

**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

# ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Студент Журавлев Н.В.

Группа РК6-42Б

Тип задания Лабораторная работа №4

Студент

**Журавлев Н.В.**

*подпись, дата фамилия, и.о.*

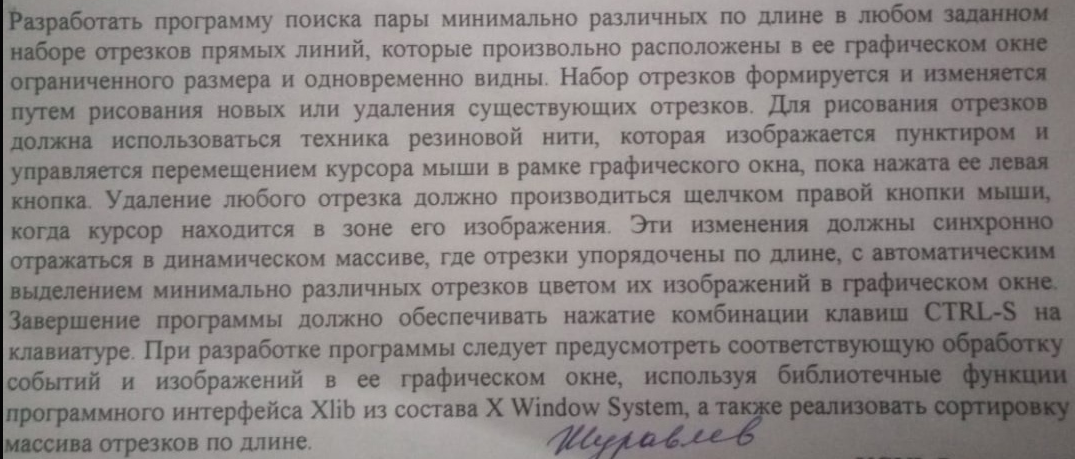
Преподаватель

**Волосатова Т.М.**

*подпись, дата фамилия, и.о.*

*Москва, 2021*

**Текст задания**

**Описание алгоритма**

Функция main поочередно вызывает функции: resmng, gcing, canvas, pass1, dispatch.

resmng - управление цветными ресурсами. Создает базу цветных ресурсов программы по спецификациям в ресурсном файле .XSort, аргументах командной строки или из значений по умолчанию в своем статическом массиве. В любом случае спецификация цветов задается в Х-формате. Цвета из базы ресурсов распределяются в цветовую карту палитру (или идентифицируются в ней) по умолчанию. Затем создается массив графических контекстов для распределенных цветов палитры в поле foreground их структуры и с одинаковым фоном в поле background.

gcing - корректировка граф. контекстов. Установка тонкого пунктира с функцией Xor для резинового граф. контекста. Для остальных контекстов устанавливается двойная сплошная линия с функцией copy.

canvas - создание основного граф. окна с начальным размером 640 х 480. Его фон должен совпадать с фоном в массиве графических контекстов. Также задается маска событий, которые будут обрабатываться диспетчером dispatch.

pass1 - передача граф. параметров из управляющего модуля через свои аргументы соответствующих адресных типов.

dispatch - диспетчер событий. Чтение очереди событий для графокна программы в основном для их резиновой обработки.

Dispatch, в свою очередь вызывает следующие функции:

expo - перерисовка изображений геофигур в граф. окне вызовом функции refrag из резинового модуля (xsort1). Минимизацию перерисовок обеспечивает техника отсечения. Функция вызывается по событию Expose диспетчером dispatch при любой потере изображения в графокне программы.

rubin – идентификация базовой точки резинового редактирования по координатам мыши вызовом фигурной функции frag0, если была нажата левая кнопка мыши. В любом случае обеспечивает сохранение BAK-копии экстра-фигур.

rubout – удаление геофигуры по указателю мыши при нажатии правой и средней кнопки. Изображение стирается вызовом фигурных функций XFix или XExtra для обычной и экстра фигуры. Для исключения геофигуры из массива frag вызывает функцию realloc.

fix – фиксирует контур резинового шаблона в графокне программы и добавляет его в массив геофигур.

rextra – комплексная прикладная функция для вызова экстра-метода с требуемым критерием выбора экстра-фигуры. В коде этой функции адреса всех выше перечисленных экстра-методов упорядочены в массив указателей на их функции. Он передается фигурной функции fragextra, которая обеспечивает вызов требуемого экстра-метода по его макроиндексу. Исходный текст функции rextra завершают вызов(ы) резиновой функции purgextr (из xsort1) для очистки областей изображения одной (или пары) экстра-фигур(ы) до и после перевыборов.

**Описание входных данных**

На вход аргументом командной строки передается путь к ресурсному файлу

**Описание выходных данных**

На выходе создается окно с фигурами и со всеми функциональными возможностями, согласно ТЗ.

