

**Министерство образования и науки Российской Федерации**

**Федеральное агентство по образованию**

**Государственное образовательное учреждение высшего профессионального образования**

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

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Робототехника и комплексная автоматизация» (РК)**

**Кафедра «Системы автоматизированного проектирования» (РК6)**

****

**Отчет по лабораторной работе №2 по курсу**

**«Программирование графических приложений»**

**Студент: Сергеева Диана**

**Группа:** РК6-46Б

**Преподаватель:** Волосатова Т.М.

Проверил:

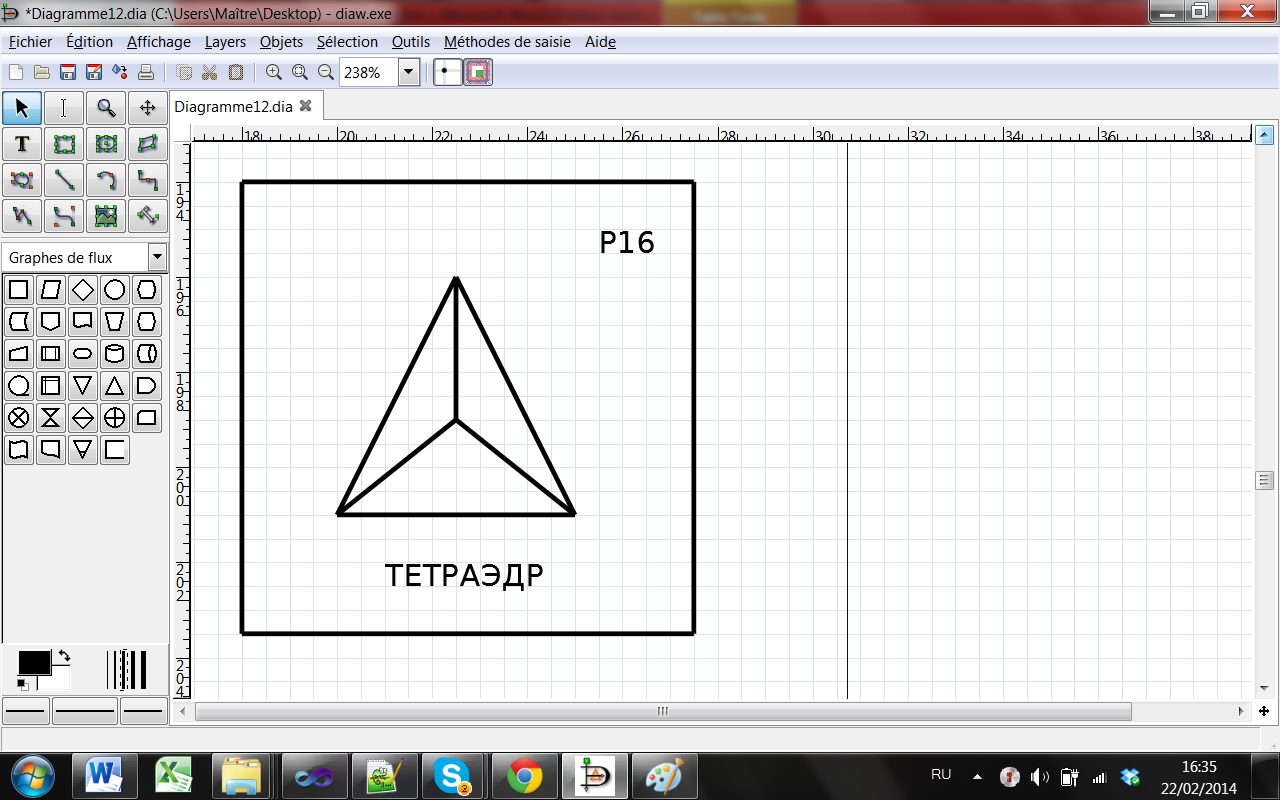
Дата:

2021 год

**16 вариант**

**Задание:**

Разработать программу раскраски граней многоугольного графа плоской прямолинейной укладки заданного (полу)правильного 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. Для каждой грани должны быть указаны: список номеров и число ее вершин. Все ребра должны быть перечислены в минимальном наборе цепей из них, специфицированных списками номеров смежных вершин. Закодированное таким образом изображение должно быть симметрично расположено в графическом окне и пропорционально реконфигурироваться при любых изменениях его размера. При этом минимальный габарит графического окна программы должен быть установлен из расчета визуальной различаемости граней заданной фигуры. В начале все грани 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 цветов (плюс еще 1 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. Кроме того, следует предусмотреть восстановление исходного цвета всех граней по нажатию клавиши Escape на клавиатуре и принудительную перерисовку графического окна по нажатию комбинации клавиш Ctrl-L. Завершение программы должно происходить по нажатию клавиши F10 . При разработке программы должна быть реализована обработка соответствующих событий и изображений в ее графическом окне с его многоугольными регионами. Для этого необходимо применить библиотечные функции программного интерфейса Xlib из XWindowSystem.



**Алгоритм действий в графическом окне:**

* 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 ALT-ESC 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.
* Для интерактивной раскраски различных многоугольных графов могут быть разработаны функционально идентичные программы. Все они будут различаться только по коду пары конфигурационных функций с координатными и структурными данными, которые допускают техническую переделку по шаблону.

**Структура программы раскраски графа:**

* 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 "polyhedron.h", который включается в каждый функциональный модуль директивой include. Он начинается подключением системных заголовков базового интерфейса и утилит X-графики следующими директивами:

#include <X11/Xlib.h>

#include <X11/Xutil.h>

* Для вершин и ребер используются типовые графические структуры XPoint и XSegment , которые переименовываются в XVertex и XEdge следующими директивами:

typedef XPoint XVertex; /\* Структура вершины \*/

typedef XSegment XEdge; /\* Структура ребра \*/

**Структура многоугольного графа:**

* Для спецификации граней многоугольного графа декларируется следующая собственная структура XFace:

typedef struct { /\* Структура грани \*/

XPoint \*top; /\* Адрес набора вершин \*/

int Cn; /\* Число вершин \*/

int tone; /\* Номер цвета \*/

Region zone; /\* Региональная зона \*/

}XFace;

* Геометрическую модель многоугольного графа, в которой адресованы его перечисленные компоненты, декларирует следующая программная структура XPolyGraph:
* typedef struct { /\* Структура многоугольного графа \*/
* XVertex\* vertex; /\* Адрес массива вершин \*/
* XEdge\* edge; /\* Адрес массива ребер \*/
* XFace\* face; /\* Адрес массива граней \*/
* } XPolyGraph;

**Поля структуры 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, причем, обычно 3<m<8. В частности, для графа пирамиды требуются следующие макроопределения, а для произвольного графа нужно снять некоторые комментарии в соответствии с его конфигурационной схемой:

* #define MB0 4 /\* 4-угольная внешняя грань \*/
* #define NF3 4 /\* 4 3-угольные внутренние грани \*/
* /\* #define NF4 0 \*/ /\* 0 4-угольных внутренних граней \*/
* /\* #define NF5 0 \*/ /\* 0 5-угольных внутренних граней \*/
* /\* #define NF6 0 \*/ /\* 0 6-угольных внутренних граней \*/
* /\* #define NF7 0 \*/ /\* 0 7-угольных внутренних граней \*/
* /\* #define NF8 0 \*/ /\* 0 8-угольных внутренних граней \*/

**Расчет значений угольных констант для других графов:**

* Для других графов эти значения угольных констант должны быть изменены, но в любом варианте они обеспечивают конструктивное определение числа вершин, ребер и граней по соотношениям многоугольных графов. Проще всего найти число внутренних граней NFACE. Его определяет сумма внутренних угольников плоской укладки. Для графа пирамиды в этой сумме остаются только NF3 3-угольников и макроопределение числа граней имеет вид:

  #define NFACE (NF3) /\* (NF3+ NF4+ NF5 \

+ NF6+ NF7+NF8) \*/

* Число ребер NEDGE любого многоугольника графа определяет половина суммарной угольности граней. Для пирамиды вместе с угольностью внешней грани MB0 учитываются опять только NF3 3-угольников. Поэтому макроопределение числа ребер имеет следующий вид, а комментарий показывает расчетную формулу для общего случая:

#define NEDGE ((3\*NF3 + MB0)/2) /\* ((3\*NF3 + 4\*NF4 \

+ 5\*NF5 + 6\*NF6 \

+ 7\*NF7 + 8\*NF8 \

+ MB0)/2) \*/

* Число ребер, согласно формуле Эйлера для любого плоского графа, на два меньше суммы числа вершин и граней (с учетом внешней грани). Вычисление числа вершин NVERT по Эйлеру обеспечивает следующее макроопределение:

#define NVERT (NEDGE-(NFACE+1)+2) /\* V+F−E=2 \*/

**Макроопределения инвариантных констант, не зависящих от структуры графа:**

* Кроме указанных топологических констант графа, заголовочный файл содержит следующие макроопределения инвариантных констант, которые не зависят от структуры графа:
* #define NTONE 4 /\* число цветов граней графа \*/
* #define DEFTONE 0 /\* номер цвета грани по умолчанию \*/
* #define VDOT 8 /\* диаметр вершин графа \*/
* #define EWIDTH 2 /\* толщина ребер графа (<VDOT) \*/

#define NUNIT 8 /\* диапазон градуировка схемы %8=0\*/

**Структура геометрического модуля:**

* 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. Исходный код модуля начинает подключение многоугольного заголовка графа следующей директивой:

#include "polyhedron.h"

* После заголовка вводятся следующие статические массивы структур вершин, ребер и граней графа для их адресации в одноименных полях его структуры XPolyGraph:

static XVertex vertex[NVERT]; /\* массив вершин \*/

static XEdge edge[NEDGE]; /\* массив ребер \*/

static XFace face[(NFACE+1)]; /\* массив граней \*/

* 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. Для графа пирамиды, который имеет только NF3=4 (боковые) 3-угольные внутренние грани, определяется следующий массив для пар координат их вершин:

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

**Задание координатных массивов вершин равноугольных граней:**

* Ряды пар координат вершин каждой грани этого массива адресуются (топовыми) полями (top) структуры граней XFace и составляют топозависимую часть геометрической модели графа, которую определяет его топология и конфигурация. Для любого заданного графа нужно заявить аналогичные координатные массивы его m-угольных граней (3<m<8), выбрав соответствующие строки из следующего блока комментария:
* /\* static XPoint face4[NF4][(4+1)]; \*/
* /\* static XPoint face5[NF4][(4+1)]; \*/
* /\* static XPoint face6[NF4][(4+1)]; \*/
* /\* static XPoint face7[NF4][(4+1)]; \*/
* /\* static XPoint face8[NF8][(8+1)]; \*/
* 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 следующей внешней графической структуры:
* static XPoint scale; /\*структура масштаба по Х и Y \*/

Ряды пар координат вершин каждой грани этого массива адресуются (топовыми) полями (top) структуры граней XFace и составляют топозависимую часть геометрической модели графа, которую определяет его топология и конфигурация.

**Координатные массивы его m-угольных граней (3<m<8) для любого графа:**

* Для любого заданного графа нужно заявить аналогичные координатные массивы его m-угольных граней (3<m<8), выбрав соответствующие строки из следующего блока комментария:

/\* static XPoint face4[NF4][(4+1)]; \*/

/\* static XPoint face5[NF4][(4+1)]; \*/

/\* static XPoint face6[NF4][(4+1)]; \*/

/\* static XPoint face7[NF4][(4+1)]; \*/

/\* static XPoint face8[NF8][(8+1)]; \*/

* 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 следующей внешней графической структуры:
* static XPoint scale; /\*структура масштаба по Х и Y \*/
* (3<m<8), выбрав соответствующие строки из следующего блока комментария:

 /\* static XPoint face4[NF4][(4+1)]; \*/

/\* static XPoint face5[NF4][(4+1)]; \*/

/\* static XPoint face6[NF4][(4+1)]; \*/

/\* static XPoint face7[NF4][(4+1)]; \*/

/\* static XPoint face8[NF8][(8+1)]; \*/

 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 следующей внешней графической структуры:

static XPoint scale; /\*структура масштаба по Х и Y \*/

**Функциональный блок геометрического модуля. Прикладная функция assoc:**

* 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. Исходный код функции assoc имеет следующий вид.
* /\* Модельная ассоциация структуры полиграфа \*/
* int assoc(XPolyGraph\* pg) {
* pg->vertex = vertex; /\* адресация массива вершин \*/
* pg->edge = edge; /\* адресация массива ребер \*/
* pg->face = face; /\* адресация массива граней \*/
* retrace(); /\* трассировка граней /\*
* return(0);
* } /\* assoc \*/

После ассоциации адресов функция assoc вызывается функцию retrace, которая обеспечивает трассировку массива граней геометрической модели графа XPolyGraph для инициализации полей их структур XFace. При этом в top-поля указанных структур адресуются статические массивы для координат вершин внутренних равноугольных граней как face3 у графа пирамиды.

**Трассировка рав(з)ноугольных граней в 1 массив:**

* В общем случае исходный код функции трассировки retrace образует набор циклов адресации и инициализации для всех равноугольных граней со сквозной индексацией их номеров i в порядке роста числа их вершин. Для графа пирамиды, в частности, требуется только один цикл по 3-угольным граням.

/\* код Функции зависит от графа \*/

 int retrace() {

int i=0; /\* сквозной индекс разноугольных граней \*/

int j; /\* индекс равноугольных граней \*/

for(j = 0; j<NF3; j++, i++) { /\* 3-угольная трассировка \*/

face[i].top = face3[j]; /\* адрес массива вершин \*/

face[i].Cn = 3; /\* число вершин грани=3 \*/

face[i].tone = DEFTONE; /\* цветной индекс грани \*/

face[i].zone = XCreateRegion(); /\* пустой регион \*/

} /\* face3 \*/

/\* for(j = 0; j < NFm; j++, i++) { ... } \*/ /\* для m>3 \*/

face[i].tone = DEFTONE; /\* цвет внешней грани \*/

return(0);

} /\* retrace \*/

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.

**Дисплейный модуль:**

В него входят 7 инвариантных функций, которые обеспечивают отображение многоугольного графа любой конфигурации в графическом окне программы, а также внешние статические переменные для их информационной связи. Исходный код дисплейного модуля начинается с подключения заголовка многоугольного графа следующей директивой:

 #include "polyhedron.h"

Затем 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:

static XVertex \*vertex; /\* адрес массива вершин \*/

static XEdge \*edge; /\* адрес массиваребер \*/

static XFace \*face; /\* адрес массива граней \*/

static unsigned long palette[(NTONE+1)]; /\* коды цветов \*/

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. Она имеет следующий исходный код.

**Адресация модельных массивов графа:**

int relink(XPolyGraph \*pg) {

vertex = pg->vertex; /\* адрес массива вершин \*/

edge = pg->edge; /\* адрес массива ребер \*/

face = pg->face; /\* адрес массива граней \*/

return(0);

} /\* relink \*/

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).

