Министерство образования и науки Российской Федерации
Московский Государственный Технический Университет (МГТУ)
имени Н. Э. Баумана
Факультет «Робототехника и комплексная автоматизация (РК)»
Кафедра «Системы автоматизированного проектирования (РК6)»

Отчет по лабораторной работе №4

По курсу «Программирование графических приложений»

Выполнил: Студент Гусаров Аркадий Андреевич
Группа РК6-43Б

Проверил: _____
Дата _____
Подпись _____

Москва, 2021 г.

Задание

XGUI. Вариант 5S

Разработать программу поиска пары минимально различных по длине в любом заданном наборе отрезков прямых линий, которые произвольно расположены в ее графическом окне ограниченного размера и одновременно видны. Набор отрезков формируется и изменяется путем рисования новых или удаления существующих отрезков. Для рисования отрезков должна использоваться техника резиновой нити, которая изображается пунктиром и управляется перемещением курсора мыши в рамке графического окна, пока нажата ее левая кнопка. Удаление любого отрезка должно производиться щелчком правой кнопки мыши, когда курсор находится в зоне его изображения. Эти изменения должны синхронно отражаться в динамическом массиве, где отрезки упорядочены по длине, с автоматическим выделением минимально различных отрезков цветом их изображений в графическом окне. Завершение программы должно обеспечивать нажатие комбинации клавиш CTRL-S на клавиатуре. При разработке программы следует предусмотреть соответствующую обработку событий и изображений в ее графическом окне, используя библиотечные функции программного интерфейса Xlib из состава X Window System, а также реализовать сортировку массива отрезков по длине.

Структурный состав программы

Исходный код программы графической сортировки и поиска составляют 4 модуля прикладных функций: основной диспетчер (xsort2.c), резиновый редактор (xsort1.c), экстра- модуль (xsort0.c) и фигурный (в данном случае овальный) модуль (xoval.c). Их информационную связь обеспечивает заголовочный файл “xsort.h”, который подключается директивой include в начале каждого модуля. В него входит декларация объединения (union) X-типов геометрических фигур (XFragment), макроопределение цветных индексов массива графических контекстов (GC) для их изображения и индексов массива экстра- методов функциональной обработки массива геофигур (FRAG).

3 поля объединения XFragment поддерживают хранение массивов отрезков прямых, прямоугольников и эллипсов. Их оконные координаты и георазмеры задаются соответствующими стандартными X-типами примитивов графического вывода (XSegment, XRectangle и XArc). Графические контексты предусматривают цветную дифференциацию фиксированных (FGC), резиновых (RGC), экстремальных (EGC) и альтернативных групповых (AGC) изображений геофигур, а также фона (BGC). Экстра-методы обеспечивают выбор минимальной (MINI), максимальной (MAXI) и средней (MEAN) по размерам в массиве геофигур. Также предусматривается выбор пары максимально различных (DIFF) и максимально идентичных (IDEN) геофигур в их массиве, а также разделение массива геофигур на 2 группы (GRP2) по средней экстра-фигуре.

Кроме того, заголовочный файл “xsort.h” содержит спецификации прототипов всех прикладных функций с их разделением по программным модулям. Прикладные

функции модулей xsort[012] инвариантны по типам геофигур, а также не зависят от критерия выбора экстра-фигуры (или их пары). Все фигурно зависимые функции сосредоточены в фигурном модуле, в данном случае xoval. В общем случае их исходный код должен быть переписан под тип геофигуры задания с соответствующим переименованием фигурного модуля. При этом имена и состав его функций должны остаться без изменений.

Основной управляющий модуль (xsort2.c)

Глобальные графические параметры (static)

Display* dpy Адрес дисплейной структуры
GC gc[] Массив цветных графических контекстов Window Win Идентификатор корневого и основного окна Прикладные функции

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 X-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 x 480. Его фон должен совпадать с фоном в массиве графических контекстов. Также задается маска событий, которые будут обрабатываться диспетчером dispatch.

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.

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

rekey – обработка событий нажатия клавиши Escape для очистки графокна программы и комбинации Ctrl-F для выхода.

main – основная функция для вызова всех прикладных функций управляющего модуля, а также функции pass1 из резинового модуля (xsort1) для адресации ему граф. параметров данного модуля.