**Текст программы**

xsort.h

/\* Rubber Sort header file \*/

/\* Union Fragment types structure \*/

typedef union {

XSegment seg[1]; /\* line segment \*/

XRectangle rec[1]; /\* (fill) rectangle \*/

XArc arc[1]; /\* (fill) ellipse \*/

} XFragment;

/\* Color GC ground index \*/

#define NGC 5 /\* GCs' number \*/

#define FGC 0 /\* fore-ground \*/

#define RGC 1 /\* rubber-ground \*/

#define EGC 2 /\* extr-ground \*/

#define AGC 3 /\* groupalt-ground \*/

#define BGC 4 /\* back-ground color \*/

/\* Extra function index \*/

#define MINIFRAG 1 /\* min (Left sort) fragment \*/

#define MAXIFRAG 0 /\* max (Right sort) fragment \*/

#define MEANFRAG 4 /\* mean (Medium sort) fragment \*/

#define DIFFFRAG 4 /\* 2 max differ (Left & Right) fragments \*/

#define IDENFRAG 3 /\* 2 max identical (beside sort) fragments \*/

#define GRP2FRAG 5 /\* devide to 2 group fragments \*/

/\* Rubber functions xsort1 \*/

int pass1(Display\*, Window, GC\*); /\* pass graphic parameters \*/

int photo(); /\* photo flex parameters to pass xsort0 \*/

int rebak(); /\* to baking extra fragments \*/

int rubin(XEvent\*); /\* new rubber or del fragment \*/

int rubout(XEvent\*); /\* rubout(del) fragment \*/

int rerub(XEvent\*); /\* deformate rubber fragment \*/

int fix(XEvent\*); /\* fix rubber fragment \*/

int miniwin(); /\* hint min window size to WM \*/

int fragcmp(const void\*, const void\*); /\* compare 2 fragments \*/

int refrag(); /\* redraw all fragments \*/

int near(int, int); /\* find near fragment to xy point \*/

int cancet(); /\* cancel template fragment \*/

int purgextr(XFragment\*, XFragment\*); /\* purge extras space \*/

int reggc(int); /\* reset alter GC for group extra \*/

int allfree();

/\* Extra types xsort0 \*/

int pass0(void\*\*); /\* pass flex parameters list from xsort1 \*/

int miniextra(); /\* mini extra fragment \*/

int maxiextra(); /\* maxi extra fragment \*/

int meanextra(); /\* mean extra fragment \*/

int diffextra(); /\* 2 max differ extra fragments \*/

int idenextra(); /\* 2 max similar extra fragments \*/

int grp2extra(); /\* devide 2 group extra fragments \*/

int rextra(int); /\* reset extra fragment \*/

int isextra(int); /\* check extra fragment index \*/

/\* Fragment functions \*/

int fragon(XFragment\*, int, int, int); /\* check hit fragment \*/

int difrag(XFragment\*, XFragment\*); /\* 2 fragment difference \*/

int frag0(XFragment\*, int, int); /\* set fragment xy-origin \*/

int fragvar(XFragment\*, int, int); /\* variate fragment size \*/

int fragmaxix(XFragment\*); /\* fragment's max x point \*/

int fragmaxiy(XFragment\*); /\* fragment's max y point \*/

int tinyfrag(XFragment\*); /\* tiny fragment test \*/

int fragextra(int (\*[])()); /\* call fragment extra method \*/

/\* Draw Fragment Functions \*/

int XFixes(Display\*, Window, GC, XFragment\*, int);

int XContour(Display\*, Window, GC, XFragment\*);

int XExtra(Display\*, Window, GC, XFragment\*);

int XFix(Display\*, Window, GC, XFragment\*);

/\* Resource & dispatch functions xsort2 \*/

int resmng(int, char\*[]); /\* rgb resource managment \*/

int canvas(); /\* main window \*/

int gcing(); /\* custom GC drawing \*/

int dispatch(); /\* dispatch event \*/

int expo(XEvent\*); /\* expose fragments window \*/

int rekey(XEvent\*); /\* key press reaction \*/

xseg.c

/\* RubberSort & Search: Oval functions overload \*/

#include <X11/Xlib.h>

#include "xsort.h"

#include <math.h>

#include <stdlib.h>

/\* figure type set {XSegment, XRectangle, XArc} \*/

typedef XSegment XFig; /\* set figure type for oval \*/

/\* figure type macro converter from (XFragment\* ) \*/

#define REFIG(F) (F->seg) /\* address seg, rec or arc \*/

/\* Check (x,y) hited fragment \*/

int fragon(XFragment\* ff, int x, int y, int t) {

XFig \* f = REFIG(ff);

float x1 = f->x1;

float x2 = f->x2;

float y1 = f->y1;

float y2 = f->y2;

if (((( y - 10 <= ((x - x1)/(x2 - x1)) \* (y2 - y1) + y1) &&

( y + 10 >= ((x - x1)/(x2 - x1)) \* (y2 - y1) + y1)) ||

(( x - 10 <= ((y - y1)/(y2 - y1)) \* (x2 - x1) + x1) &&

( x + 10 >= ((y - y1)/(y2 - y1)) \* (x2 - x1) + x1)))

