1, y_1)$ van pravougaonika i mora da se nađe nova tačka $p_1\prime \equiv (x_1\prime, y_1\prime)$, koja će biti na ivici pravougaonika. Proces nalaženja preseka dve prave, u ovom slučaju prave (p_1, p_2) sa najbližom stranicom pravougaonika, obrađen je u poglavlju 3.1 na strani 17. U slučaju da je $ix_1 = iy_1 = 0$, onda je $p_1\prime = p_1$. Na isti način se određuje i tačka $p_2\prime$. Ukoliko je $p_1\prime = p_2\prime$, prava se ignoriše (linija $\overline{IJ}$ na slici 3.11), a u suprotnom, crta se prava $(p_1\prime, p_2\prime)$ (linija $\overline{EF}$ prelazi u $\overline{E\prime F\prime}$, a $\overline{GH}$ u $\overline{GH\prime}$ gde je $G = G\prime$).

Ukoliko je potrebno ograničiti linije negde drugde na crtežu, novi „granični pravougaonik" mora da sadrži još par informacija, i to koordinate centra, odn. preseka dijagonala (x_c, y_c) i ugao nagiba u odnosu na x-osu sistema, α. Tada se mogu izvršiti geometrijjske transformacije (poglavlje 3.2) pre poziva algoritma **??** za računanje ix odnosno iy. Nadalje se sve dešava isto kao i kod prethodno objašnjenog, pojednostavljenog LINE CLIPPINGa. Postoji i suprotan algoritam od LINE CLIPPINGa. To je LINE BLANKING, kod kog se, upravo obratno, briše sve unutar pravougaonika, a van njega se ostavlja. Programi za CLIPPING i BLANKING su dati u fajlu CLIP2D.H / .C u listinzima 3.4.1.1 na strani 36 i 3.4.1.2 na strani 37.

3.4.1.1 Definicija funkcija i promenljivih za LINE CLIPPING/BLANKING

CLIP2D.H

```
/* Fajl: <clip2D.h> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#ifndef __CLIP2D_H
#define __CLIP2D_H

#include "window2vport.h"

bool clipping,blanking; /* Da li treba da se vrši clipping/blanking */
float clipedX,clipedY; /* granice pisanja */
float blankedX,blankedY; /* granice brisanja */

int pointMode(float coord,float clipBound);
void setClippingMode(float boundX,float boundY);
void setBlankingMode(float boundX,float boundY);
bool clipLine(struct vector2D p1,struct vector2D p2);
bool blankLine(struct vector2D p1,struct vector2D p2);
```

```
#endif /* __CLIP2D_H */
/* Kraj fajla <clip2D.h> */
```

3.4.1.2 Implementacija funkcija za LINE CLIPPING/BLANKING

CLIP2D.C

```
/* Fajl: <clip2D.c> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#include "clip2D.h"

/*
 * Funkcija koja vraća IX (ili IY) za zadatu
 * X (ili Y) koordinatu i granicu prikazivanja
 */
int pointMode(float coord,float clipBound){
    if(coord < -clipBound)
       return -1; /* manje od granice */
    if(coord > clipBound)
       return +1; /* više od granice */
    return 0; /* unutar granica */
}

void setClippingMode(float boundX,float boundY){
    clipping = true;
    clipedX = boundX;
    clipedY = boundY;
}

void setBlankingMode(float boundX,float boundY){
    blanking = true;
    blankedX = boundX;
    blankedY = boundY;
}

bool clipLine(struct vector2D p1,struct vector2D p2){
    struct vector2D p1d,p2d;
    int ix1,iy1,ix2,iy2;
    float xx,yy;

    if(!clipping) /* ERROR: Nema granica ! */
       return false;
    /* Inicijalizacija p1d i p2d */
    p1d.x = p1.x; p1d.y = p1.y;
    p2d.x = p2.x; p2d.y = p2.y;
    /* Inicijalizacija ix1,iy1,ix2,iy2 */
    ix1 = pointMode(p1d.x,clipedX); iy1 = pointMode(p1d.y,clipedY);
    ix2 = pointMode(p2d.x,clipedX); iy2 = pointMode(p2d.y,clipedY);
```

