Министерство образования и науки Российской Федерации

Московский государственный технический университет им. Н. Э. Баумана

Кафедра «РК6»

**Отчет по лабораторной работе № 2**

По курсу «Программирование графических приложений»

**Вариант P12**

Выполнил: Петраков С. А.

Проверил: Родионов С. В.

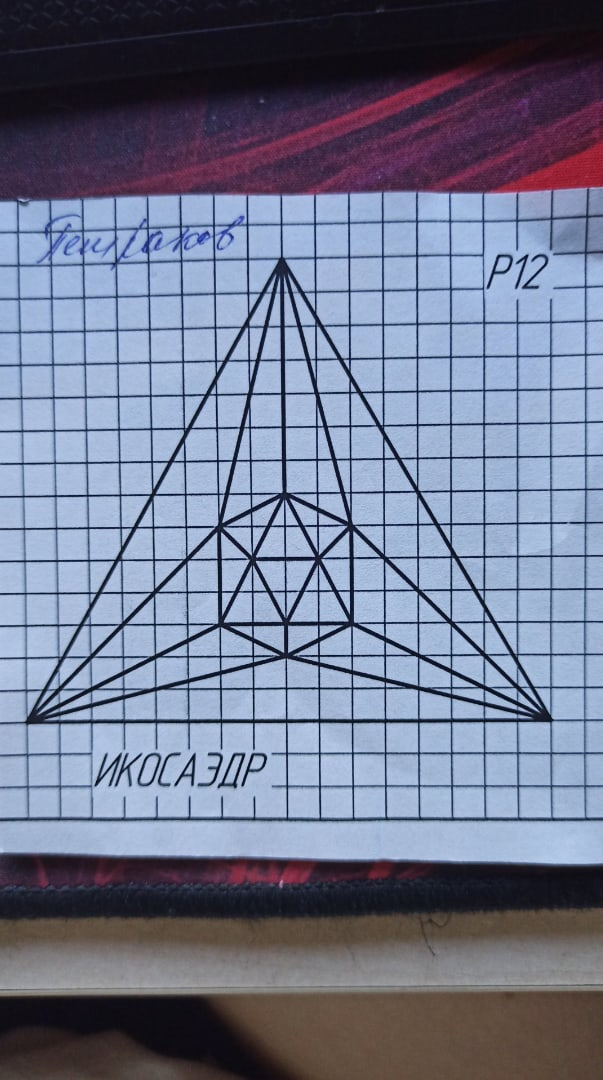
Дата: 16.03.2021

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2021 г.

**Задание**

Разработать программу раскраски граней многоугольного графа плоской прямолинейной укладки любого заданного правильного или полуправильного многогранника. Требуемая фигура должна формироваться по массивам его вершин, граней и ребер, которые определяют их взаимное расположение в графическом окне программы. При этом положение каждой вершины должно фиксироваться ее координатами в условных единицах, пропорциональных размеру графического окна программы, по заданной схеме. Для каждой грани должны быть ,указаны список номеров и число их вершин. Все ребра должны быть заданы списком инциденций из пар номеров своих вершин (или перечислены в минимальном наборе цепей из них, которые специфицированы списками номеров смежных вершин). Закодированное таким образом изображение должно симметрично располагаться в графическом окне и пропорционально реконфигурироваться при любых изменениях его размера. При любы реконфигурациях размер графического окна программы должен быть ограничен сверху габаритами экрана дисплея, а его минимальный размен должен быть установлен из расчета визуальной различимости граней заданной фигуры. В начале выполнения программы графическое окно должно быть занимать четверть площади экрана в его центре, а все грани изображения графа в нем должны иметь одинаковый цвет фона. Изменение цвета каждой грани должно осуществляться по щелчку любой кнопки мыши, когда ее курсор находится внутри грани. Для раскраски граней в программе должна быть распределена палитра из n=4 различных цветов (плюс еще один цвет для изображения вершин и ребер). Чтобы установить необходимый цвет для любой грани в программе должен быть реализован циклический перебор цветов палитры с перекраской указанной грани последовательно в каждый из них по щелчку любой кнопки мыши. Кроме того, следует предусмотреть перезагрузку изображения графа с перекраской в одинаковый фоновый цвет всех граней по нажатию клавиши ESC на клавиатуре, а также принудительную перерисовку графического окна по нажатию комбинации клавиш Ctrl+L с сохранением раскраски граней. Завершение программы должно происходить по нажатию клавиши F10 клавиатуре. При разработке программы должна быть реализована обработка соответствующих событий и изображений в ее графическом окне с многоугольными регионами для граней графа. Для этого следует применить библиотечные функции базового программного интерфейса X Window System. При выполнении программы требуется построить правильную раскраску граней заданной фигуры многоугольного графа минимальным числом цветов, когда все смежные грани имеют различные цвета.



