**МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

**(ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)**

**«МАИ»**

**Компьютерная графика. Лабораторный практикум**

Учебное пособие

Утверждено

на заседании редсовета

\_\_\_ \_\_\_\_\_2022

Москва

Издательство МАИ-ПРИНТ

2022

**УДК 681.3**

Компьютерная графика. Лабораторный практикум: Учебное пособие.— М.: Из-во МАИ-ПРИНТ, 2022. — 96 с.

Содержание учебного пособия включает краткие теоретические сведения по компьютерной графике и геометрическому моделированию. Рассмотрены вопросы аффинных преобразований, удаления скрытых линий и поверхностей, а таже закраски граней. Приведена методика выполнения лабораторных работ.

Пособие предназначено для студентов старших курсов МАИ, направления: 09.03.01 «Информатика и вычислительная техника» — профили: 09.03.01.Б4 «Вычислительные машины, комплексы, системы и сети», 09.03.01 Б5 «Программное обеспечение средств вычислительной техники и автоматизированных систем»; 09.03.04 «Программная инженерия» — профиль: 09.03.04.Б1 «Программно-информационные системы», обучающихся по дисциплине «Компьютерная графика».

Выполнение работ ориентировано на применение ЭВМ и требует навыков программирования, а также знаний в области линейной алгебры и аналитической геометрии.

Рецензент:

|  |  |  |
| --- | --- | --- |
| **ISBN** ‑ ‑ ‑ |  | Машкин М.Н., 2022 |
|  |  |  |

# Предисловие

Лабораторный практикум по дисциплине «Компьютерная графика» содержит четыре лабораторных работы:

1. Аффинные преобразования.

2. Удаления невидимых линий методом Z-буфера для многогранников.

3. Удаления невидимых линий методом Вейлера-Азертона.

4. Закраска поверхностей тел методом Гуро.

Все четыре работы соответствуют лекционному курсу и обеспечивают практическое закрепление теоретического материала.

Основной подход при выполнении лабораторных работ базируется на постановке задач построения геометрических моделей с последующей разработкой программы на основе существующих алгоритмов машинной графики, обеспечивающих визуализацию на экране компьютера.

# Работа 1. Аффинные преобразования

**Цель работы:** изучить аффинные преобразования координат на примере выпуклых тел.

## 1.1. Порядок выполнения работы

1. Ознакомиться с матричными операциями по преобразованию координат методами проективной геометрии.
2. Рассчитать координаты точек многогранников №1 и №2.
3. Написать программу преобразования координат, обеспечивающую вращение, растяжение, отражение и перенос.
4. Выполнить указанные преобразования над многогранниками.
5. Составить отчёт, который должен содержать:

— задание и рисунки многогранников,

— результаты расчёта вершин многогранников,

— результаты аффинных преобразований координат вершин многогранников,

— выводы о проделанной работе.

— текст программы.

1. Выводы о проделанной работе.

## 1.2. Методические указания

**Аффинные преобразования** (от лат. affinis «соприкасающийся, близкий, смежный») — отображение плоскости или пространства в себя, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся — в пересекающиеся, скрещивающиеся — в скрещивающиеся. *Аффинные преобразования — биекция евклидова пространства или плоскости в себя, отображающая параллельные прямые в параллельные прямые.*

С математической точки зрения решение задачи графических построений геометрическое моделирование сводится к аффинным преобразованию декартовой прямоугольной системы координат на плоскости и пространстве.

Предположим, что на плоскости введена прямолинейная координатная система. Тогда каждой точке *М* ставится в соответствие упорядоченная пара чисел (*x, у*) её координат (рис.1.1). Вводя на плоскости ещё одну прямолинейную систему координат, мы ставим в соответствие той же точке М другую пару чисел (*x\*, y\**).

|  |
| --- |
|  |
| *Рис.1.1.* |

Переход от одной прямолинейной координатной системы на плоскости к другой описывается следующими соотношениями

*x\* = x + y + ,*

*y\* = x + y + ,* (1.1)

где *, , , * произвольные числа, но

. (1.2)

Начнём рассмотрение с нескольких частных случаев, считая для простоты, что заданная система координат является прямоугольной декартовой.

A. Поворот (вокруг начальной точки на угол **) (рис. 1.2) описывается формулами:

|  |
| --- |
| , (1.3) |
| . |

Б. Растяжение (сжатие) вдоль координатных осей можно задать так

*x\* = x,*

*y\* = y.*  (1.4)

* > 0,  > 0*

Растяжение (сжатие) вдоль оси абсцисс обеспечивается при условии, что ** > 1 (** < 1 соответственно). На рис. 1.3 ** = ** > 1.

|  |  |
| --- | --- |
|  |  |
| *Рис. 1.2.* | *Рис. 1.3.* |
|  |  |
| *Рис. 1.4.* | *Рис. 1.5.* |

В. Отражение (относительно оси абсцисс) (рис. 1.4) задаётся при помощи формул

*x\* = x*  *,*  (1.5)

*y\* = -y*.

Г. На рис. 1.5 вектор переноса *ММ\** имеет координаты ** и **. Перенос обеспечивают соотношения

*x\* = x + * *,* (1.6)

*y\* = y + *.