Резиновый модуль (xsort1.c)

Глобальные графические параметры (static)

Адресуются из управляющего модуля (xsort2.c)

Глобальные геометрические параметры (static)

XFragment* frag – адрес динамического массива геофигуры

int nfrag – длина массива геофигур

XFragment ftmp[1] – шаблон резиновой геофигуры

XFragment bak[2] – BAK-копия для сохранения (пары) геофигур(ы) с требуемым экстра- свойством

int extra[2] – индекс(ы) экстра фигур(ы) в массиве геофигур

int GGC – индекс групповой раскраски геофигур

Прикладные функции

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

photo – передача геопараметров в экстра-модуль после каждого редактирования вызовом экстра-функции pass0.

XFixes – перерисовка заданного числа адресованных геофигур с заданным графконтекстом. Реализует циклический вызов фигурной функции XFix из фигурного модуля (Передвинуть в фигурный?)

rebak - сохранение BAK-копии экстра-фигур(ы) для стирания после перевыборов экстра- фигур(ы), когда их расположение в массиве геофигур frag будет изменено.

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.

near – поиск геофигуры по координатам в её аргументах. Вызывается функцией rubout для удаления геофигуры. Проверку близости к каждый геофигуре из их массива обеспечивает вызов фигурной функции fragon из фигурного модуля.

rerub – реализует деформацию резинового шаблона геофигуры ftmp по событию MotionNotify при перемещении мыши с нажатой левой кнопкой. Резиновое редактирование реализует вызов фигурных функций fragvar и XContour из фигурного модуля, чтобы изменить размеры резинового шаблона и изображение его контура (стереть прошлый габарит и нарисовать новый габарит в резиновом графконтексте).

reggc – установить индекс альтернативного графконтекста (когда нужно делить массив геофигур на 2 группы) для групп овой раскраски геофигур. Вызов этой функции предусмотрен в групповом экстра-методе grp2extr из экстра-модуля, где требуемое значение индекса AGC передается через её аргумент. В коде функции это значение присваивается геопараметру GGC. При всех остальных экстра-критериях групповой раскраски не требуется и значение геопараметра GGC=FGC. Значение геопараметра GGC используется только функцией refrag.

refrag – тотальная перерисовка геофигур. Производится явно после завершения каждой операции редактирования массива геофигур при перевыборах экстра-фигур(ы) или неявно при обращении из функции expo основного модуля, которая вызывается диспетчером dispatch для обработки события Expose при любой потере изображения в графокно программы. Для перерисовки геофигур используются резиновая функция XFixes, а для экстра-фигур(ы) вызывается фигурная функция XExtra.

widewin – расширение графокна программы, когда зафиксированное изображение новой геофигуры пересекает его правую или/и нижнюю границу. Вычисление максимальных координат геофигуры обеспечивают фигурные функции fragmaxix и fragmaxiy. Если эти значения превышают габариты графокна программы, оно расширяется, чтобы геофигура стала полностью видна. Функция widewin необходима только для геофигур с центральной базой, например, круг или овал. Для геофигур с угловой базой, например, прямоугольни к или отрезок прямой она не нужна.

cancet – отменить фиксацию шаблона резиновой геофигуры при его наложении на изображение любой геофигуры в графокне программы, или когда размеры шаблона меньше минимально допустимых значений. Для анализа этих ситуаций вызываются фигурные функции fragover и tinyfrag.

fix – фиксирует контур резинового шаблона в графокне программы и добавляет его в массив геофигур. Эти действия сопровождаются при необходимости расширением габаритов окна программы функцией widewin и сортировкой массива геофигур стандартной функцией qsort. Для сравнения геофигур ей адресуется функция fragcmp, которая обязана быть специфицирована в одном модуле с точкой вызова qsort. Расширение массива геофигур обеспечивает стандартная функция realloc с одновременным инкрементом его длины nfrag.

