Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное общеобразовательное учреждение высшего образования «Московский государственный технический университет имени Н. Э. Баумана

(национальный исследовательский университет)»

(МГТУ им. Н. Э. Баумана)

Факультет «Робототехника и комплексная автоматизация»

Кафедра «Системы автоматизированного проектирования»

Лабораторная работа №2 по дисциплине

«Программирование графических приложений»

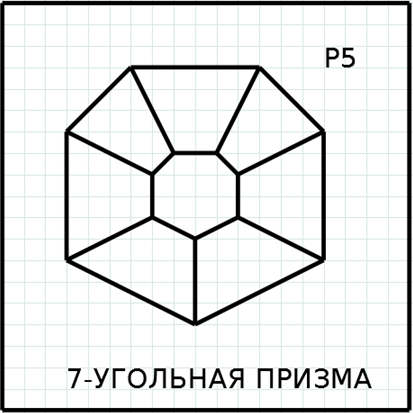
Выполнил: студент группы РК6-42Б Журавлев Н. В.

Проверил: к. т. н., доцент кафедры РК6 Волосатова Т. М.

Москва, 2021

**Текст задания**

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 n=4 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.



**Описание алгоритма**

1. 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.
2. 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).
3. 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.
4. 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.
5. 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.

**Текст программы**

**xheap.h**

/\* Paint heap graph face header file \*/

#define NFACE7 1

#define NFACE3 7 /\* 3 top face number \*/

#define NFACE (NFACE3 + NFACE7) /\* all face number \*/

#define NPATH4 7 /\* 4 edge path number \*/

#define NPATH NPATH4 /\* all edge path number \*/

#define NEDGE 8 /\* edge number \*/

#define NVERT 14 /\* vertex number \*/

#define NCOLOR 4 /\* face color number \*/

#define NCELL 14 /\* unit cell number by X & Y \*/

#define LWIDTH 2 /\* contour line width \*/

#define DOT 8 /\* vertex dot size \*/

#define DEFTONE 2 /\* default face tone \*/

/\* Face declaratiom \*/

typedef struct {

XPoint\* top; /\* face vertex location array \*/

int Cn; /\* face vertex number \*/

int tone; /\* face tone color index \*/

Region regi; /\* face space region \*/

} XFace;

/\* Edge path declaration \*/

typedef struct {

XPoint\* node; /\* edge path vertex location array \*/

int Pn; /\* edge path vertex number \*/

} XPath;

/\* Polytop graph declaration \*/

typedef struct {

XPoint\* vertex; /\* vertex array \*/

XFace\* face; /\* face array \*/

XPath\* edge; /\* edge path array \*/

} XGraph;

/\* Graph display function declaration (xheap2) \*/

GC MakeGC(Display\*);

Window WinMain(Display\*);

int colorite(Display\*);

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

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

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

int rekey(Display\*, Window, GC, XEvent\*);

int assoc(XGraph\*);

/\* Graph configure function declaration (xheap1) \*/

int relink(XGraph\*);

int resize(unsigned, unsigned);

int rescale(unsigned, unsigned);

int rebuild();

int retrace();

int reconf(unsigned, unsigned);

int rereg();

int inface(int, int);

/\* Dispatch function declaration (xheap0) \*/

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

**xheap0.c**

/\* Paint polytop graph main & dispatch \*/

#include <X11/Xlib.h>

#include <X11/Xutil.h>

#include <stdio.h>

#include "xheap.h"

/\* Event Dispatcher \*/

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,

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

regraph(dpy, win, gc, NoFillFace);

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 \*/

/\* Main function \*/

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

XGraph 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 = MakeGC(dpy); /\* Create & custom Graphic Context \*/

win = WinMain(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 \*/

**xheap1.c**

/\* Graph configure functions \*/

/\* Change retrace() & rebuild() code for other graph! \*/

#include <X11/Xlib.h>

#include <X11/Xutil.h>

#include "xheap.h"

static XPoint face7[NFACE7][(7+1)]; /\* 3-top faces top location \*/

static XPoint face3[NFACE3][(3+1)]; /\* 3-top faces top location \*/

static XPoint path4[NPATH4][4]; /\* 4-edge path tops location \*/

static XPoint scale; /\* scale (pixel/cell) for x & y \*/

static XPoint\* vertex; /\* vertex location array address \*/

static XFace\* face; /\* face array address \*/

static XPath\* edge; /\* edge path array address \*/

/\* Link with pass Graph structure \*/

int relink(XGraph \*ph) {

vertex = ph->vertex;

edge = ph->edge;

face = ph->face;

return(0);