)

return 1;

else

return 0;

} /\* fragon \*/

/\* Overlap 2 fragmentes \*/

/\* 2 Fragment difference for qsort\*/

int difrag(XFragment\* ff1, XFragment\* ff2) {

XFig\* f1 = REFIG(ff1);

XFig\* f2 = REFIG(ff2);

return(pow(abs(f1->x2 - f1->x1),2) + pow(abs(f1->y2 - f1->y1),2) - pow(abs(f2->x2 - f2->x1),2) - pow(abs(f2->y2 - f2->y1),2));} /\* difrag \*/

/\* Addressed Call fragment extra method by MACRO NUMBER \*/

int fragextra(int (\*fe[])()) { /\* set extra method number \*/

return((\*fe[MEANFRAG])()); /\* MINI MAXI MEAN DIFF IDEN GRP2 \*/

} /\* fragextra \*/

/\* Fragment gabarit size \*/

int fragsize(XFragment\* ff) {

XFig\* f = REFIG(ff);

return(abs((f->x2 - f->x1) \* (f->y2 - f->y1)));

} /\* fragsize \*/

/\* Tiny Fragment test \*/

int tinyfrag(XFragment\* ff) {

XFig\* f = REFIG(ff);

if(abs((f->x2 - f->x1) \* (f->y2 - f->y1)) < 64)

return(1);

return(0);

} /\* minifrag \*/

/\* Stick fragment base xy-point \*/

int frag0(XFragment\* ff, int x, int y) {

XFig\* f = REFIG(ff);

f->x1 = f->x2 = x;

f->y1 = f->y2 = y;

return(0);

} /\* frag0 \*/

/\* Variate fragment contour by xy-point \*/

int fragvar(XFragment\* ff, int x, int y) {

XFig\* f = REFIG(ff);

f->x2 = x;

f->y2 = y;

return(0);

} /\* fragvar \*/

/\* Max fragment x-coordinate \*/

int fragmaxix(XFragment\* ff) {

XFig\* f = REFIG(ff);

if ( f->x1 > f->x2 )

return f->x1;

else

return f->x2;

} /\* fragmaxix \*/

/\* Max fragment y-coordinate \*/

int fragmaxiy(XFragment\* ff) {

XFig\* f = REFIG(ff);

if ( f->y1 > f->y2 )

return f->y1;

else

return f->y2;

} /\* fragmaxiy \*/

/\* Draw rubber fragment contour \*/

int XContour(Display\* dpy, Window win, GC gc, XFragment\* ff) {

XFig\* f = REFIG(ff);

XDrawSegments(dpy, win, gc, f, 1);

return(0);

} /\* XContours \*/

/\* Draw 1 Fixed Fragmentes \*/

int XFix(Display\* dpy, Window win, GC gc, XFragment\* ff) {

XFig\* f = REFIG(ff);

XDrawSegments(dpy, win, gc, f, 1); /\* need at any case \*/

return(0);

} /\* XFix \*/

/\* Draw Extra Fragment(s) \*/

int XExtra(Display\* dpy, Window win, GC gc, XFragment\* ff) {

XFig\* f = REFIG(ff);

XDrawSegments(dpy, win, gc, f, 1);

return(0);

} /\* XExtra \*/

xsort0.c

/\* RubberSort: Extra module \*/

#include <values.h>

#include <X11/Xlib.h>

#include "xsort.h"

/\* Rubber parameter passed from rubber module xsort1 \*/

static int\* pnfrag; /\* all fragments' number \*/

static XFragment\* pfrag; /\* fragments' array address \*/

static int\* extra; /\* extra fragment index array \*/

static XFragment\* bak; /\* bak extra fragment \*/

/\* Pass flex parameters from xsort1 \*/

int pass0(void\* p[]) {

bak = (XFragment\* ) p[0];

extra = (int\* ) p[1];

pfrag = (XFragment\* ) p[2];

pnfrag = (int\*) p[3];

return(0);

} /\* pass0 \*/

/\* min (left) fragment extra \*/

int miniextra() {

extra[0] = extra[1] = 0;

return(MINIFRAG);

} /\* miniextra \*/

/\* max (right) fragment extra \*/

int maxiextra() {

extra[0] = extra[1] = (\*pnfrag - 1);

return(MAXIFRAG);