fragcmp – сравнение пары геофигур для стандартной функции qsort, которую вызывает функция fix для сортировки массива геофигур (см. выше). Парное сравнение геофигур обеспечивает вызов фигурной функции difrag. Такое косвенное сравнение геофигур является вынужденным, т.к. функция сравнения, адресуемая

qsort не может быть специфицирована с другом, в частности, в фигурном модуле (см. выше fix)

miniwin – устанавливает минимально допустимые габариты графокна программы для оконного менеджера. Их определяют максимальные координаты контуров имеющихся геофигур. Тогда при любых интерактивных изменениях размеров графокна программы все геофигуры будут полностью видны. Для этих координатных и габаритных расчетов используются фигурные функции fragmaxix и fragmaxiy, которые должны быть вызваны для каждой геофигуры. Функция miniwin должна вызываться после каждой модификации массива геофигур.

purgextra – стирает изображение BAK-копии экстра-фигур(ы) и изображение геофигуры, которая теперь обладает требуемым экстремальным свойством после редактирования и перевыбора экстра-фигур(ы) в функции rextra из экстра-модуля. Для стирания вызываются фигурные функции XExtra и XFix с графконтекстом фона графокна. Стирание необходимо, чтобы исключить наложение контурных и закрашенных изображений гео- и экстра-фигур, когда на месте закрашенного изображения должен быть нарисован контур. Ясно, что стирание требуется только при изменении экстра-фигур(ы).

allfree – очистка массива геофигур и изображения в графокне программы с обнулением всех геопараметров. Эта функция должна вызываться диспетчером событий dispatch из основного модуля при нажатии клавиши Escape и при завершении программы по нажатию комбинации клавиш Ctrl-F.

Экстра-модуль (xsort0.c)

Глобальные геометрические параметры (static)

Адресуются из резинового модуля (xsort1.c) вызовом прикладной функции pass0 в коде резиновой функции photo. Передача геопараметров осуществляется через массив их адресов, который заполняется в резиновой функции photo и является аргументом функции pass0, где присваиваются соответствующим адресным геопараметрам экстра-модуля.

Прикладные функции (экстра-методы)

Реализуют методы выбора геофигуры или пары геофигур, который обладают требуемым экстремальным свойством в глобальном массиве геофигур, где они упорядочены по размеру. Индекс(ы) экстра-фигур(ы) фиксируются в массиве extra[2], который адресован из резинового модуля и доступен его функциям также как в следующих экстра-методах:

miniextra – выбор минимальной экстра-фигуры.
maxiextra - выбор максимальной экстра-фигуры.
meanextra –выборсреднейпопорядкуэкстра-фигуры. diffextra – выбор пары максимально различных экстра-фигур. idenextra – выборпарымаксимальносходныхэкстра-фигур.

grp2extra – деление массива геофигур на 2 группы до и после центральной экстра-фигуры. Кроме того, предусмотрен внутренний вызов резиновой функции reggc с альтернативным групповым индексом графконтекста для присвоения его значения соответствующему геопараметру GGC резинового модуля.

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.

Аналогично резиновому модулю, исходный код экстра-модуля не зависит от типа геофигур и критерия выбора экстра-фигур(ы) в стандартном наборе. Экстра-модуль может быть модифицирован, когда нужно изменить критерии выбора эсктра-фигур или расширить их набор с дополнительными макроиндексами в заголовочном файле.

Фигурный (рамочный) модуль (xframe.c)

Определение и преобразование типов геометрических фигур

В фигурном модуле сосредоточен исходный код прикладных функций, который зависит от типа геофигур. Требуемый тип обозначается XFig и должен быть определён в начале модуля директивой typedef. При этом допустимый набор типов ограничен стандартным X-типами XSegment (отрезок), XRectangle (прямоугольник) и XArc (овал или круг) по X-типам полей объединения XFragment (см. xsort.h), которое адресует геофигуры для инвариантной обработки в обоих типонезависимых модулях. Явное адресное преобразование (XFragment *) => (XFig *) для аргументов фигурных функций, которое обеспечит корректный доступ к полям необходимого X типа, реализует макрос REFIG. Он макроопределён в начале фигурного модуля и присутствует во всех фигурных функциях с аргументами типа (XFragment *)