**Выполнение работы**

Исходный текст этой программы раскраски графа целесообразно разделить на 3 модуля из геометрических, дисплейных и контролирующих прикладных функций. Декларации их информационных структур и макроопределение их констант следует сосредоточить в заголовочном файле "lab2.h", который включается в каждый функциональный модуль директивой include. Для вершин и ребер используются типовые графические структуры XPoint и XSegment , которые переименовываются в XVertex и XEdge. Для спецификации граней многоугольного графа декларируется собственная структура XFace. Геометрическую модель многоугольного графа, в которой адресованы его перечисленные компоненты, декларирует программная структура XPolyGraph. Поля структуры XPolyGraph предназначены для адресации одноименных статических массивов геометрического модуля. Их размеры фиксируют число вершин, ребер и внутренних граней (плюс одна внешняя грань) на схеме графа макросами NVERT, NEDGE и NFACE, которые вычисляются в заголовочном файле. Этим вычислениям предшествует макроопределение константы MB0 для числа вершин внешней грани и констант NFm, каждая из которых фиксирует число m-угольных внутренних граней по схеме графа. Кроме указанных топологических констант графа, заголовочный файл также содержит макроопределения инвариантных констант, которые не зависят от структуры графа. Многогранный заголовочный файл завершает спецификация прототипов всех прикладных функций с разделением их по модулям. В геометрический модуль входят 7 прикладных функций для формирования и обработки геометрической модели многоугольного графа по его программной структуре XPolyGraph. Их информационную связь обеспечивают внешние статические массивы и структуры. После заголовка вводятся статические массивы структур вершин, ребер и граней графа для их адресации в одноименных полях его структуры XPolyGraph. Эти массивы образуют инвариантную часть

геометрической модели графа, которая не зависит от его топологии. Их требуется дополнить координатными массивами вершин равноугольных граней, набор и размер которых определяет конфигурация многоугольного графа. Сегмент внешних статических данных геометрического модуля завершает декларация измерительной структуры для установки масштаба графа по горизонтали и вертикали, который зависит от размеров окна программы. Соответствующие коэффициенты масштабирования задают размеры клеток градуировки схемы графа в пикселях графического окна полями внешней графической структуры. Функциональный блок геометрического модуля начинается с прикладной функции assoc. Ее первой вызывает основная функция main, чтобы ассоциировать поля вершин, ребер и граней структуры графа XРolyGraph c одноименными статическими массивами геометрического модуля. Такая ассоциация обеспечивает распределение статической памяти программы для геометрической модели графа, адрес структуры которой передается в функцию assoc. После ассоциации адресов функция assoc вызывается функцию retrace, которая обеспечивает трассировку массива граней геометрической модели графа XPolyGraph для инициализации полей их структур XFace. Вычисление и заполнение координатных данных во всех полях структуры XPolyGraph геометрической модели графа осуществляет функция rebuild. Она вызывается из функции reconf при отработке габаритных реконфигураций графического окна. Для пересчета координат функция rebuild использует свои внутренние статические массивы, которые кодируют конфигурацию вершин, ребер и равноугольных граней по заданной схеме графа. Рассмотренная функция rebuild вызывается при изменении коэффициентов масштабирования изображения графа в окне программы. Их значения вычисляет функция rescale по габаритам окна в своих аргументах и числу делений градуировки схемы графа NUNIT. Результаты вычислений фиксируют поля масштабной структуры scale. Код возврата функции rescale позволяет контролировать наличие изменений масштаба. Если масштаб изменился, возвращается число граней NFACE. Возврат 0 означает сохранение масштаба при малых изменениях габаритов окна. Вычисление масштаба функцией rescale имеет смысл, когда изменяются габариты окна графа. Габаритный контроль окна выполняет функция resize, которой передаются его текущие размеры для сравнения с их прошлыми значениями в ее BAK-структуре. Функция resize запоминает габариты окна из своих аргументов и завершается с кодом числа граней NFACE. Комплексное использование функций resize, rescale и rebuild обеспечивает функция reconf. Ее вызывает диспетчер событий для обработки реконфигурации окна при изменении его размеров. Их текущие значения передаются в функцию reconf парой ее аргументов для контроля изменения размеров окна и масштаба изображения функциями resize и rescale. При их ненулевом возврате вызывается функция rebuild, которая модифицирует геометрическую модель графа для последующей перерисовки его изображения. В любом случае код возврата функции reconf определяется возвратом функции resize и используется для оптимизации серийных перерисовок графа.