} /\* maxiextra \*/

/\* mean (medium) fragment extra in size order \*/

int meanextra() {

extra[0] = extra[1] = ((\*pnfrag)/2);

return(MEANFRAG);

} /\* meanextra \*/

/\* 2 max differ fragments pair extra \*/

int diffextra() {

extra[0] = 0; extra[1] = \*pnfrag - 1;

return(DIFFFRAG);

} /\* diffextra \*/

/\* 2 max identical (beside) fragments pair extra \*/

int idenextra() {

int i, j; /\* fragment & fragment++ indexes \*/

int d; /\* 2 beside fragments difference \*/

unsigned dmin=MAXINT; /\* min difference or (1<<16)-1 \*/

extra[0] = extra[1] = 0; /\* init extra pair \*/

for(i = 0, j = 1; j < pnfrag[0]; i++, j++) { /\* check by 2 \*/

if((d = difrag(pfrag + j, pfrag + i)) < 0) /\* check abs \*/

d -= d; /\* 2 beside fragments difference \*/

if(d < dmin) { /\* check current difference \*/

extra[0] = i; extra[1] = j; /\* fix max ident pair now \*/

dmin = d; /\* fix min difference now \*/

} /\* if \*/

} /\* for \*/

return(IDENFRAG);

} /\* idenextra \*/

/\* Devide fragments to 2 group by medium \*/

int grp2extra() {

extra[0] = extra[1] = (pnfrag[0] / 2); /\* extra medium fragment \*/

reggc(AGC); /\* fragment (after) group alt foreground \*/

return(GRP2FRAG);

} /\* grp2extra \*/

/\* Reset Extra fragment(s) with clear space for redraw \*/

int rextra(int n /\* to cosmetic repass return \*/ ) {

static int (\*extrafunc[])() = { /\* extra functions array \*/

miniextra, maxiextra, meanextra,

diffextra, idenextra, grp2extra

}; /\* extrafunc \*/

fragextra(extrafunc); /\* reset extra method \*/

purgextr(bak, pfrag+extra[0]); /\* purge 1st bak & new extra space \*/

if(extra[0] != extra[1]) /\* purge 2nd bak & new extra space \*/

purgextr(bak+1, pfrag+extra[1]);

return(0);

} /\* rextra \*/

/\* Check extra fragment by index i \*/

int isextra(int i) {

return((i == extra[0]) || (i == extra[1]));

} /\* isextra \*/

xsort1.c

/\* RubberSort: Rubber flex functions \*/

#include <X11/Xlib.h>

#include <X11/Xutil.h>

#include <string.h>

#include <stdlib.h>

#include "xsort.h"

#define FRAGSIZ sizeof(XFragment)

/\* Graphics parameters \*/

static Display\* dpy; /\* Display address \*/

static Window win; /\* Main Window id \*/

static GC\* gc; /\* GC array address \*/

/\* Geometry parameters \*/

static XFragment ftmp[1]; /\* template (temporary) fragment \*/

static XFragment bak[2]; /\* bak extra fragments \*/

static XFragment\* frag; /\* fragments' set address \*/

static int nfrag=0; /\* all fragments' number \*/

static int extra[2]={0,0}; /\* extra fragment index \*/

static int GGC=FGC; /\* alt (after) ForeGC index (for grp2extr) \*/

/\* Pass graphic parameters from main (xsort2) \*/

int pass1(Display\* d, Window w, GC\* g) {

dpy = d; win = w; gc = g;

return(0);

} /\* pass \*/

/\* Photo flex parametes to pass xsort0 by theare address \*/

int photo() {

static void\* p[4]; /\* passing list: bak, extra, frag, &nfrag \*/

p[0] = bak; p[1] = extra, p[2] = frag; p[3] = &nfrag;

pass0(p); /\* pass p=list to extra module xsort0 \*/

return(nfrag); /\* when 1-st fragment has been drawed \*/

} /\* photo \*/

/\* Redraw n fixed fragmentes \*/

int XFixes(Display\* dpy, Window win , GC gc, XFragment\* f, int n) {

int i; /\* fragment index \*/

for(i = 0; i < n; i++, f++)

XFix(dpy, win, gc, f); /\* redraw i fixed fragment \*/

return(0);

} /\* XFixes \*/

/\* Store extra fragments to bak-copy \*/

int rebak() {

if(nfrag > 0) /\* baking 1-st extra fragment info \*/

memcpy(bak, frag+extra[0], FRAGSIZ);

if(extra[1] > extra[0]) /\* baking 2-nd extra fragment info \*/

memcpy(bak+1, frag+extra[1], FRAGSIZ);

return(0);