**Исходный код:**

**lab2.h**

#define MB0 4

#define NF3 3

#define NFACE (NF3)

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

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

#define NCOLOR 4

#define NUNIT 12

#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\_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);

}

**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\_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, 11}, {6, 1}, {11, 11}, {6, 7}

};

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

{0, 1, 3, 0},

{0, 2, 3, 0},

{1, 2, 3, 1},

};

static int econf[NEDGE][2] = {

{0, 1}, {0, 2}, {0, 3},

{1, 2}, {1, 3},

{2, 3},

};

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)

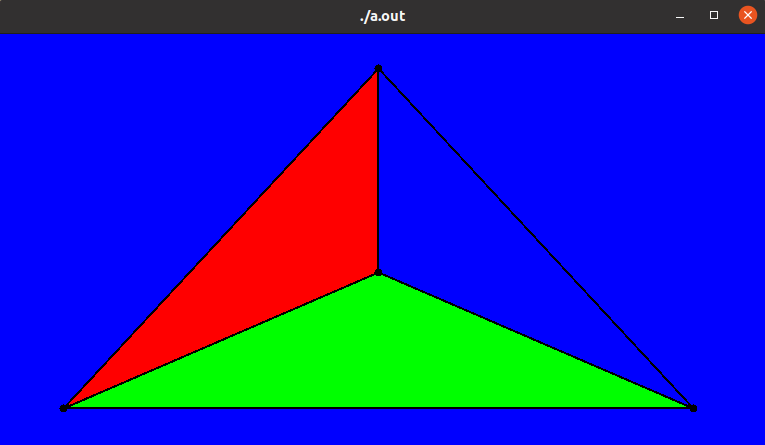
break;

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

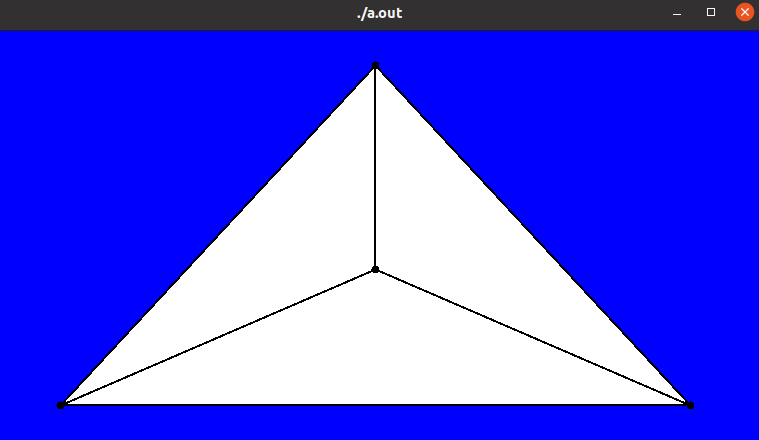
return(f);

}

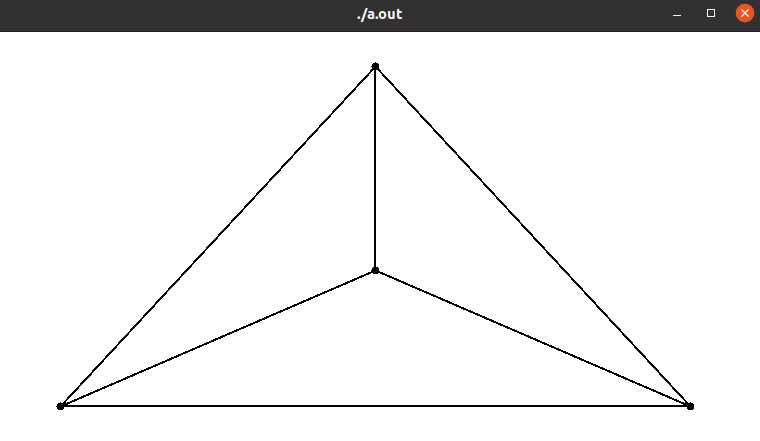
**Результат работы программы:**

****

При нажатии esc:



При нажатии ctrl+l:



**Литература:**

1) bigor.bmstu.ru – база и генератор образовательных ресурсов

2) Лекции по курсу «Программирование графических приложений»

3) Электронный учебник: http://eufs.bmstu.ru/ee8d5d88-abfa-11e6-aa39-005056960017/09-02-2021-%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B8\_\_%D1%83%D1%87%D0 %D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5\_%D0%B2\_X\_Window\_System.htm