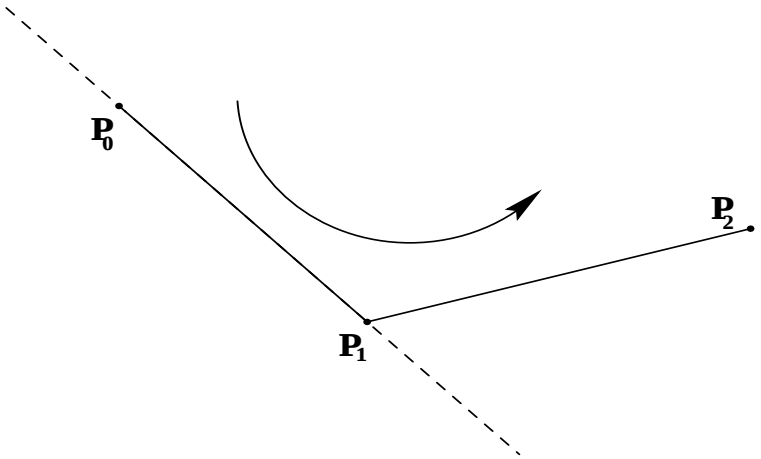
```
    /* Ignoriši prvi slučaj */
    if((ix1*ix2 != 1) && (iy1*iy2 != 1)){
        if(iy1 != 0){ /* p1d je van granica y ose */
        yy = clipedY*iy1;
        p1d.x = p1d.x + (p2d.x-p1d.x)*(yy-p1d.y)/(p2d.y-p1d.y);
        p1d.y = yy;
        ix1 = pointMode(p1d.x,clipedX);
        }
        if(ix1 != 0){ /* p1d je van granica x ose */
        xx = clipedX*ix1;
        p1d.y = p1d.y + (p2d.y-p1d.y)*(xx-p1d.x)/(p2d.x-p1d.x);
        p1d.x = xx;
        }
        /* Sad isto za drugu tačku */
        if(iy2 != 0){ /* p2d je van granica y ose */
        yy = clipedY*iy2;
        p2d.x = p1d.x + (p2d.x-p1d.x)*(yy-p1d.y)/(p2d.y-p1d.y);
        p2d.y = yy;
        ix2 = pointMode(p2d.x,clipedX);
        }
        if(ix2 != 0){ /* p2d je van granica x ose */
        xx = clipedX*ix2;
        p2d.y = p1d.y + (p2d.y-p1d.y)*(xx-p1d.x)/(p2d.x-p1d.x);
        p2d.x = xx;
        }
        /* Linija od p1d do p2d uz moguć blanking */
        if((fabs(p1d.x-p2d.x) > epsilon) ||
        (fabs(p1d.y-p2d.y) > epsilon)){ /* Linija postoji */
        if(blanking)
           return blankLine(p1d,p2d);
        else{
            moveTo(p1d);
            lineTo(p2d);
        }
        }
    }
    return true;
}

bool blankLine(struct vector2D p1,struct vector2D p2){
    struct vector2D p1d,p2d;
    int ix1,iy1,ix2,iy2;

    if(!blanking) /* Nema granica */
       return false;
    /* Nađi ix1,iy1,ix2,iy2 */
    ix1 = pointMode(p1.x,blankedX); iy1 = pointMode(p1.y,blankedY);
    ix2 = pointMode(p2.x,blankedX); iy2 = pointMode(p2.y,blankedY);
    /* Ako su tačke slučaja jedan, nacrtaj liniju */
    if((ix1*ix2 == 1) || (iy1*iy2 == 1)){
        moveTo(p1);
        lineTo(p2);
        return true;
    }
```

```
    if(ix1 != 0){ /* Tačka p1 je van x granica, izračunaj drugu tačku */
        p1d.x = blankedX*ix1;
        p1d.y = p1.y + (p2.y-p1.y)*(p1d.x-p1.x)/(p2.x-p1.x);
        iy1 = pointMode(p1d.y,blankedY);
    }else{
        p1d.x = p1.x;
        p1d.y = p1.y;
    }
    if(iy1 != 0){ /* p1d je van y granica */
        p1d.y = blankedY*iy1;
        p1d.x = p1.x + (p2.x-p1.x)*(p1d.y-p1.y)/(p2.y-p1.y);
    }
    /* Prvi deo linije je gotov, nacrtaj ga */
    if((fabs(p1d.x-p1.x) > epsilon) ||
        (fabs(p1d.y-p1.y) > epsilon)){
        moveTo(p1);
        lineTo(p1d);
    }
    /* Isto za tačke p2 i p2d */
    if(ix2 != 0){ /* Tačka p2 je van x granica, izračunaj drugu tačku */
        p2d.x = blankedX*ix2;
        p2d.y = p1.y + (p2.y-p1.y)*(p2d.x-p1.x)/(p2.x-p1.x);
        iy2 = pointMode(p2d.y,blankedY);
    }else{
        p2d.x = p2.x;
        p2d.y = p2.y;
    }
    if(iy2 != 0){ /* p2d je van y granica */
        p2d.y = blankedY*iy2;
        p2d.x = p1.x + (p2.x-p1.x)*(p2d.y-p1.y)/(p2.y-p1.y);
    }
    /* Drugi deo linije je gotov, nacrtaj ga */
    if((fabs(p2d.x-p2.x) > epsilon) ||
        (fabs(p2d.y-p2.y) > epsilon)){
        moveTo(p2);
        lineTo(p2d);
    }
    return true;
}

/* Kraj fajla <clip2D.c> */
```

