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 №2

По курсу «Программирование графических приложений»

Выполнил: Студент Гусаров Аркадий Андреевич
Группа РК6-43Б

Проверил: _____
Дата _____
Подпись _____

Москва, 2021 г.

1. **Задание:**
   Разработать программу раскраски граней многоугольного графа плоской прямолинейной укладки любого заданного правильного или полуправильного многогранника.

2. **Требования:**
   - 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.
   - Разработать программу раскраски граней многоугольного графа плоской прямолинейной укладки любого заданного правильного или полуправильного многогранника.
   - Разработать программу раскраски граней многоугольного графа плоской прямолинейной укладки любого заданного правильного или полуправильного многогранника.
   - Разработать программу раскраски граней многоугольного графа плоской прямолинейной укладки любого заданного правильного или полуправильного многогранника.

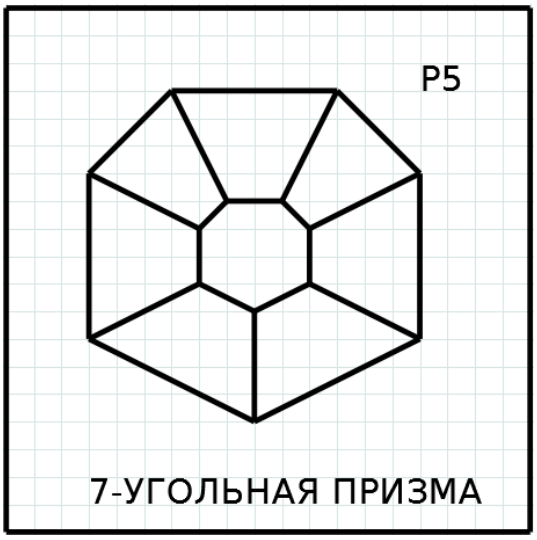
3. **Алгоритм действий в графическом окне:**

   - 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.

4. Программа для раскраски графа пирамиды:



5. Код программы:

- Файл polyhedron.h:

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

typedef XPoint XVertex;                    /* Структура вершины */
typedef XSegment XEdge;                    /* Структура ребра */

typedef struct {                           /* Структура грани */
  XPoint *top;                             /* Адрес набора вершин */
  int Cn;                                  /* Число вершин */
  int tone;                                /* Номер цвета */
  Region zone;                             /* Региональная зона */
}XFace;
```

```
typedef struct {                        /* Структура многоугольного графа */
  XVertex* vertex;                       /* Адрес массива вершин */
  XEdge* edge;                           /* Адрес массива ребер */
  XFace* face;                          /* Адрес массива граней */
} XPolyGraph;

#define MB0  7                           /* 7-угольная внешняя грань */
#define NF4  7                           /* 7 4-угольные внутренние грани */
#define NF7  1                           /* 1 7-угольные внутренние грани */
#define NFACE (NF4+NF7)                        /* (NF4+NF7) - число граней*/
#define NEDGE ((4*NF4 + 7*NF7 + MB0)/2)          /* ((4*NF4 + 7*NF7+ MB0)/2)- число ребер */
#define NVERT (NEDGE-(NFACE+1)+2)               /* число вершин */
#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    17                        /* диапазон градуировка схемы  %8=0 */

/*Геометрический модуль (cub1) */
int assoc(XPolyGraph*);
GC congraph(Display*);
Window wingraph(Display*);
int colorite(Display*);
int regraph(Display*, Window, GC, int);
int reset(Display*, Window, int);
int reface(Display*, Window, GC, int);
/*Дисплейный модуль (cub2) */
int relink(XPolyGraph*);
int retrace();
int resize(unsigned, unsigned);
int rescale(unsigned, unsigned);
int rebuild();
int reconf(unsigned, unsigned);
int zotone(int, int);
/*Контрольный модуль (cub3) */
int rekey(Display*, Window, GC, XEvent*);
int dispatch(Display*, Window, GC);
int main(int, char* argv[]);
```

- Файл cub1.c:

```
#include "polyhedron.h"

static XVertex vertex[NVERT];            /* массив вершин */
static XEdge edge[NEDGE];                /* массив ребер */
static XFace face[(NFACE+1)];            /* массив граней */

static XPoint face4[NF4][(4+1)];
static XPoint face7[NF7][(7+1)];

static XPoint scale;                     /*структура масштаба по X и Y */

/* 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 */

int retrace() {
int i=0;                                 /* сквозной индекс разноугольных граней */
int j;                                   /* индекс равноугольных граней */
for(j = 0; j<NF4; j++, i++)              /* 4-угольная трассировка */
{
  face[i].top = face4[j];                /* адрес массива вершин */
  face[i].Cn = 4;                        /* число вершин грани=4 */
  face[i].tone = DEFTONE;                /* цветной индекс грани */
  face[i].zone = XCreateRegion();        /* пустой регион */
} /* face4 */
for(j = 0; j < NF7; j++, i++)            /* для m>3 */
{
  face[i].top = face7[j];                /* адрес массива вершин */
  face[i].Cn = 7;                        /* число вершин грани=7 */
  face[i].tone = DEFTONE;                /* цветной индекс грани */
  face[i].zone = XCreateRegion();        /* пустой регион */
```

```
        }
        face[i].tone = DEFTONE;                 /* цвет внешней грани */
        return(0);
        } /* retrace */

        int rebuild() {
        static XPoint vconf[] = {               /* Конфигурация вершин в схеме куба*/
         {2, 6}, {5, 3}, {11, 3}, {14, 6}, {14, 12}, {8, 15}, {2, 12}, {7, 7}, {9, 7}, {10, 8}, {10, 10}, {8, 11}, {6, 10},
{6, 8}
        }; /* vconf */
        static int fconf4[NF4][(4+1)] = {       /* Циклические индексы вершин для 4-угольных граней
куба */
         {0,1, 7, 13, 0},
         {1, 2, 8, 7, 1},
         {2, 3, 9, 8, 2},
         {3, 4, 10, 9, 3},
         {4, 5, 11, 10, 4},
         {5, 6, 12, 11, 5},
         {6, 0, 13, 12, 6}
        }; /* fconf4 */
        static int fconf7[NF7][(7+1)] = {       /* Циклические индексы вершин для 7-угольных граней
куба */
         {7, 8, 9, 10, 11, 12, 13, 7}
        }; /* fconf7 */
        static int econf[NEDGE][2] = {          /* Пары вершин ребер: */
            {0, 1}, {0, 13},                    /* инцидентные V0 */
            {1, 2}, {1, 7},                     /* инцидентные V1 */
            {2, 3}, {2, 8},                     /* инцидентные V2 */
            {3, 4}, {3, 9},                     /* инцидентные V3 */
            {4, 5}, {4, 10},                    /* инцидентные V4 */
            {5, 6}, {5, 11},                    /* инцидентные V5 */
            {6, 0}, {6, 12},                    /* инцидентные V6 */
            {7, 8},                            /* инцидентные V7 */
            {8, 9},                            /* инцидентные V8 */
            {9, 10},                           /* инцидентные V9 */
            {10, 11},                          /* инцидентные V10 */
            {11, 12},                          /* инцидентные V11 */
            {12, 13},                          /* инцидентные V12 */
            {13, 7}                            /* инцидентные V13 */
        }; /* edge */
```

```
int i, j;                          /* индексы вершин, ребер и граней */
for(i = 0; i < NVERT; i++) {            /* Расчет оконных координат вершин  */
 vertex[i].x = scale.x * vconf[i].x;
 vertex[i].y = scale.y * vconf[i].y;
} /* for-vertex */
for(i = 0; i < NEDGE; i++) {                /* Фиксировать оконные координаты пар вершин всех
ребер */
 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 < NF4; i++)                    /* Фиксировать оконные координаты вершин 4-угольных
граней */
 for(j = 0; j < (4+1); j++) {
  face4[i][j].x = vertex[fconf4[i][j]].x;
  face4[i][j].y = vertex[fconf4[i][j]].y;
 }
for(i = 0; i < NF7; i++)                    /* Фиксировать оконные координаты вершин 7-угольных
граней */
 for(j = 0; j < (7+1); j++) {
  face7[i][j].x = vertex[fconf7[i][j]].x;
  face7[i][j].y = vertex[fconf7[i][j]].y;
 }
return(0);
} /* rebuild */