Прикладные фигурные функции

fragon: обеспечивает идентификацию геофигуры по заданным координатам (x, y). Возвращает 0 значение (FALSE), если точка не принадлежит области (или контуру) геофигуры. Иначе возвращается положительный код (TRUE), значение которого показывает, например, предельно-допустимую погрешность идентификации.

fragover: обеспечивает контроль взаимного перекрытия изображений для пары адресованных геофигур. Возвращает 0-код (FALSE), когда перекрытия нет и 1-код (TRUE), если геофигуры имеют общие точки.

difrag: оценка различия для адресованной пары геофигур. Вызывается при сортировке массива геофигур в коде резиновой функции fragcmp, которая адресуется qsort (см. xsort1.c)

fragextra: обеспечивает адресный вызов экстра-метода из массива их адресов её аргумента. Идентификацию требуемого экстра-метода обеспечивает подстановка соответствующего макроиндекса из набора их допустимых значений (см. xsort.h). Вызов этой функции осуществляет комплексный экстра-метод rextra при перевыборе экстра-фигуры.

fragsize: возвращает габаритный размер адресованной геофигуры.

tinyfrag: сравнение габаритов адресованной геофигуры с минимально допустимыми размерами, которые установлены в исходном тексте этой фигурной функции. Если габарит меньше предельно-допустимой величины, возвращает значение 1 (TRUE), иначе возвращается 0.

frag0: фиксирует базовую точку адресуемой геофигуры в начале операции резинового редактирования. Для круга или овала база совпадает с его центром. Для прямоугольника база фиксирует одну из его вершин, а для отрезка указывает его начало. В любом случае базовые (x, y)-координаты должны передаваться в функцию frag0 вместе с адресом геофигуры.

fragvar: обеспечивает вариацию контурных габаритов адресованной геофигуры в ходе её резинового редактирования. Габаритные размеры вычисляются по отклонению (x, y)- координат (курсора мыши) в аргументах этой фигурной функции от базы резинового контура геофигуры, которая была зафиксирована предшествующим вызовом фигурной функции frag0. В коде fragvar координаты базы получаются из X-структуры геофигуры при её нулевых габаритах. Их значения сохраняют внутренние статические переменные для последующих оценок габаритных отклонений от базы.

fragmaxix: возврат максимальной координаты X в габарите адресованной геофигуры. Она необходима при выходе за границы окна контуров геофигур с центральной базой (овал или круг). Для геофигур с угловой базой (прямоугольник или отрезок) эта фигурная функция не является необходимой.

fragmaxiy: возврат максимальной координаты y в габарите адресованной геофигуры (см. fragmaxix).

XContour: изображение контура адресованной геофигуры с заданным обычно резиновым графконтекстом (RGC).

XFix: изображение адресованной геофигуры с дополнительной прорисовкой её контура (исключая отрезки) в заданном графконтексте.

XExtra: изображение адресованной экстра-фигуры с дополнительной прорисовкой её контура (исключая отрезки) в заданном графконтексте.

Исходный код 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 0   /* min (Left sort) fragment */
#define MAXIFRAG 1   /* max (Right sort) fragment */
#define MEANFRAG 2   /* mean (Medium sort) fragment */
#define DIFFFRAG 3   /* 2 max differ (Left & Right) fragments */
#define IDENFRAG 4   /* 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 widewin();                     /* extend window */
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();                        /* free screen */

/* 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 fragsize(XFragment*);            /* check fragment size */
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 radical(int);                    /* root square */
int fragover(XFragment*, XFragment*); /* overlap 2 fragmentes */
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 */
```

xsort0.c

```
#include <limits.h>
#include <X11/Xlib.h>
#include "xsort.h"

#define MAXINT INT_MAX

/* Rubber parameter passed from rubber module xsort1 */

static int* nfrag;       /* all fragments' number */
static XFragment* frag;  /* 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];
   frag = (XFragment* ) p[2];
   nfrag = (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] = (*nfrag - 1);
   return(MAXIFRAG);
```

```
} /* maxiextra */

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

int meanextra() {
   extra[0] = extra[1] = ((*nfrag)/2);
   return(MEANFRAG);
} /* meanextra */