Любое преобразование (1.1) всегда можно представить как последовательное исполнение (суперпозицию) простейших преобразований вида А, Б, В и Г (или части этих преобразований).

Значит, и любое отображение вида (1.1) можно описать при помощи отображений, задаваемых формулами А, Б, В и Г.

Для эффективного использования этих известных формул более удобной является их матричная запись. Матрицы, соответствующие случаям А, Б и В; строятся легко. Они имеют соответственно следующий вид:

. (1.7)

Для охвата матричным подходом всех четырёх простейших преобразования (в том числе и перенос), переходят к описанию произвольной точки плоскости уже не парой чисел, а тройкой.

Для этого применяют однородные координаты произвольной точки, используемые в проективной геометрии

Пусть М - произвольная точка плоскости с координатами *x* и *у*. Однородными координатами точки называется тройка одновременно неравных нулю чисел *x*1, *x*2, *x*3, если выполнены следующие соотношения

, (1.8)

|  |
| --- |
|  |
| *Рис. 1.6.* |

При построениях геометрического моделирования однородные координаты обычно вводятся так: точке *M’*(*x, у*) ставится в соответствие точка *M*(*x, у,*1) в пространстве (рис. 1.6). Заметим, что произвольная точка на прямой, соединяющей начало координат, точку *O(0, 0, 0),* с точкой *М’(x, у,1),* может быть задана тройкой

*(hx, hy, h)*

Исключая точку *O* из рассмотрения, будем считать, что *h0*.

Вектор, определяемый тройкой *hx, hy, h* является направляю­щим вектором прямой, соединяющей точки *O* и *М’*. Эта прямая пересекает плоскость *z=1* в точке *(x, у, 1),* которая однозначно определяет точку *(x, у)* координатной плоскости *xОу*. Тем самым между точкой *(x, у)* и множеством троек *(hx, hy, h)*, *h  0* устанавливается взаимно однозначное соответствие. Это позволяет считать числа *hx, hy, h* ее координатами.

Как уже говорилось выше, такие координаты называются однородными. В проективной геометрии для них принято следующее обозначение

*x*; *y*; *1*

или, более общо,

*x1*; *x2*; *x3*

(напомним, что здесь непременно требуется, чтобы числа *x*1, *x*2, *x*3, одновременно в нуль не обращались).

Считая *h =1*, сравним две записи: (1.1) и нижеследующую, матричную:

(1.9)

Нетрудно заметить, что после перемножения выражений, стоящих в правой части последнего соотношения, мы получим две формулы (1.1) и верное числовое равенство 1 = 1.

Тем самым эти две сравниваемые записи можно считать равносильными.

Иногда в литературе используется другая запись - запись по столбцам

. (1.10)

Эта запись эквивалентна приведённой выше записи по строкам.

Чтобы реализовать то или иное отображение по заданным геометрическим характеристикам, надо найти элементы соответ­ствую­щей матрицы. Обычно её построение разбивают на этапы. На каждом этапе строится матрица для одного из выделенных выше случаев А, Б, В или Г. Выпишем эти матрицы.

A. Матрица вращения (rotation)

, (1.11)

Б. Матрица растяжения (сжатия) (dilatation)

, (1.12)

В. Матрица отражения (reflection)

, (1.13)

Г. Матрица переноса (translation)

. (1.14)

В этом случае матрица поворота вокруг точки А(a, b) на угол  (рис. 1.7) будет иметь вид

. (1.15)

Или в виде полного преобразования для данного случая

|  |  |
| --- | --- |
| . (1.16) | |
|  |  |
| Рис. 1.7. | Рис.1.8. |

Если необходимо растяжение с коэффициентами растяжения ** вдоль оси абсцисс и ** вдоль оси ординат и с центром в точке А(a, b) (рис. 1.8), то будем иметь матрицу

, (1.17)

или в развернутом виде

. (1.18)

Для преобразований в трёхмерном пространстве используют также однородные координаты.

Поступая аналогично тому, как это было сделано в размерности два, заменим тройку (x, y, z), задающую точку в пространстве, на четвёрку

(x, y, z, 1)

или, более общо, на

(hx, hy, hz, h),   h  0.

Тем самым каждая точка пространстве (кроме начальной точки O) может быть задана четвёркой одновременно не равных нулю чисел, эта четвёрка определена однозначно с точностью до общего множителя.

Предложенный переход даёт возможность воспользоваться матричной записью и в более сложных, трёхмерных задачах.

Как известно, любое аффинное преобразование в трёхмерном пространстве может быть представлено в виде суперпозиции вращений, растяжений, отражений и переносов. Поэтому достаточно подробно описать матрицы только этих последних преобразований.

A. Матрицы вращения в пространстве.

Матрица вращения вокруг оси абсцисс на угол :

. (1.19)

Матрица вращения вокруг оси ординат на угол :

. (1.20)

Матрица вращения вокруг оси аппликат на угол :

. (1.21)

Б. Матрица растяжения (сжатия):

, (1.22)

здесь  > 0 - коэффициент растяжения (сжатия) вдоль оси абсцисс,

 > 0 - коэффициент растяжения (сжатия) вдоль оси ординат,

 > 0 - коэффициент растяжения (сжатия) вдоль оси аппликат.