} /\* rebak \*/

/\* Press Button1 new rubber fragment origin \*/

int rubin(XEvent\* ev) {

XGrabPointer(dpy, win, False, (ButtonReleaseMask | Button1MotionMask),

GrabModeAsync, GrabModeAsync, win, None, CurrentTime);

rebak(); /\* to backing extra fragments \*/

if(ev->xbutton.button != Button1) /\* to Erase fragment by Button 2 \*/

return(~Button1); /\* return to delete pointed fragment \*/

frag0(ftmp, ev->xbutton.x, ev->xbutton.y); /\* set reform base point \*/

return(Button1); /\* return to rubber deform & resize \*/

} /\* rubin \*/

/\* Find fragment near with (x,y) point to delete by Button 2 or 3 \*/

int near(int x, int y) {

int i; /\* fragment index \*/

for(i = 0; i < nfrag; i++)

if(fragon(frag+i, x, y, isextra(i)) > 0)

break;

return(i); /\* nfrag if miss or near fragment index \*/

} /\* near \*/

/\* Erase Button2 or 3 clicked fragment (draw by bg BGC) \*/

int rubout(XEvent\* ev) {

int i; /\* deleted fragment index \*/

if(nfrag < 1)

return(0); /\* no one fragment to rub out (delete) \*/

if((i = near(ev->xbutton.x, ev->xbutton.y)) == nfrag)

return(nfrag); /\* no delete when far from any fragment \*/

if((i == extra[0]) || (i == extra[1]))

XExtra(dpy, win, gc[BGC], frag+i); /\* rubout extra fragment \*/

XFix(dpy, win, gc[BGC], frag+i); /\* rubout 1 fixed fragment \*/

XFlush(dpy); /\* flush out \*/

if(--nfrag > i) /\* shift fragment list to left \*/

memmove((frag + i), (frag + i + 1), (nfrag - i)\*FRAGSIZ);

frag = realloc(frag, nfrag\*FRAGSIZ); /\* free 1 fragment \*/

if(nfrag == 0)

frag = NULL; /\* empty fragment list \*/

return(nfrag);

} /\* rubout \*/

/\* Deformate Rubber fragment (double draw by rg RGC) \*/

int rerub(XEvent\* ev) {

static int x, y; /\* previous cursor point \*/

XContour(dpy, win, gc[RGC], ftmp);

if(fragvar(ftmp, ev->xmotion.x, ev->xmotion.y) < 0) {

XContour(dpy, win, gc[RGC], ftmp);

return(XWarpPointer(dpy, None, win, 0, 0, 0, 0, x, y));

} /\* when base centered fragment connect top or left border \*/

XContour(dpy, win, gc[RGC], ftmp);

x = ev->xmotion.x; y = ev->xmotion.y; /\* store cursor point \*/

return(0);

} /\* rerub \*/

/\* Reset Alter GC index for fragments group (call grp2extr) \*/

int reggc(int g) {

return(GGC = g);

} /\* regc \*/

/\* Redraw all Fragments in expo or after edit fragment set \*/

int refrag() {

int n; /\* redraw fragments number \*/

if((n = extra[0]) > 0) /\* fix before extra \*/

XFixes(dpy, win, gc[FGC], frag, n);

if((n = (extra[1] - extra[0])) > 1) /\* fix between extras \*/

XFixes(dpy, win, gc[FGC], (frag + extra[0] + 1), n - 1);

if((n = (nfrag - extra[1])) > 1) /\* fix after extra \*/

XFixes(dpy, win, gc[GGC], (frag + extra[1] + 1), n - 1);

if(nfrag > 0) /\* redraw 1-st extra \*/

XExtra(dpy, win, gc[EGC], frag + extra[0]);

if(extra[1] > extra[0]) /\* redraw 2-nd extra \*/

XExtra(dpy, win, gc[EGC], frag + extra[1]);

return(nfrag);

} /\* refrag \*/

/\* Extend window by base centered fragment only \*/

/\* No need for base corner fragment or KDE \*/

int cancet() {

int i; /\* fragment index \*/

if(tinyfrag(ftmp) > 0) /\* escape tiny template \*/

return(1);

return(0); /\* No cancel fragment \*/

} /\* cancet \*/

/\* Fix rubber fragment with all redrawing by fg GC \*/

int fix(XEvent\* ev) {

int w, h; /\* window min width & height for temp fragment \*/

XUngrabPointer(dpy, CurrentTime); /\* uncatch window pointer \*/

if(ev->xbutton.button != Button1) /\* if erase fragment \*/

return(nfrag); /\* then return with no fix \*/

fragvar(ftmp, ev->xbutton.x, ev->xbutton.y); /\* frag ending pos \*/

XContour(dpy, win, gc[RGC], ftmp); /\* erase last rubber \*/

XFlush(dpy); /\* fragment contour \*/

if(cancet() > 0) /\* test size & overlap template fragment \*/

return(0); /\* to cancel fix \*/

frag = realloc(frag, (nfrag + 1)\*FRAGSIZ); /\* append \*/

memcpy((frag+nfrag), ftmp, FRAGSIZ); /\* new fragment \*/

if(++nfrag > 1) /\* quick resort fragments list \*/

qsort(frag, nfrag, FRAGSIZ, fragcmp);

return(nfrag);