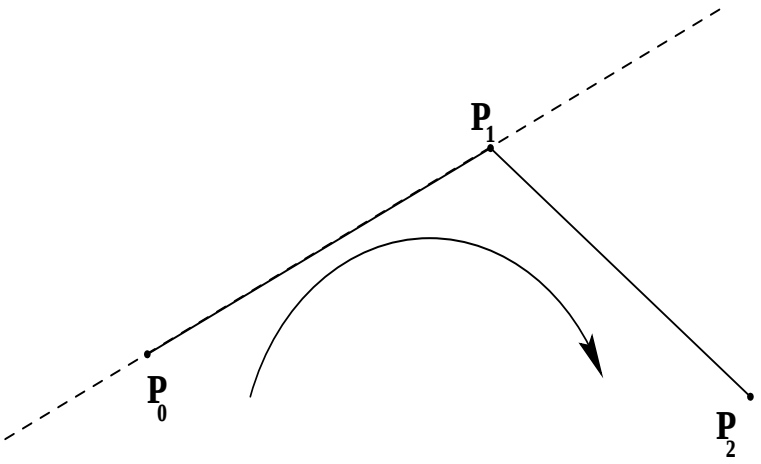
3.4.2 Orjentacija konveksnog mnogougla

Već smo definisali mnogougao kao uređenu n-torku njegovih temena $\{p_i \equiv (x_i, y_i) \mid i = 0, 1, ..., n-1\}$. Ta temena mogu niti zadana u direktnom (pozitivnom) ili retrogradnom(negativnom) smeru. Obzirom da je mnogougao konveksan, na osnovu dve uzastopne stranice, odn. tri uzastopna temena, može da se zaključi kakva je orjentacija poligona (vidi slike 40 i 40). Analitički oblik jednačine prave od tačke p_0 do p_1 je:

$$f_0(x, y) \equiv ay - bx - c \tag{3.41}$$



Slika 3.12: Deo direktno orjentisanog mnogougla



Slika 3.13: Deo retrogradno orjentisanog mnogougla

gde su

$$\begin{aligned} a &= x_1 - x_0 \\ b &= y_1 - y_0 \\ c &= ay_1 - bx_1 \end{aligned} \tag{3.42}$$

kao što smo već videli u poglavlju 3.1.1 na strani 17. Ako je $f_0(x, y) > 0$, tačka (x, y) leži levo od linije $\overline{p_0, p_1}$, ako je $f_0(x, y) < 0$, onda je (x, y) sa desne strane, a ukoliko je $f_0(x, y) = 0$, onda tačka (x, y) leži na pravoj $\overline{p_0, p_1}$. Sa slika 3.12 i 3.13 se lako zaključuje da je dovoljno izračunati $f_0(x_2, y_2)$, da se odredi sa koje strane leži tačka p_2. Ako je tačka p_2 levo od $\overline{p_0, p_1}$, mnogougao je zadan u direktnom smeru, a ako je p_2 desno, u retrogradnom. Za detalje implementacije pogledajte funkciju ORIENTATION2D() iz fajla GRAPHUTILITIES.H / .C (dodatak A na strani 45).

3.4.3 Presek dva konveksna mnogougla

Presek dva konveksna mnogougla A i B može biti ili prazan skup ili novi konveksan mnogougao C. Pretpostavimo da su oba poligona orjentisana u direktnom smeru. Mnogougao C se dobija od mnogougla A „odsecanjem" svih delova koji su van mnogougla B. Zbog toga ću prvo pretpostaviti da je $C = A$, a onda u svakom koraku seći sve što u „trenutnom" mnogouglu C viri van aktivne stranice mnogougla B.