В. Матрицы отражения.

Матрица отражения относительно плоскости xОу:

. (1.23)

Матрица отражения относительно плоскости уОz:

. (1.24)

Матрица отражения относительно плоскости zОx:

. (1.25)

Г. Матрица переноса:

; (1.26)

здесь (, , ) - вектор переноса.

Заметим, что, как и в двумерном случае, все выписанные матрицы невырождены.

В случае вращения объекта или точки на угол ** вокруг прямой L (рис. 1.9), проходящей через точку *А(a, b, c)* и имеющую направляющий вектор *(l, m, n)* будем иметь матрицу

|  |
| --- |
|  |
| Рис. 1.9. |

[Т][Rx][Ry][Rz][Ry]-1[Rx]-1[Т]-1. (1.27)

Рассматривая другие подобные примеры, будем получать в результате невырожденные матрицы вида

. (1.28)

При помощи этих матриц можно подвергать простейшим преобразованиям любые плоские и пространственные фигуры. Возьмём, к примеру, выпуклый многогранник (элемент трёхмерной модели манекена). Он однозначно задаётся массивом своих вершин. Подействовав на этот массив найденной матрицей, мы получаем новый массив, задающий выпуклый многогран­ник - образ исходного (рис.1.10).

|  |
| --- |
|  |
| Рис. 1.10. |

## 1.3. Пример программы аффинных преобразований вершин кристаллов

Программный комплекс для выполнения лабораторных работ написан на языке С++ в среде разработки Qt и состоит из четырёх программ: лабораторных работ №1 – 4.

Программа лабораторной работы №1 содержит описание классов: **Cristall** для описания главной экранной формы и **Frame** для описания кристалла**.** Файл «cristall.h» содержит описания структуры класса, файл «cristall.cpp» содержит описания функций обработки событий в окне.Файл «Frame.h» содержит описания структуры класса, файл «Frame.cpp» содержит описания функций обработки кристала

Ввод исходных данных кристаллов выполняется из текстовых файлов в структуру, описанную в классе Frame при помощи встроенной функции file.readLine().

Аффинные преобразования выполняются при помощи функций: Frame::rotateZLeft(), Frame::rotateZRight(),Frame::rotateX(),Frame::rotateY() – вращение; Frame::scaleX(double scaleValue), Frame::scaleY(double scaleValue), scaleZ(double scaleValue) – масштабирование, Frame::moveToCoord(double posX, double posY, double posZ) – перемещение; Frame::reflectX(),Frame::reflectY(),Frame::reflectZ() - отражение

Вывод на экран – встроенная функция repaint() и функция Frame::drawFigure() класса Frame

### 1.3.1. Файл «cristall.cpp»

#include "cristall.h"

#include "ui\_cristall.h"

Cristall::Cristall(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::Cristall)

{

ui->setupUi(this);

connect(ui->addPolygons, SIGNAL(triggered()), this, SLOT(slotAddPolygons()));

connect(ui->addPoints, SIGNAL(triggered()), this, SLOT(slotAddPoints()));

connect(ui->uploadFiles, SIGNAL(clicked()), this, SLOT(slotUploadFiles()));

connect(ui->exit, SIGNAL(triggered()), this, SLOT(slotExit()));

connect(ui->FiX, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

connect(ui->FiY, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

connect(ui->FiZ, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

canvas = new Frame(this);

ui->canvasLayout->addWidget(canvas);

}

Cristall::~Cristall()

{

canvas->deleteLater();

delete ui;

}

void Cristall::slotAddPolygons()

{

QString tmp = QFileDialog::getOpenFileName(0, "Choose file with polygons", "", "\*.txt");

if (tmp == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл!");

return;

}

else

{

pathPolygons = tmp;

ui->uiPathPolygons->setText(pathPolygons);

ui->uploadFiles->setText("Загрузить");

}

}

void Cristall::slotAddPoints()

{

QString tmp = QFileDialog::getOpenFileName(0, "Choose file with points", "", "\*.txt");

if (tmp == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл!");

return;

}

else

{

pathPoints = tmp;

ui->uiPathPoints->setText(pathPoints);

ui->uploadFiles->setText("Загрузить");

}

}

void Cristall::slotUploadFiles()

{

if (pathPolygons == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл полигонов!");

return;

}

if (pathPoints == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл вершин!");

return;

}

bool resultUpload = canvas->upload(pathPolygons, pathPoints);

if (resultUpload)

{

ui->uploadFiles->setText("Сбросить");

}

}

void Cristall::slotExit()

{

this->close();

}

void Cristall::slotChangeAngle()

{

canvas->setFiX(ui->FiX->text().toDouble());

canvas->setFiY(ui->FiY->text().toDouble());

canvas->setFiZ(ui->FiZ->text().toDouble());

}

void Cristall::on\_goStepX\_clicked()

{

canvas->rotateX();

}

void Cristall::on\_goStepY\_clicked()

{

canvas->rotateY();

}

void Cristall::on\_goStepZ\_clicked()

{

canvas->rotateZ("Right");

}

void Cristall::on\_goScaleX\_clicked()

{

canvas->scaleX(ui->scaleX->text().toDouble());