} /\* fix \*/

/\* Purge bak extra & space for new extra fragment \*/

int purgextr(XFragment\* b, XFragment\* e) {

if(memcmp(b, e, FRAGSIZ) != 0) {

XExtra(dpy, win, gc[BGC], b); /\* purge bak extra space \*/

XFix(dpy, win, gc[BGC], e); /\* purge new extra space \*/

} /\* if \*/

return(0);

} /\* purgextr \*/

/\* Compare method to 2 fragments by by increase order \*/

/\* typedef int (\*FCMP)(const void\*, const void\*); \*/

int fragcmp(const void\* s1, const void\* s2) {

return(difrag((XFragment\*) s1, (XFragment\*) s2));

} /\* fragcmp \*/

/\* set main window minimum size to WM \*/

int miniwin() {

XSizeHints hint; /\* WM geom hints \*/

int i=0; /\* fragment index \*/

unsigned w=128; /\* window min width \*/

unsigned h=128; /\* window min height \*/

int xm, ym; /\* max x & y coordinate \*/

for(i=0; i < nfrag; i++) {

if((xm = fragmaxix(frag+i)) > w)

w = xm;

if((ym = fragmaxiy(frag+i)) > h)

h = ym;

} /\* for \*/

hint.min\_width = w; hint.min\_height = h;

hint.flags = PMinSize;

XSetNormalHints(dpy, win, &hint);

return(0);

} /\* miniwin \*/

/\* Free all fragmentes \*/

int allfree() {

if(nfrag > 0) /\* delete all fragments' \*/

free(frag); /\* memory space \*/

nfrag = 0; /\* reset fragments' count \*/

frag = NULL; /\* reset empty fragments' set \*/

extra[0] = extra[1] = 0; /\* reset extra index(es) \*/

XClearWindow(dpy, win); /\* Clear window \*/

return(0);

} /\* allfree \*/

xsort2.c

/\* Sort Rubber resource & dispatch main module \*/

#include <X11/Xlib.h>

#include <X11/Xresource.h>

#include <X11/keysym.h>

#include <X11/keysymdef.h>

#include <string.h>

#include <stdio.h>

#include "xsort.h"

static Display\* dpy; /\* Graphic Display \*/

static GC gc[NGC]; /\* all Graphic Context \*/

static Window win; /\* root & main windows id \*/

/\* RGB Resource managment \*/

int resmng(int argc, char\* argv[]) {

int scr; /\* screen number \*/

Window root; /\* screen root window \*/

Colormap cmap; /\* color paletter \*/

XColor rgb, exact; /\* color structure \*/

int i; /\* Resource index \*/

XrmDatabase rdb=NULL; /\* Resource data base \*/

char restype[64]; /\* Resource type space \*/

char\* rtype = restype; /\* resourse type (String) \*/

XrmValue resval; /\* resource value space \*/

XrmValue\* rval = &resval; /\* resource value (size & addr) \*/

static char\* rname[] = { /\* resource spec names \*/

"xsort.foreground", /\* -fg \*/

"xsort.rubbground", /\* -rg \*/

"xsort.extrground", /\* -eg \*/

"xsort.galtground", /\* -ag \*/

"xsort.background" /\* -bg \*/

}; /\* rname \*/

static char\* rdef[] = { /\* defaults color resource \*/

"green", /\* foreground all fragments \*/

"green", /\* rubber fragment color \*/

"red", /\* extra fragment color \*/

"red", /\* group altforeground fragment color \*/

"black" /\* window background \*/

}; /\* rdef in rname order \*/

static XrmOptionDescRec rtab[] = { /\* Resource Command Options \*/

{"-fg ", ".foreground", XrmoptionSepArg, NULL},

{"-rg ", ".rubbground", XrmoptionSepArg, NULL},

{"-eg ", ".extrground", XrmoptionSepArg, NULL},

{"-ag ", ".galtground", XrmoptionSepArg, NULL},

{"-bg ", ".background", XrmoptionSepArg, NULL},

{"-xrm ", NULL, XrmoptionResArg, NULL}

}; /\* rtab[] \*/

/\* Check display defaults \*/

dpy = XOpenDisplay(NULL);

scr = DefaultScreen(dpy);

cmap = DefaultColormap(dpy, scr);

root = DefaultRootWindow(dpy);