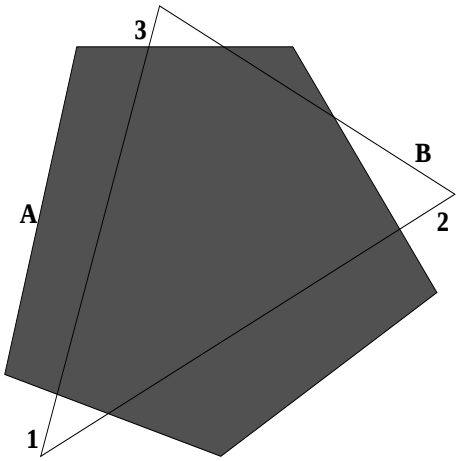
Mnogouglovi A i B zadati su brojem temena $numA$ i $numB$, redom, i nizom koordinata svojih temena $A[0..numA - 1]$ i $B[0..numB - 1]$. Za potrebe „sečenja" ću da napravim dva privremena mnogougla $f[0]$ i $f[1]$, i dva brojača $l1(= 0)$ i $l2(= 1)$. Pošto je možda i ceo mnogougao A unutar mnogougla B za početak je $f[l1] = A$. U stvari $f[l1][0..numC - 1]$ predstavlja mnogougao C, a $f[l2][0..numC\prime - 1]$ mnogougao $C\prime$. Mnogougao $C\prime$ se dobija „sečenjem" mnogougla C o stranicu mnogougla B. Na kraju se vrednosti $l1$ i $l2$ razmene i „seče" ponovo za novu stranicu mnogougla B. Sam proces „sečenja" odvija se slično kao određivanje položaja tačke iz prethodnog poglavlja. Analitički oblik linije o koju se „seče", od tačke $B[i]$ do $B[j]$ je

$$f_i \equiv ay - bx - c \tag{3.43}$$

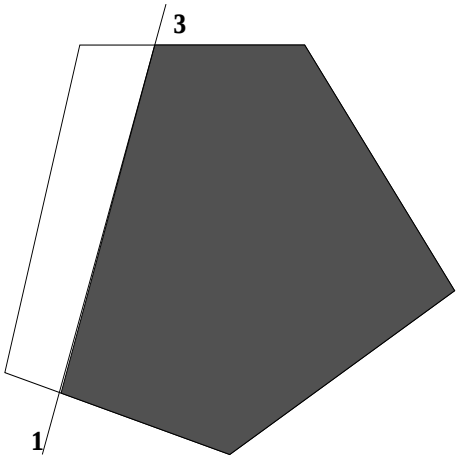
gde su

$$\begin{aligned} a &= B[j].x - B[i].x \\ b &= B[j].y - B[i].y \\ c &= a \cdot B[i].y - b \cdot B[i].x \end{aligned} \tag{3.44}$$

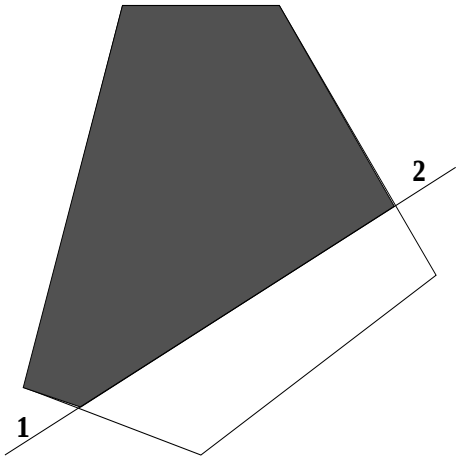
Pogledajmo liniju od temena $v_k = f[l1][k]$ do $v_l = f[l1][l]$. Ako je $f_i(v_k) \geq 0$, onda v_k može da se ubaci u $f[l2]$. Ako tačke v_k i v_l leže na suprotnim stranama linije $(B[i], B[j])$, mora da se nađe tačka preseka (pogledaj poglavlje 3.1.1 na strani 17). Celokupan proces „sečenja" se bolje razume sa slika 3.14, 3.15, 3.16 i 3.17.Ovaj algoritam može da se primeni i za sečenje poligona o stranice ekrana o čemu je bilo reči u poglavlju 3.4.1 naravno, samo u kontekstu linija. Za dalje informacije o implementaciji, pogledajte funkciju OVERLAP2D() iz fajla GRAPHUTILITIES.H / .C (dodatak A na strani 45).



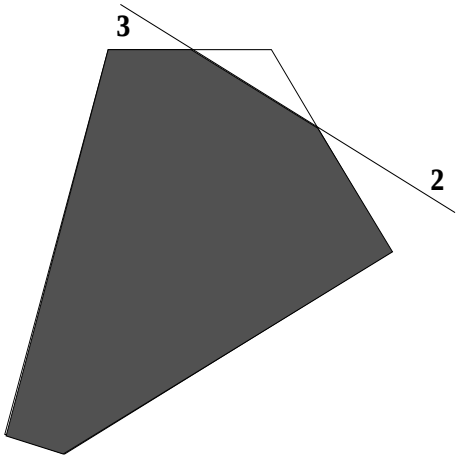
Slika 3.14: Mnogouglovi A i B (osenčen je „trenutni presek”)



Slika 3.15: Prvi korak: Sečenje o stranicu 3-1 mnogougla B



Slika 3.16: Drugi korak: Sečenje o stranicu 1-2 mnogougla B



Slika 3.17: Treći korak: Sečenje o stranicu 2-3 mnogougla B

Dodatak A

Fajl GRAPHUTILITIES

Tokom celog rada spominjem funkcije iz fajla GRAPHUTILITIES.H / .C, pa evo sada dajem celokupan listing tih fajlova