}

void Cristall::on\_goScaleY\_clicked()

{

canvas->scaleY(ui->scaleY->text().toDouble());

}

void Cristall::on\_goScaleZ\_clicked()

{

canvas->scaleZ(ui->scaleZ->text().toDouble());

}

void Cristall::on\_goMove\_clicked()

{

canvas->moveToCoord(ui->posX->text().toDouble(), ui->posY->text().toDouble(), ui->posZ->text().toDouble());

}

void Cristall::on\_goReflectX\_clicked()

{

canvas->reflectX();

}

void Cristall::on\_goReflectY\_clicked()

{

canvas->reflectY();

}

void Cristall::on\_goReflectZ\_clicked()

{

canvas->reflectZ();

}

### 1.3.2. Файл «Frame.cpp»

#include "Frame.h"

Frame::Frame(QWidget \*parent) : QWidget(parent)

{

QPen pen(Qt::black, 1, Qt::DashLine, Qt::SquareCap, Qt::RoundJoin);

painter.setPen(pen);

}

/\* ------------------ Draw Figure ------------------ \*/

void Frame::paintEvent(QPaintEvent \*event)

{

Q\_UNUSED(event);

painter.begin(this);

drawFigure();

painter.end();

}

void Frame::drawFigure()

{

for (int i = 0; i < dataPolygons.size(); i++)

{

QPointF points[dataPolygons[i].size()];

for (int j = 0; j < dataPolygons[i].size(); j++)

{

points[j] = QPointF(dataPoints[dataPolygons[i][j]].x + 250, dataPoints[dataPolygons[i][j]].y + 250);

}

painter.drawPolygon(points, dataPolygons[i].size());

}

}

/\* ------------------ Figure operations------------------ \*/

void Frame::mouseMoveEvent(QMouseEvent \*event)

{

if (!\_p.isNull())

{

if (abs(rotationX - (event->x() - 250)) > 5 && abs(rotationY - (event->y() - 250)) > 5)

{

if (rotationX > (event->x() - 250))

{

if (rotationY > (event->y() - 250))

{

FiX = abs(FiX);

rotateX();

if ( FiY > 0) FiY = -FiY;

rotateY();

}else

{

if ( FiX > 0) FiX = -FiX;

rotateX();

if ( FiY > 0) FiY = -FiY;

rotateY();

}

}else

{

if (rotationY > (event->y() - 250))

{

FiX = abs(FiX);

rotateX();

FiY = abs(FiY);

rotateY();

}else

{

if ( FiX > 0) FiX = -FiX;

rotateX();

FiY = abs(FiY);

rotateY();

}

}

rotationX = event->x() - 250;

rotationY = event->y() - 250;

}

}

}

// Rotation

void Frame::rotateZLeft()

{

rotateZ("Left");

}

void Frame::rotateZRight()

{

rotateZ("Right");

}

void Frame::rotateX()

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, cos(FiX), sin(FiX), 0,

0, -sin(FiX), cos(FiX), 0,

0, 0, 0, 1

);

calculate(R);

repaint();

}

void Frame::rotateY()

{

QMatrix4x4 R

(

cos(FiY), 0, -sin(FiY), 0,

0, 1, 0, 0,

sin(FiY), 0, cos(FiY), 0,

0, 0, 0, 1

);

calculate(R);

repaint();

}

void Frame::rotateZ(QString diraction)

{

if (diraction == "Left")

{

FiZ = abs(FiZ);

}else if(diraction == "Right")

{

if ( FiZ > 0) FiZ = -FiZ;

}

QMatrix4x4 R

(

cos(FiZ), sin(FiZ), 0, 0,

-sin(FiZ), cos(FiZ), 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(R);

repaint();

}

// Scale

void Frame::scaleX(double scaleValue)

{

QMatrix4x4 R

(

scaleValue, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(R);

repaint();

}

void Frame::scaleY(double scaleValue)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, scaleValue, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(R);

repaint();

}

void Frame::scaleZ(double scaleValue)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, scaleValue, 0,

0, 0, 0, 1

);

calculate(R);

repaint();

}

// Move

void Frame::moveToCoord(double posX, double posY, double posZ)

{

QMatrix4x4 R

(

1, 0, 0, posX,

0, 1, 0, posY,

0, 0, 1, posZ,

0, 0, 0, 1

);

calculate(R);

repaint();

}

// Reflect

void Frame::reflectX()

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, -1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(R);

repaint();

}

void Frame::reflectY()

{

QMatrix4x4 R

(

-1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(R);

repaint();

}

void Frame::reflectZ()

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, -1, 0,

0, 0, 0, 1

);

calculate(R);

repaint();

}

void Frame::calculate(QMatrix4x4 &R)

{

for (int i = 0; i < dataPoints.size(); i++)

{

QVector4D vectCoord(dataPoints[i].x, dataPoints[i].y, dataPoints[i].z, 1);

QVector4D result = R \* vectCoord;

dataPoints[i].x = result[0];

dataPoints[i].y = result[1];

dataPoints[i].z = result[2];

}

}

/\* ------------------ Upload data for figure ------------------ \*/

bool Frame::upload(QString initPathPolygons, QString initPathPoints)