} /\* relink \*/

/\* Check Resize window when configure event \*/

int resize(unsigned w, unsigned h) {

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

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

return(0); /\* remove window \*/

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

return(NFACE); /\* resize window \*/

} /\* resize \*/

/\* Check window scale when resize \*/

int rescale(unsigned w, unsigned h) {

int x, y; /\* pixel for cell by x & y \*/

x = w / NCELL; y = h / NCELL;

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

return(0); /\* small resize without change scale \*/

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

return(NFACE); /\* change scale \*/

} /\* rescale \*/

/\* Rebuild graph configuration \*/

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

static XPoint vconf[] = { /\* vertex location in x,y cells \*/

{1, 4}, {1, 10}, {7, 13}, {13, 10}, {13, 4}, {10, 1}, {4, 1},

{6, 5}, {8, 5}, {9, 6}, {9, 8}, {7, 9}, {5, 8}, {5, 6}

}; /\* vconf \*/

static int fconf3[NFACE3][(4+1)] = { /\* Vertex index \*/

{0, 1, 12, 13, 0}, /\* for 3-top faces \*/

{1, 2, 11, 12, 1}, /\* (begin = end vertex) \*/

{2, 3, 10, 11, 2},

{10, 3, 4, 9, 10},

{9, 4, 5, 8, 9},

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

{0, 13, 7, 6, 0}

}; /\* fconf3 \*/

static int fconf7[NFACE7][(7+1)] = { /\* Vertex index \*/

{7, 8, 9, 10, 11, 12, 13, 7}

}; /\* fconf3 \*/

static int pconf4[NPATH4][4] = { /\* Vertex index \*/

{1, 0, 13, 12}, /\* for 4-edge path \*/

{2, 1, 12, 11},

{3, 2, 11, 10},

{4, 3, 10, 9},

{5, 4, 9, 8},

{6, 5, 8, 7},

{0, 6, 7, 13}

}; /\* edge 4 path \*/

int i, j; /\* vertex index \*/

for(i = 0; i < NVERT; i++) { /\* compute vertex pixel location \*/

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

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

} /\* for-vertex \*/

for(i = 0; i < NPATH4; i++) /\* store vertex pixel location \*/

for(j = 0; j < 4; j++) { /\* for 4-edge path \*/

path4[i][j].x = vertex[pconf4[i][j]].x;

path4[i][j].y = vertex[pconf4[i][j]].y;

} /\* for 4-edge path \*/

for(i = 0; i < NFACE3; i++) /\* store vertex pixel location \*/

for(j = 0; j < (4+1); j++) { /\* for 3-top faces \*/

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

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

} /\* for 3-top face \*/

for(i = 0; i < NFACE7; i++) /\* store vertex pixel location \*/

for(j = 0; j < (7+1); j++) { /\* for 3-top faces \*/

face7[i][j].x = vertex[fconf7[i][j]].x;

face7[i][j].y = vertex[fconf7[i][j]].y;

} /\* for 3-top face \*/

return(0);

} /\* rebuild \*/

/\* Trace face & edge path array \*/

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

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

int p = 0; /\* total edge path index \*/

int i; /\* n-edge path index \*/

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

for(j = 0; j < NFACE3; j++) { /\* fix 3-top faces in face array \*/

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

face[f].Cn = 4; /\* fix 3-top face top number=3 \*/

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

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

f++;

} /\* face3 \*/

for(j = 0; j < NFACE7; j++) { /\* fix 3-top faces in face array \*/

face[f].top = face7[j]; /\* fix 3-top face array address \*/

face[f].Cn = 7; /\* fix 3-top face top number=3 \*/

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

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

f++;