A.1 Definicija funkcija - GRAPHUTILITIES.H

```
/* Fajl: <graphUtilities.h> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#ifndef __GRAPH_UTILITIES_H
#define __GRAPH_UTILITIES_H

#include "window2vport.h"

/* lineIntersect2D - presek dve prave u dvodimenzionalnom prostoru
 *    IN : v1,v2   - prava p1: v1 + μ * v2
 *    IN : v3,v4   - prava p2: v3 + μ * v4
 *    OUT: v        - v = p1 ∩ p2
 *    RETURN:       - ima li preseka<TRUE/FALSE>
 */
bool lineIntersect2D(struct vector2D v1,
                         struct vector2D v2,
                         struct vector2D v3,
                         struct vector2D v4,
                         struct vector2D *v);

/* angle - ugao izmedju x-ose i vektora p(x,y)
*    IN : x,y      - x i y koordinate vektora p
*    RETURN: gore navedeni ugao
 */
float angle(float x,float y);

/* Znak promenljive */
```

```
int sign(float r);
/* Orjentacija konveksnog mnogougla:
 *   Argumenti su tri uzastopna temena
 *   -1 - u smeru kazaljke na satu (retrogradno)
 *   +1 - u matematičkom smeru (direktno)
 *    0 - degeneracija - linija ili tačka
 */
int orientation2D(struct vector2D p0, /* Prva tačka */
                  struct vector2D p1, /* Druga tačka */
                  struct vector2D p2); /* Treća tačka */
/* Presek dva mnogougla:
 * mnogouglovi moraju da budu orjentisani u istom smeru
 * vraća false ako nema preseka
 */
bool overlap2D(int numA,struct vector2D A[], /* Mnogougao A */
                  int numB,struct vector2D B[], /* Mnogougao B */
                  int *numC,struct vector2D C[]); /* Mnogougao C = A presek B */

#endif /* __GRAPH_UTILITIES_H */
/* Kraj fajla <graphUtilities.h> */
```