{

pathPolygons = initPathPolygons;

pathPoints = initPathPoints;

bool resultFillPolygons = fillingDataPolygons();

bool resultFillPoints = fillingDataPoints();

repaint();

if (resultFillPolygons && resultFillPoints) return true;

}

bool Frame::fillingDataPolygons()

{

dataPolygons.clear();

QFile file(pathPolygons);

if(!file.open(QIODevice::ReadOnly))

{

QMessageBox::warning(this, "Предупреждение!", "Файл " + pathPolygons + " не найден!");

return false;

}

while (!file.atEnd())

{

QStringList list;

QString tmpStr = file.readLine();

list = tmpStr.split(QRegExp(" "));

QVector<int> tmpVec;

for (int i = 1; i <= list.at(0).toInt(); i++)

{

tmpVec.push\_back(list.at(i).toInt());

}

dataPolygons.push\_back(tmpVec);

}

file.close();

return true;

}

bool Frame::fillingDataPoints()

{

dataPoints.clear();

QFile file(pathPoints);

if(!file.open(QIODevice::ReadOnly))

{

QMessageBox::warning(this, "Предупреждение!", "Файл " + pathPoints + " не найден!");

return false;

}

while (!file.atEnd())

{

QStringList list;

QString tmp = file.readLine();

list = tmp.split(QRegExp(" "));

dataPoints.push\_back({100 \* list.at(0).toFloat(), 100 \* list.at(1).toFloat(), 100 \* list.at(2).toFloat()});

}

\

file.close();

return true;

}

### 1.3.3. Файл «cristall.h»

#ifndef CRISTALL\_H

#define CRISTALL\_H

#include <QMainWindow>

#include <QFileDialog>

#include <QMessageBox>

#include <QString>

#include <Frame.h>

QT\_BEGIN\_NAMESPACE

namespace Ui { class Cristall; }

QT\_END\_NAMESPACE

class Cristall : public QMainWindow

{

Q\_OBJECT

public:

Cristall(QWidget \*parent = nullptr);

~Cristall();

private slots:

void slotAddPolygons();

void slotAddPoints();

void slotUploadFiles();

void slotExit();

void slotChangeAngle();

void on\_goStepX\_clicked();

void on\_goStepY\_clicked();

void on\_goStepZ\_clicked();

void on\_goScaleX\_clicked();

void on\_goScaleY\_clicked();

void on\_goScaleZ\_clicked();

void on\_goMove\_clicked();

void on\_goReflectX\_clicked();

void on\_goReflectY\_clicked();

void on\_goReflectZ\_clicked();

private:

Ui::Cristall \*ui;

QString pathPolygons;

QString pathPoints;

Frame \*canvas;

};

#endif // CRISTALL\_H

### 1.3.4. Файл «Frame.h»

#ifndef FRAME\_H

#define FRAME\_H

#include <QWidget>

#include <QPainter>

#include <QVector>

#include <QFile>

#include <QMessageBox>

#include <QtMath>

#include <QDebug>

#include <QMatrix4x4>

#include <QMatrix>

#include <QMouseEvent>

#include <QWheelEvent>

class Frame : public QWidget

{

Q\_OBJECT

public:

Frame(QWidget \* parent = 0);

bool upload(QString, QString);

void drawFigure();

void setFiX(double initFiX){FiX = initFiX \* M\_PI / 180;};

void setFiY(double initFiY){FiY = initFiY \* M\_PI / 180;};

void setFiZ(double initFiZ){FiZ = initFiZ \* M\_PI / 180;};

// Rotation

void rotateX();

void rotateY();

void rotateZ(QString);

// Scale

void scaleX(double);

void scaleY(double);

void scaleZ(double);

// Move

void moveToCoord(double, double, double);

// Reflect

void reflectX();

void reflectY();

void reflectZ();

protected:

void paintEvent(QPaintEvent \*) override;

void mousePressEvent(QMouseEvent \*event) override

{

if (event->button() == Qt::LeftButton)

{

\_p = event->pos();

rotationX = \_p.x() - 250;

rotationY = \_p.y() - 250;

}

}

void mouseMoveEvent(QMouseEvent \*) override;

void wheelEvent(QWheelEvent \*event) override

{

if (event->delta() > 0)

{

rotateZLeft();

}

else

{

rotateZRight();

}

};

private:

QString pathPolygons;

QString pathPoints;

QPainter painter;

QPoint \_p;

double FiX = 0.07;

double FiY = 0.07;

double FiZ = 0.07;

int rotationX;

int rotationY;

struct coord{

float x, y, z;

};

QVector<QVector<int>> dataPolygons;

QVector<coord> dataPoints;

/\* --------- Functions --------- \*/

bool fillingDataPolygons();

bool fillingDataPoints();

void rotateZLeft();

void rotateZRight();

void calculate(QMatrix4x4 &);

};

#endif // FRAME\_H

### 1.3.5. Результаты работы программы.

Результаты работы программы приведены на рис. 1.11 — 1.19.