Геометрический модуль завершает функция inface, которую вызывает диспетчер событий при выборе грани курсором мыши с целью перекраски в другой цвет. Начальный блок функции inface обеспечивает реформацию регионов всех внутренних граней по top-массивам координат их вершин последовательными запросами XDestroyRegion и XPolygonRegion, если был изменен масштаб изображения. Контроль масштаба по его структуре scale и внутренним BAK- данным реализован как в функции resize. Когда scale- и BAK-структуры совпадают по полям, регионы не изменяются. В любом случае, во втором блоке осуществляется региональный поиск грани по координатам ее внутренней точки, которые заданы аргументами функции inface. Номер этой грани определяется по запросу XPointInRegion для полей zone регионов всех внутренних граней или равен NFACE для внешней грани. Конечный блок изменяет цветное поле tone этой грани, устанавливая для него следующее значение в циклическом порядке индексов цветов. Номер грани передает код возврата функции inface для последующей ее перекраски в установленный цвет функцией reface. В дисплейный модуль входят 7 инвариантных функций, которые обеспечивают отображение многоугольного графа любой конфигурации в графическом окне программы, а также внешние статические переменные для их информационной связи. Декларируются следующие адресные внешние переменные для общего доступа дисплейных функций к массивам вершин, ребер и граней геометрической модели графа, а также для палитры кодов цветов их изображения. Статические адреса vertex, edge и face являются косметическими алиасами одноименных адресных полей модельной структуры графа XPolyGraph, которые введены для удобства доступа. Инициализацию их значений обеспечивает функция relink, которую вызывает основная функция main для адресации структуры геометрической модели графа XPolyGraph в дисплейный модуль. После адресации модельных данных инициализируется статический массив palette, который кодирует доступ дисплейных функций к палитре цветов экрана. Его заполняет функция colorite, которую вызывает основная функция main для распределения цветов раскраски граней и контура графа. Ее аргумент адресует дисплейную структуру Display для идентификации палитры цветов экрана по умолчанию дисплейным макросом DefaultColormap. Набор цветов палитры задают символьные строки внутреннего массива spector спецификаций их RGB- компонент, которые записаны цифровыми парами системы счисления по основанию 16 в традиционном формате обозначения цветных X-ресурсов ("#RRGGBB"). Вместо цифрового кода, цвета могут быть заданы своими текстовыми именами из ресурсного файла (обычно, /usr/X11R6/lib/X11/rgb.txt) X Window System. При желании из него могут быть выбраны и специфицированы произвольные цифровые коды или имена цветов для распределения с учетом ограничений по экранной палитре. Для распределения заданных цветов в палитру по умолчанию используются графические запросы XParseColor и XAllocColor последовательно для каждого цвета спектра. Они заполняют поля цветной структуры XColor для числовых значений RGB-компонент (red, green, blue) и пиксельного кода цвета (pixel). Для цветного рисования графа дисплейные функции используют графический контекст, который формирует сервисная функция congraph. При этом за основу принимается графический контекст по умолчанию, который предоставляет дисплейный макрос DefaultGC для экрана по умолчанию. Его номер идентифицирует дисплейный макрос DefaultScreen. По запросу XChangeGC