A.2 Implementacija funkcija - GRAPHUTILITIES.C

```
/* Fajl: <graphUtilities.c> */
/*
 * Projekat: Osnovi računarske grafike
 * Organizacija: Matematička gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Brčić IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#include "graphUtilities.h"

/* lineIntersect2D - presek dve prave u dvodimenzionalnom prostoru
 *    IN : v1,v2   - prava p1: v1 + μ * v2
 *    IN : v3,v4   - prava p2: v3 + μ * v4
 *    OUT: v        - v = p1 ∩ p2
 *    OUT: flag    - ima li preseka<TRUE/FALSE>
 */
bool lineIntersect2D(struct vector2D v1,
                     struct vector2D v2,
                     struct vector2D v3,
                     struct vector2D v4,
                     struct vector2D *v){
    float mu,delta;

    assert(v!=NULL);
    delta = v2.x*v4.y-v2.y*v4.x;
    if(fabs(delta) < epsilon)
       return false;
    mu = ((v3.x-v1.x)*v4.y-(v3.y-v1.y)*v4.x)/delta;
    v->x = v1.x+mu*v2.x;
```

```
    v->y = v1.y+mu*v2.y;
    return true;
}

/* angle - ugao izmedju x-ose i vektora p(x,y)
*    IN : x,y     - x i y koordinate vektora p
*    RETURN: gore navedeni ugao
 */
float angle(float x,float y){
    if(fabs(x) < epsilon){
        if(fabs(y) < epsilon)
       return 0.0;
        else if(y > 0.0)
       return pi*0.5;
        else return pi*1.5;
    }else if(x < 0.0)
       return atan(y/x)+pi;
    else return atan(y/x);
}

/* Znak promenljive */
int sign(float r){
    if(r > epsilon)
       return 1;
    else if(r < -epsilon)
       return -1;
    else
       return 0;
}

/* Orjentacija mnogougla:
 * Argumenti su tri uzastopna temena
 *   -1 - u smeru kazaljke na satu (retrogradno)
 *   +1 - u matematičkom smeru (direktno)
 *    0 - degeneracija - linija ili tačka
 */
int orientation2D(struct vector2D p0, /* Prva tačka */
                  struct vector2D p1, /* Druga tačka */
                  struct vector2D p2){ /* Treća tačka */
    struct vector2D d1,d2;
    d1.x = p1.x-p0.x; d1.y = p1.y-p0.y;
    d2.x = p2.x-p1.x; d2.y = p2.y-p1.y;
    return sign(d1.x*d2.y - d1.y*d2.x);
}

/* Presek dva mnogougla:
 * mnogouglovi moraju da budu orjentisani u istom smeru
 * vraća false ako nema preseka
 */
bool overlap2D(int numA,struct vector2D A[], /* Mnogougao A */
                  int numB,struct vector2D B[], /* Mnogougao B */
                  int *numC,struct vector2D C[]){ /* Mnogougao C = A presek B */
    int i,j,index1,index2,l1,l2,numCdash,orientation;
    struct vector2D f[2][maxPoly],end1,end2,v1,v2;
```

```
float ca,cb,cc,fv1,fv2,absFv1,absFv2,delta;

orientation = orientation2D(A[0],A[1],A[2]);
i = orientation2D(B[0],B[1],B[2]);
assert(orientation == i); /* Ista orjentacija */
assert(orientation != 0); /* Nema degeneracije */
assert(numA <= maxPoly); /* Mnogouglovi A i B */
assert(numB <= maxPoly); /* odgovaraju specifikaciji */

/* Pretpostavimo da je ceo A unutar B */
l1 = 0;
*numC = numA;
for(i=0;i<*numC;i++)
   f[l1][i] = A[i];
/* Seci Miško - odseca sve van aktivne stranice mnogougla B*/
end1 = B[numB-1]; /* Prvo od poslednje do prve tačke */
for(i=0;i<numB;i++){
    /* Sve isečeno ide u f[l2][...] */
    l2 = 1-l1;
    end2 = B[i];
    /* Izračunaj f-ju stranice f(x,y):= ca*y+cb*x+cc */
    ca = end2.x-end1.x;
    cb = end1.y-end2.y;
    cc = -end1.x*cb - end1.y*ca;
    /* Idemo sad kroz f[l1][...] stranu po stranu */
    v1 = f[l1][*numC-1];
    /* f(v1) = ?, gde je v1 u odnosu na aktivnu stranicu (end1,end2) */
    fv1 = ca*v1.y + cb*v1.x + cc;
    absFv1 = fabs(fv1);
    if(absFv1 < epsilon)
   index1 = 0;
    else{
    index1 = sign(fv1)*orientation;
    numCdash = 0;
    for(j=0;j<*numC;j++){
        v2 = f[l1][j];
        /* f(v2) = ?, gde je v2 u odnosu na aktivnu stranicu (end1,end2) */
        fv2 = ca*v2.y + cb*v2.x + cc;
        absFv2 = fabs(fv2);
        if(absFv2 < epsilon)
           index2 = 0;
        else index2 = sign(fv2)*orientation;
        /* Ako je v1 unutar ili na stranici (end1,end2), uključi je */
        if(index1 >= 0)
           f[l2][numCdash++] = v1;
        /* Ako su v1 i v2 na suprotnim stranama, nađi presek */
        if((index1 != 0) && (index1 != index2) && (index2 != 0)){
            delta = absFv1 + absFv2;
            f[l2][numCdash].x = (absFv2*v1.x + absFv1*v2.x)/delta;
            f[l2][numCdash].y = (absFv2*v1.y + absFv1*v2.y)/delta;
            numCdash++;
        }
        /* Idemo dalje */
        fv1 = fv2;
```

```
            absFv1 = absFv2;
            index1 = index2;
            v1 = v2;
        }
        if(numCdash < 3){
            /* Nema preseka */
            *numC = 0;
            return false;
        }else{
            /* Idemo na novu stranicu mnogougla B */
            *numC = numCdash;
            l1 = l2;
            end1 = end2;
        }
        }
    }
    /* Ubaci f[l1][...] gde treba */
    for(i=0;i<*numC;i++)
       C[i] = f[l1][i];
    return true;
}

/* Kraj fajla <graphUtilities.c> */
```