/* Контроль масштаба изображения */
int rescale(unsigned w, unsigned h) {
int x, y;                          /* коэффициенты масштабирования по x и y */
x = w / NUNIT; y = h / NUNIT;           /* пересчет масштаба */
if((scale.x == x) && (scale.y == y))
 return(0);                        /* код сохранения масштаба */
scale.x = x; scale.y = y;              /* запомнить масштаб */
return(NFACE);                     /*  код изменения масштаба */
} /* rescale */

/* Контроль изменения размеров окна */
int resize(unsigned w, unsigned h) {
static XRectangle bak = {0, 0, 0, 0 };     /* прошлые размеры */
```

```
if((bak.width == w) && (bak.height == h))
  return(0);                          /* код сохранения размеров окна */
bak.width = w; bak.height = h;            /* запомнить размеры */
return(NFACE);                        /* код изменения размеров окна */
} /* resize */

/* Реконфигурация графа */
int reconf(unsigned w, unsigned h) {
if(resize(w, h) == 0)                 /* Габаритный контроль */
  return(0);
if(rescale(w, h) != 0)                /* Контроль масштаба */
  rebuild();                          /* Перестройка геометрии графа */
return(NFACE);
} /* reconf */

int zotone(int x, int y) {
static XPoint bak = {0, 0};               /* прошлый масштаб */
int f = 0;                            /* индекс грани */
if((bak.x == scale.x) && (bak.y == scale.y))/* Контроль */
  f = NFACE;                          /* изменений масштаба изображения */
for( ; f < NFACE; f++) {              /* Перестройка регионов граней */
  XDestroyRegion(face[f].zone);
  face[f].zone = XPolygonRegion(face[f].top, face[f].Cn, 0);
} /* for */
bak.x = scale.x; bak.y = scale.y;         /* запомнить масштаб */
for(f = 0; f < NFACE; f++)            /* поиск грани по точке внутри */
  if(XPointInRegion(face[f].zone, x, y) == True)
    break;
face[f].tone = (face[f].tone + 1) % NTONE;  /* новый цвет */
return(f);                            /* возврат индекса грани для перекраски */
} /* zotone */
```

- Файл cub2.c:

```
#include "polyhedron.h"
#include <X11/XKBlib.h>

static XVertex *vertex;                   /* адрес массива вершин */
static XEdge *edge;                       /* адрес массива ребер */
static XFace *face;                       /* адрес массива граней */
```

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

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 */

int colorite(Display* dpy)
{
   int scr;                              /* номер экрана (по умолчанию) */
   Colormap cmap;                             /* палитра (карта) цветов экрана */
   XColor rgb;                           /* цветная структура */
   int i;                               /* спектральный номер цвета */
   static char* spector[] = {                 /* Спектр кодов (имен) цветов */
                     "#ffffff",          /* или "W(w)hite" (белый) */
                     "#ff0000",          /* или "R(r)ed" (красный) */
                     "#00ff00",          /* или "G(g)reen" (зеленый) */
                     "#0000ff",          /* или " B(b)lue" (синий) */
                     "#000000"           /* или "B(b)lack" (черный) */
                     };                 /* RGB-спецификация цветов */
   scr = DefaultScreen(dpy);                  /* получить номер экрана (0) */
   cmap = DefaultColormap(dpy, scr);          /* экранная палитра */
   for(i = 0; i < (NTONE+1); i++ ) {          /* Спектральный цикл */
   XParseColor(dpy, cmap, spector[i], &rgb);      /* –> RGB */
   XAllocColor(dpy, cmap, &rgb);              /* –> pixel-код */
   palette[i] = rgb.pixel;                    /* запомнить pixel-код цвета */
   } /* for */
   return(0);
} /* colorite */

GC congraph(Display* dpy)
{
   int scr = DefaultScreen(dpy);              /* номер экрана */
   XGCValues gcval;                           /* параметры графконтекста */
   GC gc;                                /* идентификатор графконтекста */
   gcval.line_width = EWIDTH;                 /* толщина контура графа */
   gcval.background = palette[DEFTONE];       /* код фона */
```

```
    gc = DefaultGC(dpy, scr);                    /* Установка графконтекста */
    XChangeGC(dpy, gc, GCLineWidth | GCBackground, &gcval);
    return(gc);                                  /* GC –> main */
} /* congraph */

Window wingraph(Display* dpy)
{
    Window root;                                 /* идентификатор корневого окна экрана */
    int scr;                                     /* номер экрана по умолчанию */
    int depth;                                   /* число цветовых плоскостей экрана */
    Window win;                                  /* идентификатор окна программы */
    XSetWindowAttributes attr;                   /* структура атрибутов окна */
    XSizeHints hint;                             /* геометрия окнного менеджмента */
    int x, y;                                    /* координаты окна */
    unsigned w, h;                               /* габариты окна */
    unsigned long mask;                          /* маска атрибутов окна */
    mask = CWOverrideRedirect | CWBackPixel | CWEventMask;
    attr.override_redirect = False;              /* WM-контроль окна */
    attr.background_pixel = palette[DEFTONE];    /* цвет фона */
    attr.event_mask = (ButtonPressMask | KeyPressMask |
                       ExposureMask | StructureNotifyMask |
                       FocusChangeMask);         /* Маска событий */
    root=DefaultRootWindow(dpy);                 /* корневое окно */
    scr = DefaultScreen(dpy);                    /* номер экрана */
    depth=DefaultDepth(dpy, scr);                /* глубина экрана */
    w = DisplayWidth(dpy, scr) / 2;              /* Расположить окно */
    h = DisplayHeight(dpy, scr) / 2;             /* площадью 1/4 экрана */
    x = w / 2; y = h / 2;                        /* в центре экрана */
    win = XCreateWindow(dpy, root, x, y, w, h, 1, depth,
                InputOutput, CopyFromParent, mask, &attr);
    hint.flags = (PMinSize | PPosition | PMaxSize);   /* Задать поля для геометрического
свойства WM */
    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);            /* –> свойство WM */
    //XStoreName(dpy, win);                      /* Задать заголовок окна */
    XMapWindow(dpy, win);                        /* Отобразить окно на экране */
    return(win);                                 /* возврат идентификатока окна в main */
} /* wingraph */
```

```
/* Перерисовка контура и перекраска граней графа */
int regraph(Display* dpy, Window win, GC gc, int NoFillFace)
{
   int i;                                  /* счетчик вершин и граней */
   /* Раскраска всех или 0 внутренних граней */
   for(i = NoFillFace; i < NFACE; i++)
   {
      XSetForeground(dpy, gc, palette[face[i].tone]);    /* цвет грани */
      XFillPolygon(dpy, win, gc, face[i].top, face[i].Cn,
                                 Convex, CoordModeOrigin);
   } /* for face */
/* Перерисовка всех ребер и вершин */
   XSetForeground(dpy, gc, palette[NTONE]);            /* –> Black */
   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);
} /* regraph */

int reface(Display* dpy, Window win, GC gc, int f)
{
   int i;                                  /* счетчик вершин грани */
   if(f == NFACE)                             /* перекраска внешней грани */
   return(reset(dpy, win, f));
   XSetForeground(dpy, gc, palette[face[f].tone]);
   XFillPolygon(dpy, win, gc, face[f].top, face[f].Cn,
                  Convex, CoordModeOrigin);        /* Перекраска */
   XFlush(dpy);                                  /* внутренней грани */

   /* Перерисовка контура грани */
   XSetForeground(dpy, gc, palette[NTONE]);            /* –> Black*/
   XDrawLines(dpy, win, gc, face[f].top, face[f].Cn + 1,
                  CoordModeOrigin);              /* перерисовка ребер */
   for(i = 0; i< face[f].Cn; i++)                 /* перерисовка вершин */
   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);
} /* reface */

/* Перезагрузка раскраски граней */
int reset(Display* dpy, Window win, int FillFace)
{
    int f = FillFace;                       /* индекс грани */
    /* Сохранить или Установить цвета внутренних граней по фону внешней грани*/
    for(f ; f < NFACE; f++)
        face[f].tone = DEFTONE;
    /* Установить фон окна и инициировать Expose */
    XSetWindowBackground(dpy, win, palette[face[f].tone]);
    XClearArea(dpy, win, 0, 0, 0, 0, True);          /* –> Expose */
    return(f);
} /* reset */

int rekey(Display* dpy, Window win, GC gc, XEvent* ev)
{
    KeySym sym;                             /* Key symbol code */
    sym = XkbKeycodeToKeysym(dpy, ev->xkey.keycode, 0, 0);
    if(sym == XK_F10)               /* Press F10 for programm exit */
    return(10);       /* exit return for event dispatcher in main() */
    if((sym == XK_l) && (ev->xkey.state & ControlMask))   /* Press C-L */
    reset(dpy, win, NFACE);             /* to Refresh graph display */
    if(sym == XK_Escape)   /* Press Escape to redraw default face tone */
    reset(dpy, win, 0);
    return(0);      /* continue return for event dispatcher in main() */
} /* rekey */
```