в структуре XGCValues параметров графического контекста переустанавливаются толщина линий для контура графа и цвет фона. Еще одна сервисная функция wingraph вызывается из основной функции main, чтобы создать окно изображения графа. Окно создается по графическому запросу XCreateWindow в центре экрана и сначала занимает 1/4 его площади. Такие начальные размеры и положение окна вычисляются с помощью дисплейных макросов DefaultWidth и DefaultHeight для номера экрана по умолчанию, который сообщает дисплейный макрос DefaultScreen. Окно программы также является подокном корневого окна экрана, которое идентифицирует дисплейный макрос DefaultRootWindow, наследуя его визуальный класс (CopyFromParent) с числом цветовых плоскостей по умолчанию, установленное дисплейным макросом DefaultDepth. Дисплейную функцию regraph вызывает диспетчерская функция dispatch в цикле обработки событий после создания графического окна или потери изображения в нем. При вызове функции regraph передается адрес дисплейной структуры, идентификатор окна, графический контекст и флаг закраски граней NoFillFace. Если NoFillFace=0, производится раскраска граней в их цвета по запросам XSetForeground и XFillPolygon. Для перерисовки отдельной грани после изменения ее цвета диспетчерская функция dispatch вызывает дисплейную функцию reface. При вызове ей передаются дисплейные параметры как в функцию regraph и номер грани для перерисовки (вместо флага закраски). Действие функции reface различается для внешней и внутренних граней графа. Перекраску внешней грани с номером NFACE обеспечивает вызов функции reset с аргументами функции reface. При этом происходит перезагрузка окна с перекраской его фона, который задает цвет внешней грани графа. Дисплейная функция reset обеспечивает перезагрузку окна программы по клавиатурным сигналам в диспетчере событий dispatch или для перекраски внешней грани в функции reface. В любом случае ей передаются дисплейный адрес, идентификатор окна и параметр FillFace, который определяет результат перезагрузки. Если FillFace=0 (False), то фоновые индексы всех граней устанавливаются по цвету внешней грани для отмены раскраски. Когда FillFace=NFACE (True), они сохраняют свои значения, чтобы восстановить (освежить) раскраску всех внутренних граней. В обоих случаях запрос XSetWindowBackground переустанавливает фон графического окна в цвет внешней грани, который реализуется очисткой всей области окна по запросу XClearArea с нулевыми геометрическими параметрами. При этом значение True его последнего параметра генерирует событие типа Expose как при потере изображения в графическом окне программы. Контрольный модуль составляют функции диспетчеризации событий dispatch, обработки клавиатурных сигналов rekey и основная функция main. Эти 3 функции обеспечивают вызов всех функций дисплейного и геометрического модулей с передачей им информационных структур графа для отображения и модельной обработки. Исходный текст контрольного модуля начинается следующими директивами подключения системных заголовочных файлов с логическими макрокодами клавиш X-графики и прикладного заголовка многоугольного графа. Остальную часть контрольного модуля составляет исходный код его управляющих функций. Наиболее простой в контрольном модуле является функция управления клавиатурой rekey. Эту функцию вызывает диспетчер событий dispatch для перезагрузки графического окна или завершения программы по нажатию управляющих клавиш ESCAPE и F10 на клавиатуре.