/\* Load resourses data base \*/

XrmInitialize();

rdb = XrmGetFileDatabase(".XSort"); /\* rdb <- .XSort \*/

XrmParseCommand(&rdb, rtab, NGC+1, "xsort", &argc, argv); /\* argv -> rdb \*/

/\* Extract resourses from data base to GC \*/

puts("\nColor resources:");

for(i=0; i < NGC; i++) {

if(XrmGetResource(rdb, rname[i], NULL, &rtype, rval) == False)

rval->addr = rdef[i]; /\* set default for missing resource \*/

/\* or append resource to rdb from default \*/

while(XrmGetResource(rdb, rname[i], NULL, &rtype, rval) == False)

XrmPutStringResource(&rdb, rname[i], rdef[i]);

XrmPutResource(&rdb, rname[i], "String", rval);

XrmPutFileDatabase(rdb, "a.XSort");

if(XParseColor(dpy, cmap, rval->addr, &rgb) == 0)

if(XLookupColor(dpy, cmap, (rval->addr = rdef[i]), &rgb, &exact) == 0)

memcpy(&rgb, &exact, sizeof(XColor));

printf("%s(%s): %s\n", rname[i], rtab[i].option, rval->addr); /\* Echo \*/

fflush(stdout);

XAllocColor(dpy, cmap, &rgb);

gc[i] = XCreateGC(dpy, root, 0, 0);

XSetForeground(dpy, gc[i], rgb.pixel);

} /\* for \*/

if(rdb != NULL) { /\* Free RDB space \*/

XrmDestroyDatabase(rdb);

}

/\* Correct background context \*/

for(i=0; i < NGC; i++) /\* Set background is last rgb \*/

XSetBackground(dpy, gc[i], rgb.pixel);

return(0);

} /\* resmng \*/

/\* Correct rubber GC \*/

int gcing() {

XGCValues gval; /\* Graphic Context values structure \*/

unsigned long gmask=GCLineWidth; /\* Graphic Context mask \*/

int i; /\* GC index \*/

gval.line\_width = 2; /\* set bold lines \*/

for(i=0; i < NGC; i++) /\* for all GCs \*/

XChangeGC(dpy, gc[i], gmask, &gval);

gmask = (GCFunction | GCLineWidth | GCLineStyle);

gval.line\_width = 1; /\* set slim, dash & XOR \*/

gval.function = GXxor; /\* for rubber GC \*/

gval.line\_style = LineOnOffDash; /\* worse LineDoubleDash; \*/

XChangeGC(dpy, gc[RGC], gmask, &gval);/\* Change Rubber GC \*/

return(0);

} /\* gcing \*/

/\* Create & display main 640x480 window \*/

int canvas() {

XSetWindowAttributes attr; /\* main window attributes structure \*/

unsigned long amask; /\* attribute mask \*/

unsigned long emask; /\* event mask \*/

Window root; /\* screen root window \*/

XGCValues gval; /\* Graphic Context values structure \*/

XGetGCValues(dpy, gc[BGC], GCBackground, &gval); /\* Get background \*/

attr.background\_pixel = gval.background; /\* from GC for window \*/

attr.override\_redirect = False; /\* with WM support \*/

attr.bit\_gravity = NorthWestGravity; /\* reconfig Anti-blink \*/

/\* or attr.bit\_gravity = StaticGravity; \*/

amask = (CWOverrideRedirect | CWBackPixel | CWBitGravity);

root = DefaultRootWindow(dpy);

win = XCreateWindow(dpy, root, 0, 0, 640, 480, 1, CopyFromParent,

InputOutput, CopyFromParent, amask, &attr);

XStoreName(dpy, win, "xsort"); /\* Window title \*/

emask = (ButtonPressMask | ButtonReleaseMask | Button1MotionMask |

ExposureMask | KeyPressMask);

XSelectInput(dpy, win, emask); /\* Select events' types for dispath \*/

XMapWindow(dpy, win); /\* display window on screen \*/

return(0);

} /\* canvas \*/

/\* Redraw Fragments when Expose with clip window \*/

int expo(XEvent\* ev) {

static XRectangle clip[32]; /\* clip expo-buffer \*/

static int n=0; /\* clib expo-count \*/

clip[n].x = ev->xexpose.x; /\* accumulate exposed \*/

clip[n].y = ev->xexpose.y; /\* rectangles in clip \*/

clip[n].width = ev->xexpose.width; /\* buffer for \*/

clip[n].height = ev->xexpose.height; /\* redrawing \*/

n++; /\* increment clip count \*/

if((ev->xexpose.count > 1) && (n < (32-1))) /\* with no \*/

return(0); /\* redrawing return \*/

XSetClipRectangles(dpy, gc[FGC], 0, 0, clip, n, Unsorted);