Dodatak B

Jedna moguća implementacija *primitiva*

Red je da pored definicije funkcija iz PRIMITIVES.H koje predstavljaju osnov za sve dalje, dam i neku njihovu moguću implementaciju. Ja sam se opredelio za biblioteku Xlib koja je deo X Window System-a i dakle predstavlja neku vrstu standarda. Zbog manjka literature za Xlib (nemam ništa od dotične), ne garantujem da bilo šta od sledećih funkcija stvarno i radi. Kod mene na RedHat Linux-u 5.0 (Hurricane) sa kernel-om 2.0.35, Xlib-om 3.10 i X Window System-om verzija 11, release 6.3, radi. Ako neko ima nešto od literature za Xlib, bio bih zahvalan ako bih istu mogao da dobijem da iskopiram.

```
/* Fajl: <primitives.c> */
/*
 * Projekat: Osnovi ra\v cunarske grafike
 * Organizacija: Matemati\v cka gimnazija u Beogradu
 * Datum: 1999/02/18
 * Autor: Filip S. Br\v ci\' c IVc
 * Email: brcha@geocities.com
 * URL: http://www.geocities.com/SiliconValley/Campus/3732/Grafika.html
 */
#include "primitives.h"
#include <X11/Xlib.h>
#include <X11/Xutil.h>

/* Setup Variables */
char myProgramName[100] = "Fusion Graphics for X Window System";
char *myDisplayName = NULL;
int myWindowGeometry_x=0;
int myWindowGeometry_y=0;
int myWindowGeometry_width=500;
int myWindowGeometry_height=500;

Display *myDisplay;
Screen *myScreen;
GC myGC;
Window myXWindow;
Colormap myColorMap;
int blackColor, whiteColor;
XEvent myEvent;
unsigned int myLineWidth = 1;
```

```
int myLineStyle = LineSolid;
int myLineCapStyle = CapButt;
int myLineJoinStyle = JoinMiter;
unsigned long *ToX;
unsigned long myEventMask;
XEvent myKeyEvent;
XEvent myExposeEvent;

void beginGraphics(void){
    int i,r,g,b;
    XWindowAttributes xgwa;
    XSetWindowAttributes xswa;
    XEvent xev;
    int forked;

    /* Init Defaults */
    nXpixels = myWindowGeometry_width;
    nYpixels = myWindowGeometry_height;

    /* Insure Connection */
    myDisplay = XOpenDisplay(myDisplayName);
    myScreen = DefaultScreenOfDisplay(myDisplay);
    myColorMap = DefaultColormap(myDisplay,DefaultScreen(myDisplay));
    blackColor = BlackPixel(myDisplay,DefaultScreen(myDisplay));
    whiteColor = WhitePixel(myDisplay,DefaultScreen(myDisplay));
    XSelectInput(myDisplay,DefaultRootWindow(myDisplay),0);

    /* Create a window */
    numColors = DisplayCells(myDisplay,DefaultScreen(myDisplay));
    if(numColors < 256)
       numColors = 256;
    ToX    = (unsigned long*) safeCalloc(numColors,sizeof(unsigned long));
    red    = (float*) safeCalloc(numColors,sizeof(float));
    green = (float*) safeCalloc(numColors,sizeof(float));
    blue   = (float*) safeCalloc(numColors,sizeof(float));
    /* Initialize */
    for(i=0;i<numColors;i++)
       ToX[i] = i;

    i=0;
    for(b=0;b<=1;b++)
       for(g=0;g<=1;g++)
          for(r=0;r<=1;r++)
           setrgb(i++,r,g,b);

    xswa.event_mask = StructureNotifyMask | ExposureMask;
    xswa.background_pixel = ToX[0];
    xswa.border_pixel = ToX[1];
    myEventMask = (StructureNotifyMask |
                ExposureMask |
                KeyPressMask |
                ButtonPressMask | ButtonReleaseMask | ButtonMotionMask);
    xswa.do_not_propagate_mask = myEventMask &
       (KeyPressMask | KeyReleaseMask | ButtonPressMask |
```

```
        ButtonReleaseMask | PointerMotionMask |
        Button1MotionMask | Button2MotionMask | Button3MotionMask |
        Button5MotionMask | ButtonMotionMask);
    xswa.colormap = myColorMap;
    myXWindow = XCreateWindow(myDisplay,DefaultRootWindow(myDisplay),
                              myWindowGeometry_x,myWindowGeometry_y,
                              myWindowGeometry_width,myWindowGeometry_height,
                              5,
                              DefaultDepth(myDisplay,DefaultScreen(myDisplay)),
                              InputOutput,
                              DefaultVisual(myDisplay,DefaultScreen(myDisplay)),
                              (unsigned long) (CWDontPropagate | CWBackPixel |
                                               CWEventMask | CWColormap),
                              &xswa);
    XStoreName(myDisplay,myXWindow,myProgramName);
    myGC = XCreateGC(myDisplay,myXWindow,(unsigned long)0,(XGCValues*)NULL);
    XMapWindow(myDisplay,myXWindow);
    XWindowEvent(myDisplay,myXWindow,ExposureMask,&xev);