**Код**

**“lab2.h”**

#define MB0 4

#define NF3 19

#define NFACE (NF3)

#define NEDGE ((3\*NF3+MB0)/2)

#define NVERT (NEDGE-(NFACE+1)+2)

#define NCOLOR 4

#define NUNIT 18

#define EWIDTH 2

#define VDOT 8

#define DEFTONE 0

typedef XPoint XVertex;

typedef XSegment XEdge;

typedef struct {

XVertex\* top;

int Cn;

int tone;

Region regi;

} XFace;

typedef struct {

XVertex\* vertex;

XFace\* face;

XEdge\* edge;

} XPolyGraph;

int relink(XPolyGraph\*);

GC congraph(Display\*);

Window canvas(Display\*);

int colorite(Display\*);

int regraph(Display\*, Window, GC, int);

int reset(Display\*, Window, int);

int reface(Display\*, Window, GC, int);

int assoc(XPolyGraph\*);

int resize(unsigned, unsigned);

int rescale(unsigned, unsigned);

int rebuild();

int retrace();

int reconf(unsigned, unsigned);

int zotone(int, int);

int rekey(XEvent\*);

int dispatch(Display\*, Window, GC);

**“lab2\_geom.c”**

#include <X11/Xlib.h>

#include <X11/Xutil.h>

#include "lab2.h"

static XVertex vertex[NVERT];

static XFace face[(NFACE + 1)];

static XEdge edge[NEDGE];

static XPoint face3[NF3][(3 + 1)];

static XPoint scale;

int assoc(XPolyGraph\* pg) {

pg->vertex = vertex;

pg->edge = edge;

pg->face = face;

return(0);

}

int resize(unsigned w, unsigned h) {

static XRectangle bak = { 0, 0, 0, 0 };

if ((bak.width == w) && (bak.height == h))

return(0);

bak.width = w; bak.height = h;

return(NFACE);

}

int rescale(unsigned w, unsigned h) {

int x, y;

x = w / NUNIT; y = h / NUNIT;

if ((scale.x == x) && (scale.y == y))

return(0);

scale.x = x; scale.y = y;

return(NFACE);

}

int rebuild() {

static XPoint vconf[] = {

{1, 15}, {17, 15}, {9, 1}, {9, 13}, {7, 12}, {9, 12}, {11, 12}, {8, 10}, {10, 10}, {7, 9}, {9, 8}, {11, 9}

};

static int fconf3[NF3][(3 + 1)] = {

{0, 1, 3, 0},

{1, 2, 11, 1},

{0, 2, 9, 0},

{0, 3, 4, 0},

{0, 4, 9, 0},

{1, 3, 6, 1},

{1, 6, 11, 1},

{2, 9, 10, 2},

{2, 10, 11, 2},

{3, 4, 5, 3},

{3, 5, 6, 3},

{4, 5, 7, 4},

{4, 7, 9, 4},

{5, 7, 8, 5},

{5, 6, 8, 5},

{6, 8, 11, 6},

{10, 7, 9, 10},

{10, 7, 8, 10},

{10, 8, 11, 10},

};

static int econf[NEDGE][2] = {

{0, 1}, {0, 3}, {0, 4}, {0, 9}, {0, 2},

{1, 3}, {1, 6}, {1, 11}, {1, 2},

{2, 9}, {2, 10}, {2, 11},

{3, 4}, {3, 5}, {3, 6},

{4, 5}, {4, 7}, {4, 9},

{5, 6}, {5, 7}, {5, 8},

{6, 8}, {6, 11},

{7, 8}, {7, 9}, {7, 10},

{8, 10}, {8, 11},

{9, 10},

{10, 11},

};

int i, j;

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

vertex[i].x = scale.x \* vconf[i].x;

vertex[i].y = scale.y \* vconf[i].y;

}

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

edge[i].x1 = vertex[econf[i][0]].x;

edge[i].y1 = vertex[econf[i][0]].y;

edge[i].x2 = vertex[econf[i][1]].x;

edge[i].y2 = vertex[econf[i][1]].y;

}

for (i = 0; i < NF3; i++)

for (j = 0; j < (3 + 1); j++) {

face3[i][j].x = vertex[fconf3[i][j]].x;

face3[i][j].y = vertex[fconf3[i][j]].y;