|  |  |
| --- | --- |
| https://sun9-52.userapi.com/impg/hBUlof4-o_0A3nN_DZMrFcL_NDvQbF962J9q_g/tyUKnA1Dq6Q.jpg?size=575x420&quality=96&proxy=1&sign=01c21e9afc097568e184d64e10776d0d |  |
| *Рис. 1.11. Кристалл 1* | *Рис. 1.12. Поворот по оси X* |

|  |  |
| --- | --- |
|  |  |
| *Рис. 1.13. Поворот по оси Y* | *Рис. 1.14. Поворот по оси Z* |

|  |  |
| --- | --- |
|  |  |
| *Рис. 1.15. Масштаб по оси X* | *Рис.1.16. Масштаб по оси Y* |
|  |  |
| *Рис. 1.17. Масштаб по оси Z* | *Рис. 1.18. Смещение по X,Y,Z* |

|  |
| --- |
|  |
| *Рис. 1.19. Отражение кристалла* |

## 1.4. Контрольные вопросы

1. Что является исходными данными для решения задачи аффинных преобразований?
2. Какие преобразования координат вами использовались для визуализации?
3. Какие методы построения поверхности объектов используются в геометрическом моделировании?
4. В чём преимущества однородных координат?
5. Какая геометрическая модель объекта визуализации использована в вашей программе?

# Работа 2. Удаления невидимых линий методом Z‑буфера для многогранников

**Цель работы:** написать программу, обеспечивающую удаление скрытых рёбер многогранника методом Z буфера, работающего в пространстве изображения

## 2.1. Порядок выполнения работы

1. Ознакомиться со способами удаления скрытых линий, работающих в пространстве изображения.
2. Написать программу удаления скрытых рёбер многогранника.
3. Выполнить удаление скрытых рёбер многогранников.
4. Составить отчёт, который должен содержать:

— задание и рисунки многогранников,

— описание алгоритма удаления скрытых линий методом Z‑буфера для многогранников,

— результаты работы программы удаления скрытых линий,

— выводы о проделанной работе.

— текст программы удаления скрытых линий,

1. Ответить устно на контрольные вопросы.

## 2.2. Методические указания

Алгоритмы удаления невидимых поверхностей определяют невидимые для пользователя части сцены и не используют их для отображения.

Многие алгоритмы ориентированы на специализированное применение. При выборе алгоритма учитываются количество объектов в сцене, их вид и способ расположения. Поэтому наилучшего решения задачи удаления скрытых линий и поверхностей не существует. Почти все алгоритмы удаления скрытых линий и поверхностей включают в себя сортировку по расстоянию от тела до точки наблюдения. Основная идея сортировки — чем дальше объект, тем больше вероятность, что он будет заслонен другим объектом.

Все алгоритмы удаления скрытых линий и поверхностей можно разделить на два типа.

1. Алгоритмы, работающие в объектном пространстве (ОП).

• Основной принцип этих алгоритмов — каждая из п граней сравнивается с оставшимися (n – 1) гранями.

• Они имеют дело с физической системой координат, в которой описаны эти объекты.

• Данные алгоритмы весьма точны (полученные изображения можно легко увеличить в несколько раз без потери качества).

• Объем вычислений теоретически — *n*2, где *n* — количество объектов.

2. Алгоритмы, работающие в пространстве изображения (ПИ).

• Основной принцип этих алгоритмов — нужно определить, какая из *n* граней видна в каждой точке экрана, то есть каждый объект сравнивается с каждым пикселом экрана.

• Они имеют дело с системой координат экрана.

• Точность ограничена разрешающей способностью экрана (если полученные результаты потом увеличивать во много раз, они не будут соответствовать исходному изображению).

• Объем вычислений теоретически — *N*´*n*, где *N* — количество пикселов на экране.

**Алгоритм, использующий Z-буфер**

Алгоритм, использующий Z-буфер, относится к алгоритмам, работающим в пространстве изображения.

Основной его принцип в том, что используются два буфера: регенерации, в котором хранятся значения пикселов, и *Z*-буфер, который хранит *z*-координаты.

Буфер регенерации заполняется значениями при параллельном анализе *z*-координаты со значениями *Z*-буфера.

Шаги алгоритма следующие.

1. В *Z*-буфер заносятся максимально возможные значения *z*.

2. Буфер регенерации заполняется значениями фона.

3. Каждый объект раскладывается в растр. Если координата *z* точки (*x*, *y*) меньше либо равна значению *Z*-буфера в элементе (*х*, *у*), то:

1) *z*(*x*, *у*) заносится в элемент (*x*, *у*) *Z*-буфера;

2) значение пиксела помещается в элемент (*х*, *у*) буфера регенерации.

Достоинствами данного алгоритма являются простота реализации и отсутствие сортировки.

Недостатки этого алгоритма такие:

□ нужен большой объем памяти для хранения Z-буфера. Информация о значении пиксела занимает 24 бита, информация о глубине — около 20 бит;

□ большая стоимость устранения лестничного эффекта.

Расчёт координаты *z* производится из уравнения плоскости:

Если в точке (*xi*, *yi*) → *zi* то в точке (*xi+*l, *yi*), где

если же то

Пример работы алгоритма приведён на рис. 2.1. В буфере регенерации (БР) хранятся номера цветов, в примере вместо них для упрощения понимания стоят буквы (б — белый, к — красный, с — синий, з — зелёный). Сначала в *Z*-буфере хранится максимальное 2, равное 5, а в БР — белый цвет. Потом происходит последовательное разложение в растр объектов, начиная с красного.