    XSync(myDisplay,0);
    XGetWindowAttributes(myDisplay,myXWindow,&xgwa);
    nXpixels = xgwa.width;
    nYpixels = xgwa.height;
    XSelectInput(myDisplay,myXWindow,myEventMask);
    XSync(myDisplay,0);
    XClearWindow(myDisplay,myXWindow);
    XSync(myDisplay,0);

    setColor(1);
    currentPixel.x = 0;
    currentPixel.y = 0;
}

void updateGraphics(void){
    XWindowAttributes xgwa;

    assert(myDisplay!=NULL);
    XGetWindowAttributes(myDisplay,myXWindow,&xgwa);
    nXpixels = xgwa.width;
    nYpixels = xgwa.height;
}

void endGraphics(void){
    assert(myDisplay!=NULL);
    XFreeGC(myDisplay,myGC);
    XFreeColormap(myDisplay,myColorMap);
    XUnmapWindow(myDisplay,myXWindow);
    XDestroyWindow(myDisplay,myXWindow);
    XCloseDisplay(myDisplay);
    safeFree(blue);
    safeFree(green);
    safeFree(red);
    safeFree(ToX);
}
```

```
void clearDisplay(void){
    assert(myDisplay!=NULL);
    XClearWindow(myDisplay,myXWindow);
}

void setColor(int color){
    if(color > numColors)
       return;
    currentColor = color;
}

void setPixel(struct pixelVector pixel){
    assert(myDisplay!=NULL);
    movePixel(pixel);
    XSetForeground(myDisplay,myGC,ToX[currentColor]);
    XDrawPoint(myDisplay,myXWindow,myGC,pixel.x,pixel.y);
    XFlush(myDisplay);
}

void movePixel(struct pixelVector pixel){
    currentPixel = pixel;
}

void linePixel(struct pixelVector pixel){
    assert(myDisplay!=NULL);
    XSetLineAttributes(myDisplay,myGC,myLineWidth,myLineStyle,
                       myLineCapStyle,myLineJoinStyle);
    XSetForeground(myDisplay,myGC,ToX[currentColor]);
    XDrawLine(myDisplay,myXWindow,myGC,
              currentPixel.x,currentPixel.y,
              pixel.x,pixel.y);
    XFlush(myDisplay);
    movePixel(pixel);
}

void polyPixel(int n,struct pixelVector poly[]){
    int i;
    XPoint *points;

    assert(myDisplay!=NULL);
    points = (XPoint*)safeMalloc((unsigned)n*sizeof(XPoint));
    for(i=0;i<n;i++){
        points[i].x = poly[i].x;
        points[i].y = poly[i].y;
    }
    XSetFillStyle(myDisplay,myGC,FillSolid);
    XSetLineAttributes(myDisplay,myGC,myLineWidth,myLineStyle,
                       myLineCapStyle,myLineJoinStyle);
    XSetForeground(myDisplay,myGC,ToX[currentColor]);
    XFillPolygon(myDisplay,myXWindow,myGC,points,n,Complex,CoordModeOrigin);
    safeFree(points);
    XFlush(myDisplay);
}
```

```
void setrgb(int i,float r,float g,float b){
    XColor newColor;

    assert(myDisplay!=NULL);
    red[i] = r;
    green[i] = g;
    blue[i] = b;
    newColor.red   = (int)(65535*r);
    newColor.green = (int)(65535*g);
    newColor.blue  = (int)(65535*b);
    XAllocColor(myDisplay,myColorMap,&newColor);
    ToX[i] = newColor.pixel;
}

void flip(void){
}

/*
 * lineStyle ::= { LineSolid | LineOnOffDash | LineDoubleDash }
 * capStyle ::= { CapNotLast | CapButt | CapRound | CapProjecting }
 * joinStyle ::= {JoinMiter | JoinRound | JoinBevel }
 */
void setLineStyle(unsigned int width,int lineStyle,int capStyle,int joinStyle){
    myLineWidth = width;
    myLineStyle = lineStyle;
    myLineCapStyle = capStyle;
    myLineJoinStyle = capStyle;
}

/*
 * lineType ::= { SET(0) | OR(1) | AND(2) | XOR(3) }
 */
void setLineType(int type){
    XGCValues xgcval;
    assert(myDisplay!=NULL);
    switch(type){
     case 0:
        /* SET */
        xgcval.function = GXcopy;
        break;
     case 1:
        /* OR */
        xgcval.function = GXor;
        break;
     case 2:
        /* AND */
        xgcval.function = GXand;
        break;
     case 3:
        /* XOR */
        xgcval.function = GXxor;
        break;
    }
    XChangeGC(myDisplay,myGC,GCFunction,&xgcval);
```

```
}

bool keyPressed(void){
    assert(myDisplay!=NULL);
    return XCheckWindowEvent(myDisplay,myXWindow,KeyPressMask,&myKeyEvent);
}

bool displayExposed(void){
    assert(myDisplay!=NULL);
    return XCheckWindowEvent(myDisplay,myXWindow,ExposureMask,&myExposeEvent);
}

int getChar(void){
    char c;
    int keyBufSize = 1;
    char keyBuf[2];
    XComposeStatus compStat;
    KeySym keySym;

    assert(myDisplay!=NULL);
    if(!keyPressed())
       return -1;
    XLookupString(&(myKeyEvent.xkey),keyBuf,keyBufSize,&keySym,&compStat);
    return keySym;
}

/* Kraj fajla <primitives.c> */
```