}

return(0);

}

int retrace() { /\* depending on graph ! \*/

int i = 0; /\* total face index \*/

int j; /\* n-top face index \*/

for (j = 0; j < NF3; j++, i++) { /\* fix 3-top faces in face array \*/

face[i].top = face3[j]; /\* fix 3-top face array address \*/

face[i].Cn = 3; /\* fix 3-top face top number=3 \*/

face[i].tone = DEFTONE; /\* set face default tone color \*/

face[i].regi = XCreateRegion(); /\* Empty region for face \*/

} /\* face3 \*/

return(0);

} /\* retrace \*/

/\* Reconfigure graph when window resize & rescale \*/

int reconf(unsigned w, unsigned h) {

if (resize(w, h) == 0)

return(0);

if (rescale(w, h) != 0)

rebuild();

return(NFACE);

}

int zotone(int x, int y) {

static XPoint past = { 0, 0 };

int f = 0;

if ((past.x == scale.x) && (past.y == scale.y))

f = NFACE;

for (; f < NFACE; f++) {

XDestroyRegion(face[f].regi);

face[f].regi = XPolygonRegion(face[f].top, face[f].Cn, 0);

}

past.x = scale.x;

past.y = scale.y;

for (f = 0; f < NFACE; f++)

if (XPointInRegion(face[f].regi, x, y) == True)

break;

face[f].tone = (face[f].tone + 1) % NCOLOR;

return(f);

}

**“lab2\_display.c”**

#include <X11/Xlib.h>

#include <X11/Xutil.h>

#include <X11/keysymdef.h>

#include "lab2.h"

static XVertex\* vertex;

static XFace\* face;

static XEdge\* edge;

static unsigned long palette[(NCOLOR + 1)];

int relink(XPolyGraph\* pg) {

vertex = pg->vertex;

edge = pg->edge;

face = pg->face;

return(0);

}

int colorite(Display\* dpy) {

Colormap cmap;

XColor rgb;

int i;

static char\* spector[] = {

"#ffffff",

"#ff0000",

"#00ff00",

"#0000ff",

"#000000"

};

cmap = DefaultColormap(dpy, DefaultScreen(dpy));

for (i = 0; i < (NCOLOR + 1); i++) {

XParseColor(dpy, cmap, spector[i], &rgb);

XAllocColor(dpy, cmap, &rgb);

palette[i] = rgb.pixel;

}

return(0);

}

GC congraph(Display\* dpy) {

XGCValues gcval;

GC gc;

gcval.line\_width = EWIDTH;

gcval.background = palette[DEFTONE];

gc = DefaultGC(dpy, DefaultScreen(dpy));

XChangeGC(dpy, gc, (GCLineWidth | GCBackground), &gcval);

return(gc);

}

Window canvas(Display\* dpy) {

Window win;

XSetWindowAttributes attr;

XSizeHints hint;

int x, y;

unsigned w, h;

int scr;

attr.override\_redirect = False;

attr.background\_pixel = palette[DEFTONE];

attr.event\_mask = (ButtonPressMask | KeyPressMask | ExposureMask |

StructureNotifyMask | FocusChangeMask);

scr = DefaultScreen(dpy);

w = DisplayWidth(dpy, scr) / 2;

h = DisplayHeight(dpy, scr) / 2;

x = w / 2; y = h / 2;

win = XCreateWindow(dpy, DefaultRootWindow(dpy), x, y, w, h, 1,

DefaultDepth(dpy, scr), InputOutput, CopyFromParent,

(CWOverrideRedirect | CWBackPixel | CWEventMask),

&attr);

hint.flags = (PMinSize | PPosition | PMaxSize);

hint.min\_width = hint.min\_height = (8 \* VDOT);

hint.max\_width = 2 \* w; hint.max\_height = 2 \* h;

hint.x = x; hint.y = y;

XSetNormalHints(dpy, win, &hint);

XMapWindow(dpy, win);

return(win);