- Файл cub3.c:

```
#include <X11/keysym.h>
#include <X11/keysymdef.h>
#include <stdio.h>
#include "polyhedron.h"

int dispatch(Display* dpy, Window win, GC gc) {
int NoFillFace = 0; /* nopaint faces number */
XEvent event;      /* event structure */
int end=0;         /* exit from event loop flag */
```

```
while(end == 0) {   /* event dispatch loop */
  XNextEvent(dpy, &event); /* read from event queue */
  switch(event.type) {     /* event processing */
    case Expose: /* Redraw graph in visible  window space */
                if(event.xexpose.count > 0)
                  break;
                putchar('E'); fflush(stdout);
                /* NoFillFace = 0; */ /* Uncomment for frame WM (olwm) */
                regraph(dpy, win, gc, NoFillFace);
                break;
    case ConfigureNotify: /* Reconfigure graph when resize window */
                      putchar('C'); fflush(stdout);
                      NoFillFace = reconf(event.xconfigure.width,
                                    event.xconfigure.height);
                      break;
    case ButtonPress: /* Repaint pointed face */
                   reface(dpy, win, gc,
                        zotone(event.xbutton.x, event.xbutton.y));
                   break;
    case FocusIn:  /* Sense resizing final for repaint WM (kde, gnome) */
                NoFillFace = 0;
                putchar('F'); fflush(stdout);
                regraph(dpy, win, gc, NoFillFace);
                reset(dpy, win, NFACE); /* uncomment for WMaker */
                break;
    case KeyPress: /* Check KB key pressed */
                end = rekey(dpy, win, gc, &event);
                           break;
     default: break;
  } /* switch */
} /* while */
return(end);
} /* dispath */

int main(int argc, char* argv[]) {
XPolyGraph heap;              /* polytop graph structure */
Display* dpy;             /* X display structure */
Window win;               /* window id */
GC gc;                   /* Graphic Context */
dpy = XOpenDisplay(NULL); /* Connect with X-server */
```

```
        assoc(&heap);           /* Associate heap for display functions */
        relink(&heap);          /* Link heap for configure functions */
        retrace();          /* trace face & edge path to assembly in graph */
        colorite(dpy);      /* allocate WRGB color palette */
        gc = congraph(dpy);     /* Create & custom Graphic Context */
        win = wingraph(dpy);    /* Create & custom programm window */
        XStoreName(dpy, win, argv[0]); /* print window tittle */
        dispatch(dpy, win, gc);      /* dispatch event queue */
        XDestroyWindow(dpy, win);     /* Close programm window */
        XCloseDisplay(dpy);         /* disconnect from X-server */
        return(0);
} /* main */
```

6. 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..