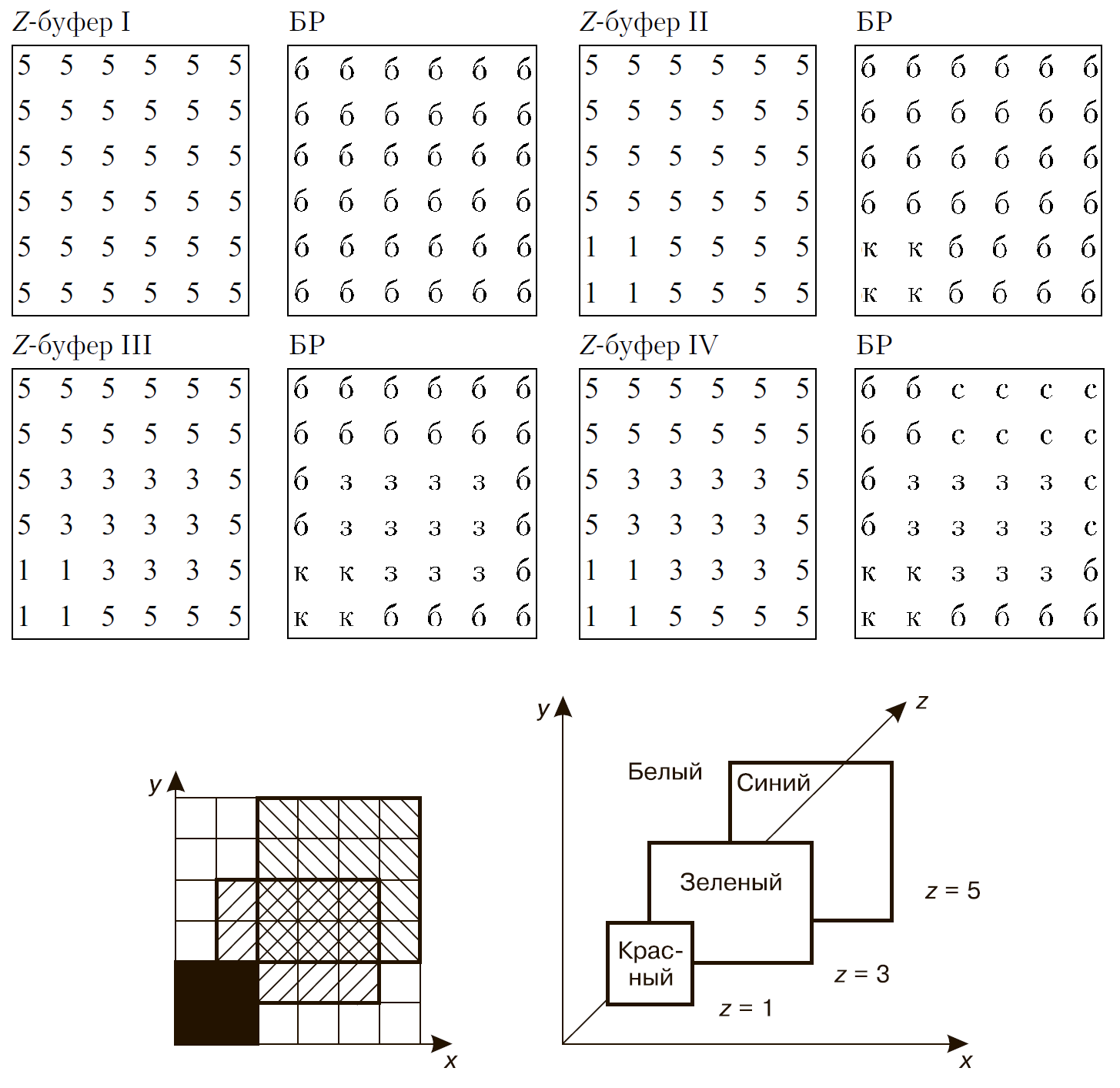


Рис. 2.1. Реализация алгоритма, использующего *Z*-буфер

## 2.3. Пример программы удаления невидимых линий методом Z‑буфера

Программа лабораторной работы №2 содержит описание классов: **Cristall** для описания главной экранной формы и **Frame** для описания кристалла**.** Файл «cristall.h» содержит описания структуры класса, файл «cristall.cpp» содержит описания функций обработки событий в окне.Файл «Frame.h» содержит описания структуры класса, файл «Frame.cpp» содержит описания функций обработки кристала

Ввод исходных данных кристаллов выполняется из текстовых файлов в структуру, описанную в классе Frame при помощи встроенной функции file.readLine().

Аффинные преобразования выполняются при помощи функций: Frame::rotateZLeft(), Frame::rotateZRight(),Frame::rotateX(),Frame::rotateY() – вращение; Frame::scaleX(double scaleValue), Frame::scaleY(double scaleValue), scaleZ(double scaleValue) – масштабирование, Frame::moveToCoord(double posX, double posY, double posZ) – перемещение; Frame::reflectX(),Frame::reflectY(),Frame::reflectZ() – отражение.