/* 2 max differ fragments pair extra */

int diffextra() {
   extra[0] = 0; extra[1] = *nfrag - 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 */
   int e[2];                        /* work extra pair */
   extra[0] = extra[1] = 0;             /* init extra pair */
   for(i = 0, j = 1; j < nfrag[0]; i++, j++) { /* check by 2 */
      if((d = difrag(frag + j, frag + 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] = (nfrag[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, frag+extra[0]); /* purge 1st bak & new extra space */
   if(extra[0] != extra[1])      /* purge 2nd bak & new extra space */
      purgextr(bak+1, frag+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 widewin() {
   int w, h; /* window min width & height for temp fragment */
   XWindowAttributes attr;            /* window attributes */
   XGetWindowAttributes(dpy, win, &attr);
   w = fragmaxix(ftmp);
   h = fragmaxiy(ftmp);
   if((w < attr.width) && (h < attr.height))  /* no expand */
      return(0);             /* window for inside template */
   if(w < attr.width)      /* check expand window by width */
      w = attr.width;            /* no expand window width */
   if(h < attr.height)    /* check expand window by height */
      h = attr.height;          /* no expand window height */
   XResizeWindow(dpy, win, w, h);    /* expand window size */
   return(0);
} /* widewin */

/* Cancel template fragment, when overlap or tiny size */

int cancet() {
   int i;   /* fragment index */
   if(tinyfrag(ftmp) > 0)            /* escape tiny template */
      return(1);
   for(i=0; i < nfrag; i++)      /* escape overlaped template */
      if(fragover(ftmp, frag + i) > 0)
         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 */
   widewin(w, h);             /* extend window to right or bottom */
   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 */
         "black",     /* foreground all fragments */
         "grey",      /* rubber fragment color */
         "black",     /* extra fragment color */
         "black",     /* group altforeground fragment color */
         "white"      /* 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("lab4_res");                  /* rdb <- lab4_res */
   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]); */
/* or rval->addr = rdef[i]; rval->size = strlen(rdef[i]);  */
/*  XrmPutResource(&rdb, rname[i], "String", rval); */
        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, "Lab3");                   /* Window title */
    emask = (ButtonPressMask | ButtonReleaseMask | ButtonMotionMask |
          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_Escape)
      return('F');                  /* CTRL-F to return exit-code */
   if(ks == XK_Delete)       /* Nothing to do if any key except ESC */
      allfree();
   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 */
            rubin(&event);
            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 */
   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 */
```

xsegment.c

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

#include <X11/Xlib.h>
#include <math.h>    /* nesessary for library sqrt() */
#include "xsort.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 */

#define EXTRAFILL 0 /* extra fragment fill (set 1 or 0) */

#define FIXEDFILL 0 /* fix fragment nofill (set 1 or 0) */

/* check fragment fill macros by type */

#define ISFILL(t) ((t) > 0 ? EXTRAFILL : FIXEDFILL)



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

int fragon(XFragment* ff, int x, int y, int t) {
   XFig* f = REFIG(ff);
   XPoint p[2];            /* focuses pos */
   if(((x < (f->x1 - 1)) || (x > (f->x2 + 1))) &&
   ((y < (f->y1 - 1)) || (y > (f->y2 + 1))))
      return(0);     /* no oval inclusion rectangle zone */
   if(t < 0)         /* for invoke by fragover */