} /\* face3 \*/

face[f].tone = DEFTONE; /\* store extern face default tone color \*/

for(i = 0; i < NPATH4; i++) { /\* fix 4-edge path in edge path array \*/

edge[p].node = path4[i]; /\* fix 4-edge path array address \*/

edge[p].Pn = 4; /\* fix 4-edge path vertex number=4 \*/

p++;

} /\* path4 \*/

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);

} /\* reconf \*/

/\* Change face region when window resize & rescale \*/

int rereg() {

int i; /\* region & face index \*/

static XPoint prescale = {0, 0}; /\* previos scale \*/

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

return(0); /\* return, when no change scale \*/

for(i=0; i < NFACE; i++) { /\* New region for face \*/

XDestroyRegion(face[i].regi);

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

} /\* for \*/

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

return(i);

} /\* rereg \*/

/\* Check pointed face \*/

int inface(int x, int y) {

int f; /\* face index \*/

rereg(); /\* change face region if need \*/

for(f = 0; f < NFACE; f++) /\* check region with (x,y) inside \*/

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

break;

face[f].tone = (face[f].tone + 1) % NCOLOR; /\* new face tone \*/

return(f); /\* return pointed face for repaint \*/

} /\* inface \*/

**xheap2.c**

/\* Graph Display funcrion \*/

#include <X11/Xlib.h>

#include <X11/Xutil.h>

#include <X11/keysymdef.h>

#include "xheap.h"

/\* Polytop Graph space allocation \*/

static XPoint vertex[NVERT];

static XFace face[NFACE+1];

static XPath edge[NEDGE];

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

/\* Associate graph address with graph space \*/

int assoc(XGraph\* ph) {

ph->vertex = vertex;

ph->edge = edge;

ph->face = face;

return(0);

} /\* assoc \*/

/\* Allocate color palette \*/

int colorite(Display\* dpy) {

Colormap cmap; /\* color map \*/

XColor rgb; /\* Structure for color allocation \*/

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

static char\* spector[] = {

"#ffffff", /\* white \*/

"#ff0000", /\* red \*/

"#00ff00", /\* green \*/

"#0000ff", /\* blue \*/

"#000000", /\* black \*/

"#ffff00", /\* yellow \*/

"#ff00ff", /\* purple \*/

"#008080", /\* teal \*/

"#00CED1",

(char\* ) 0

}; /\* color spector \*/

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

while(spector[i] != (char\*) 0) {

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

XAllocColor(dpy, cmap, &rgb);

palette[i] = rgb.pixel;

i++;

} /\* while \*/

return(0);

} /\* colorite \*/

/\* Set Graphic Context \*/

GC MakeGC(Display\* dpy) {

XGCValues gcval; /\* Graphic Contex parameters \*/

GC gc; /\* Graphic Context id \*/

gcval.line\_width = LWIDTH;

gcval.background = palette[DEFTONE];

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

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

return(gc);

} /\* MakeGC \*/

/\* Create main window \*/

Window WinMain(Display\* dpy) {

Window win; /\* return Window id \*/

XSetWindowAttributes attr; /\* window attributes \*/

XSizeHints hint; /\* connect with Window Maneger \*/

int x, y; /\* window location \*/

unsigned w, h; /\* window size \*/

int scr; /\* screen number \*/

attr.override\_redirect = False; /\* set window maneger control \*/

attr.background\_pixel = palette[DEFTONE]; /\* background color \*/

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

StructureNotifyMask | FocusChangeMask); /\* Event mask \*/

scr = DefaultScreen(dpy);

//w = DisplayWidth(dpy, scr) / 2;

//h = DisplayHeight(dpy, scr) / 2;

w = 800;

h = 800;

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 \* DOT; /\* window min size \*/

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

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

XSetNormalHints(dpy, win, &hint);

XMapWindow(dpy, win);

return(win);

} /\* WinMain \*/

/\* Redraw graph with paint face or contour only \*/

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

int i; /\* face, edge path or vertex count \*/

for(i = NoFillFace; i < NFACE; i++) { /\* redraw all faces \*/

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

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

Convex, CoordModeOrigin);