}

int regraph(Display\* dpy, Window win, GC gc, int NoFillFace) {

int i;

for (i = NoFillFace; i < NFACE; i++) {

XSetForeground(dpy, gc, palette[face[i].tone]);

XFillPolygon(dpy, win, gc, face[i].top, face[i].Cn,

Convex, CoordModeOrigin);

}

XSetForeground(dpy, gc, palette[NCOLOR]);

XDrawSegments(dpy, win, gc, edge, NEDGE);

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

XFillArc(dpy, win, gc, vertex[i].x - (VDOT >> 1),

vertex[i].y - (VDOT >> 1), VDOT, VDOT, 0, (64 \* 360));

return(0);

}

int reset(Display\* dpy, Window win, int FillFace) {

int f = FillFace;

for (; f < NFACE; f++)

face[f].tone = DEFTONE;

XSetWindowBackground(dpy, win, palette[face[f].tone]);

XClearArea(dpy, win, 0, 0, 0, 0, True);

return(f);

}

int reface(Display\* dpy, Window win, GC gc, int f) {

int i;

if (f == NFACE)

return(reset(dpy, win, f));

XSetForeground(dpy, gc, palette[face[f].tone]);

XFillPolygon(dpy, win, gc, face[f].top, face[f].Cn,

Convex, CoordModeOrigin);

XFlush(dpy);

XSetForeground(dpy, gc, palette[NCOLOR]);

XDrawLines(dpy, win, gc, face[f].top, face[f].Cn + 1,

CoordModeOrigin);

for (i = 0; i < face[f].Cn; i++)

XFillArc(dpy, win, gc, face[f].top[i].x - (VDOT / 2),

face[f].top[i].y - (VDOT / 2), VDOT, VDOT, 0, (64 \* 360));

return(0);

}

**“lab2\_control.c”**

#include <X11/Xlib.h>

#include <X11/Xutil.h>

#include <stdio.h>

#include "lab2.h"

int rekey(XEvent\* ev) {

Display\* dpy = ev->xkey.display;

Window win = ev->xkey.window;

int FillFace;

KeySym ks;

ks = XKeycodeToKeysym(dpy, ev->xkey.keycode, 0);

if (ks == XK\_F10)

return(10);

FillFace = (ev->xkey.state & Mod1Mask) ? NFACE : 0;

if (ks == XK\_Escape)

reset(dpy, win, FillFace);

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

reset(dpy, win, FillFace);

return(0);

}

int dispatch(Display\* dpy, Window win, GC gc) {

int NoFillFace = 0;

XEvent event;

int done = 0;

while (done == 0) {

XNextEvent(dpy, &event);

switch (event.type) {

case Expose:

if (event.xexpose.count > 0)

break;

putchar('E'); fflush(stdout);

regraph(dpy, win, gc, NoFillFace);

break;

case ConfigureNotify:

putchar('C'); fflush(stdout);

NoFillFace = reconf(event.xconfigure.width,

event.xconfigure.height);

break;

case ButtonPress:

reface(dpy, win, gc,

zotone(event.xbutton.x, event.xbutton.y));

break;

case FocusIn:

NoFillFace = 0;

putchar('F'); fflush(stdout);

regraph(dpy, win, gc, NoFillFace);

break;

case KeyPress:

done = rekey(&event);

break;

default: break;

}

}

return(done);

}

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

XPolyGraph heap;

Display\* dpy;

Window win;

GC gc;

int t = (argc > 1) ? 1 : 0;

dpy = XOpenDisplay(NULL);

assoc(&heap);

relink(&heap);

retrace();

colorite(dpy);

gc = congraph(dpy);

win = canvas(dpy);

XStoreName(dpy, win, argv[t]);

dispatch(dpy, win, gc);

XDestroyWindow(dpy, win);

XCloseDisplay(dpy);

return(0);

}

**Литература**

1. Конспекты лекций

2. http://bigor.bmstu.ru

3. Теория графов, Омельченко А.В., 2018

4. 09-02-2021-Электронныи\_\_учебник\_Программирование\_в\_X\_Window\_

System.htm\_\_