      return(-t);      /* to overlap control */
   if(ISFILL(t) > 0)  /* check filling oval */
      return(1);       /* hit inside oval */
   return(1);         /* hit oval contour */
} /* fragon */

/* Overlap 2 fragmentes */

int fragover(XFragment* ff1, XFragment* ff2) {
   XFig* f1 = REFIG(ff1);
   XFig* f2 = REFIG(ff2);
   XSegment s1, s2;  /* oval f1 & f2 frames */
   XSegment r;       /* overlap oval frames */
```

```
    int x, y;         /* internal overlap frame pointer */
    s1.x1 = f1->x1; s1.x2 = f1->x2;
    s1.y1 = f1->y1; s1.y2 = f1->y2;
    s2.x1 = f2->x1; s2.x2 = f2->x2;
    s2.y1 = f2->y1; s2.y2 = f2->y2;
    if((s1.x1 > s2.x2) || (s2.x1 > s1.x2) ||
       (s1.y1 > s2.y2) || (s2.y1 > s1.y2))
        return(0);                  /* No overlap oval frames */
    r.x1 = (s1.x1 > s2.x1) ? s1.x1 : s2.x1;  /* max frames x1 */
    r.y1 = (s1.y1 > s2.y1) ? s1.y1 : s2.y1;  /* max frames y1 */
    r.x2 = (s1.x2 < s2.x2) ? s1.x2 : s2.x2;  /* min frames x2 */
    r.y2 = (s1.y2 < s2.y2) ? s1.y2 : s2.y2;  /* min frames y2 */
    r.x1 -= 2; r.y1 -= 2; r.x2 += 2; r.y2 += 2; /* line width */
    for(y = r.y1; y < r.y2; y++)          /* y-scan overframe */
        for(x = r.x1; x < r.x2; x++)        /* x-scan overframe */
            if(fragon(ff1, x, y, -1) > 0)   /* inner point oval 1 */
                if(fragon(ff2, x, y, -1) > 0) /* inner point oval 2 */
                    return(1);          /* inner point overlap ovals */
    return(0);                          /* No overlap ovals */
} /* fragover */

/* 2 Fragment difference for qsort*/

int difrag(XFragment* ff1, XFragment* ff2) {
    XFig* f1 = REFIG(ff1);
    XFig* f2 = REFIG(ff2);
    int s1 = sqrt(pow(f1->x2 - f1->x1, 2) + pow(f1->y2 - f1->y1, 2));
    int s2 = sqrt(pow(f2->x2 - f2->x1, 2) + pow(f2->y2 - f2->y1, 2));
    return s1 - s2;
} /* difrag */

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

int fragextra(int (*fe[])()) {   /* set extra method number */
    return((*fe[IDENFRAG])()); /* MINI MAXI MEAN DIFF IDEN GRP2 */
} /* fragextra */

/* Fragment gabarit size */

int fragsize(XFragment* ff) {
    XFig* f = REFIG(ff);
    int size = sqrt(pow(f->x2 - f->x1, 2) + pow(f->y2 - f->y1, 2));
} /* fragsize */

/* Tiny Fragment test */

int tinyfrag(XFragment* ff) {
    XFig* f = REFIG(ff);
    if( fabs(f->x2 - f->x1) < 8 && fabs(f->y2 - f->y1) < 8)
        return 1;
    return 0;
} /* minifrag */

/* Stick fragment base xy-point */

int frag0(XFragment* ff, int x, int y) {
    XFig* f = REFIG(ff);
    f->x1 = x;
    f->y1 = y;
    f->x2 = x;
    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 */


/* 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 */

int fragmaxix(XFragment* ff) {
   XFig* f = REFIG(ff);
   return(f->x2);
} /* fragmaxix */

/* Max fragment y-coordinate */

int fragmaxiy(XFragment* ff) {
   XFig* f = REFIG(ff);
   return(f->y2);
} /* fragmaxiy */
```

Список литературы

1. O'Reilly & Associates, Inc. - Table of contents for Xlib Programming Manual (O'Reilly & Associates, Inc.): Режим доступа к ст. http://www.sbin.org/doc/Xlib/index_contents.html.
2. Adrian Nye - Volume One: Xlib Programming Manual: Режим доступа к ст. http://www.ac3.edu.au/SGI_Developer/books/XLib_PG/sgi_html.
3. Вадим Годунко - Xlib - интерфейс с X Window на языке C : Режим доступа к ст. http://motif.opennet.ru/book3.html..

4. Kluwer Academic Publishers - Fundamentals of X Programming GUI and Beyond.pdf Режим доступа http://ftp.homei.net.ua/index.
5. Kenton Lee - Technical Window System and Motif WWW Sites: Режим доступа к ст. http://www.rahul.net/kenton/xsites.html..
6. Robert W. Scheifler - RFC 1013 - X Window System Protocol. Режим доступа к ст. http://www.apps.ietf.org/rfc/rfc1013.html.
7. Theo Pavlidis - Fundamentals of X Programming GUI and Beyond. Режим доступа http://www.maives.ru/modules/news..