XSetClipRectangles(dpy, gc[EGC], 0, 0, clip, n, Unsorted);

refrag(); /\* redraw all fragments \*/

XSetClipMask(dpy, gc[FGC], None); /\* restore fore- & \*/

XSetClipMask(dpy, gc[EGC], None); /\* extra-grount GCs \*/

return(n=0); /\* return with zeroing clip count \*/

} /\* refrag \*/

/\* Key exit or clean driver \*/

int rekey(XEvent \* ev) {

KeySym ks = XLookupKeysym((XKeyEvent \* ) ev, 1);

if((ks == XK\_S) && (ev->xkey.state == ControlMask))

return('S'); /\* CTRL-S to return exit-code \*/

if (ks != XK\_Delete) /\* Nothing to do if any key except ESC \*/

return (0);

allfree(); /\* Escape all fragments \*/

return (0);

} /\* rekey \*/

/\* Dispatch event function \*/

int dispatch() {

int done = 0; /\* event loop done flag (= false) \*/

XEvent event; /\* graphic event structure \*/

while(done == 0) { /\* event loop \*/

XNextEvent(dpy, &event); /\* Read next event \*/

switch(event.type) { /\* check event type \*/

case Expose: expo(&event); /\* redraw all exposed fragments \*/

break;

case ButtonPress: /\* begin new rubber or delete fragment \*/

if(rubin(&event) != Button1) /\* new fragment \*/

rubout(&event); /\* delete any fragment \*/

break;

case MotionNotify: rerub(&event); /\* reform rubber fragment \*/

break;

case ButtonRelease: /\* end rubber fragment \*/

if(fix(&event) > 0) /\* fix new fragment \*/

rextra(photo()); /\* reset extra fragment \*/

refrag(); /\* redraw all fragments \*/

miniwin(); /\* min window size for WM \*/

break;

case KeyPress: done = rekey(&event); /\* Check quit or clear \*/

break;

default: break;

} /\* swith \*/

} /\* while \*/

allfree();

XDestroyWindow(dpy, win); /\* Close main window \*/

XCloseDisplay(dpy); /\* Disconnect X-server \*/

return(0);

} /\* dispatch \*/

/\* main function \*/

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

resmng(argc, argv); /\* manage color resources \*/

gcing(); /\* set all GC \*/

canvas(); /\* create main window \*/

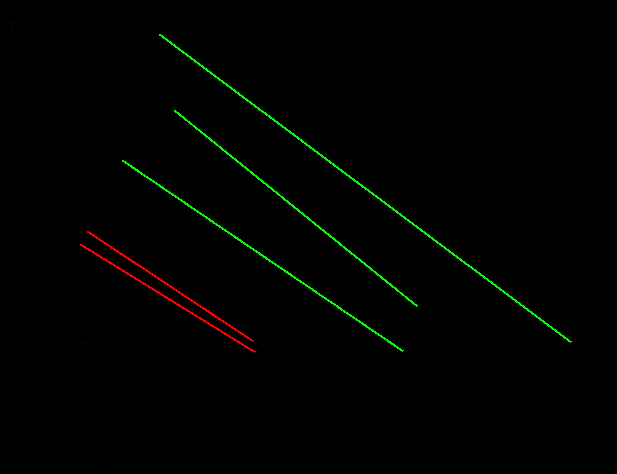
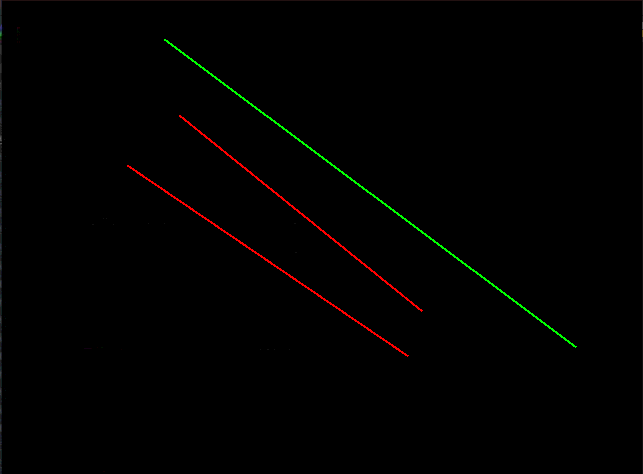
pass1(dpy, win, gc); /\* pass graphics to xsort1 \*/

dispatch(); /\* dispath events \*/

return(0);

} /\* main \*/

**Тестирование**



**Список литературы**

1. Материалы учебного плана по предмету «Программирование графических приложений»

2. Волосатова Т.М., Родионов С. В., Беломойцев Д.Е. «Разработка графических приложений в среде X Window System»

3. Документация по библиотеке Xlib https://tronche.com/gui/x/xlib/ http://www.asvcorp.ru/tech/linux/xwinprg