} /\* for face \*/

XSetForeground(dpy, gc, palette[NCOLOR]); /\* BlackPixel color\*/

for(i = 0; i < NPATH; i++) /\* redraw face contour \*/

XDrawLines(dpy, win, gc, edge[i].node,

edge[i].Pn, CoordModeOrigin);

for(i=0; i < NVERT; i++) /\* redraw even vertex \*/

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

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

return(0);

} /\* regraph \*/

/\* Reset graph with default or current face's color \*/

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

int i; /\* face index \*/

for(i = f; i < NFACE; i++) /\* set default tone for inner faces \*/

face[i].tone = DEFTONE;

if(f == 0) /\* set default tone for extern face \*/

face[i].tone = DEFTONE;

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

XClearArea(dpy, win, 0, 0, 0, 0, True); /\* Expose Event \*/

return(f);

} /\* reset \*/

/\* Redraw one face \*/

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

int i; /\* face's vertex count \*/

if(f == NFACE) /\* redraw extern face \*/

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

XSetForeground(dpy, gc, palette[face[f].tone]); /\* inner face color \*/

XFillPolygon(dpy, win, gc, face[f].top, face[f].Cn, /\* redraw inner \*/

Convex, CoordModeOrigin); /\* face space \*/

XFlush(dpy);

XSetForeground(dpy, gc, palette[NCOLOR]); /\* BlackPixel color \*/

XDrawLines(dpy, win, gc, face[f].top, face[f].Cn + 1, /\* redraw inner \*/

CoordModeOrigin); /\* face contour \*/

for(i = 0; i < face[f].Cn; i++) /\* redraw inner face's vertex \*/

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

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

return(0);

} /\* reface \*/

/\* Recognize & process key pressed \*/

int rekey(Display\* dpy, Window win, GC gc, XEvent\* ev) {

KeySym sym; /\* Key symbol code \*/

sym = XKeycodeToKeysym(dpy, ev->xkey.keycode, 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 \*/

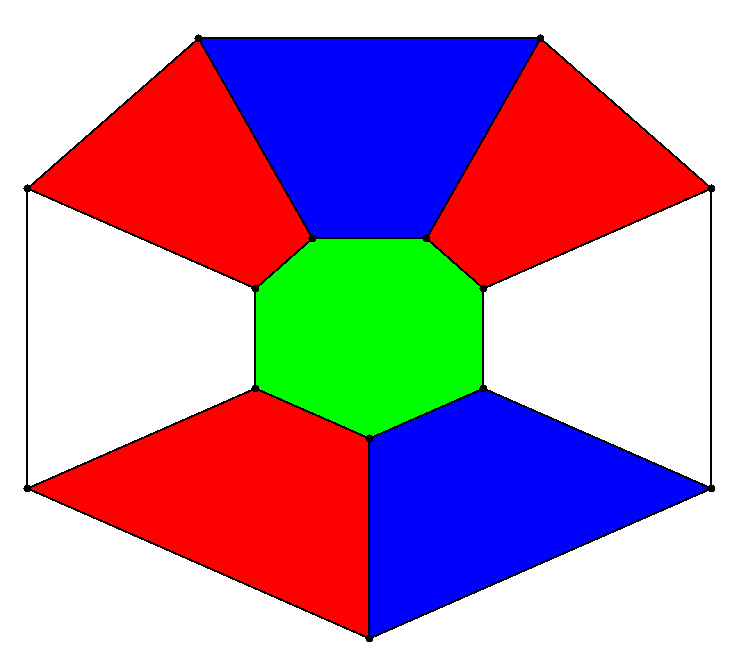
if(sym == XK\_Escape) /\* Press Escape to redraw default face tone \*/

reset(dpy, win, 0);

return(0); /\* continue return for event dispatcher in main() \*/

} /\* rekey \*/

**Результат работы программы**



**Список литературы**

1. Разработка графических приложений в среде X Window System -

Д. Е. Беломойцев, Т. М. Волосатова, С. В. Родионов

1. Программирование в среде X Window на основе библиотеки Xlib - http://dfe3300.karelia.ru/koi/posob/X/index.html