Формирование и вывод Z‑буфера выполняется функциями Frame::**fillPolygon**(int idSegment, QVector<intCoord> &points), Frame::**customLine**(int idSegment, intCoord &p1, intCoord &p2, QMap<int, QVector<test>> &boundMap и

Вывод на экран – встроенная функция repaint() и функция Frame::drawFigure() класса Frame

### 2.3.1. Cristall.h

#ifndef CRISTALL\_H

#define CRISTALL\_H

#include <QMainWindow>

#include <QFileDialog>

#include <QMessageBox>

#include <QString>

#include <Frame.h>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **Cristall**; }

QT\_END\_NAMESPACE

class **Cristall** : public QMainWindow

{

Q\_OBJECT

public:

**Cristall**(QWidget \*parent = nullptr);

~***Cristall***();

private slots:

void **slotAddPolygons**();

void **slotAddPoints**();

void **slotUploadFiles**();

void **slotExit**();

void **slotChangeAngle**();

void **on\_goStepX\_clicked**();

void **on\_goStepY\_clicked**();

void **on\_goStepZ\_clicked**();

void **on\_goScaleX\_clicked**();

void **on\_goScaleY\_clicked**();

void **on\_goScaleZ\_clicked**();

void **on\_goMove\_clicked**();

void **on\_goReflectX\_clicked**();

void **on\_goReflectY\_clicked**();

void **on\_goReflectZ\_clicked**();

private:

Ui::Cristall \*ui;

QString pathPolygons;

QString pathPoints;

Frame \*canvas;

};

#endif // CRISTALL\_H

### 2.3.2. Frame.h

#ifndef FRAME\_H

#define FRAME\_H

#include <QWidget>

#include <QPainter>

#include <QVector>

#include <QFile>

#include <QMessageBox>

#include <QtMath>

#include <QDebug>

#include <QMatrix4x4>

#include <QMatrix>

#include <QMouseEvent>

#include <QWheelEvent>

class **Frame** : public QWidget

{

Q\_OBJECT

public:

**Frame**(QWidget \* parent = 0);

bool **upload**(QString, QString);

void **drawFigure**();

void **setFiX**(double initFiX){FiX = initFiX \* M\_PI / 180;};

void **setFiY**(double initFiY){FiY = initFiY \* M\_PI / 180;};

void **setFiZ**(double initFiZ){FiZ = initFiZ \* M\_PI / 180;};

// Rotation

void **rotateX**(bool b\_repaint);

void **rotateY**(bool b\_repaint);

void **rotateZ**(QString, bool b\_repaint);

// Scale

void **scaleX**(double);

void **scaleY**(double);

void **scaleZ**(double);

// Move

void **moveToCoord**(double, double, double);

// Reflect

void **reflectX**();

void **reflectY**();

void **reflectZ**();

protected:

void ***paintEvent***(QPaintEvent \*) override;

void ***mousePressEvent***(QMouseEvent \*event) override

{

if (event->button() == Qt::LeftButton)

{

\_p = event->pos();

rotationX = \_p.x() - 250;

rotationY = \_p.y() - 250;

}

}

void ***mouseMoveEvent***(QMouseEvent \*) override;

void ***wheelEvent***(QWheelEvent \*event) override

{

if (event->delta() > 0)

{

rotateZLeft();

}else{

rotateZRight();

}

};

private:

QString pathPolygons;

QString pathPoints;

QPainter painter;

static uint const sizeCanvas = 500;

QPoint \_p;

QImage screen;

int buffFrame[sizeCanvas][sizeCanvas];

double buffZ[sizeCanvas][sizeCanvas];

double FiX = 0.07;

double FiY = 0.07;

double FiZ = 0.07;

int rotationX;

int rotationY;

struct **intCoord**{

int x, y;

double z;

};

struct **test**{

int y;

double z;

};

struct **coord**{

double x, y, z;

};

QVector<QVector<int>> dataPolygons;

QVector<coord> dataPoints;

/\* --------- Functions --------- \*/

bool **fillingDataPolygons**();

bool **fillingDataPoints**();

void **rotateZLeft**();

void **rotateZRight**();

void **calculate**(QMatrix4x4 &);

void **fillPolygon**(int, QVector<intCoord>&);

void **customLine**(int, intCoord&, intCoord&, QMap<int, QVector<test>>&);

void **addInBuffFrame**(int, int, int);

};

#endif // FRAME\_H

Cristall.cpp

#include "cristall.h"

#include "ui\_cristall.h"

Cristall::**Cristall**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::Cristall)

{

ui->setupUi(this);

connect(ui->addPolygons, SIGNAL(triggered()), this, SLOT(slotAddPolygons()));

connect(ui->addPoints, SIGNAL(triggered()), this, SLOT(slotAddPoints()));

connect(ui->uploadFiles, SIGNAL(clicked()), this, SLOT(slotUploadFiles()));

connect(ui->exit, SIGNAL(triggered()), this, SLOT(slotExit()));

connect(ui->FiX, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

connect(ui->FiY, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

connect(ui->FiZ, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

canvas = new Frame(this);

ui->canvasLayout->addWidget(canvas);

}

Cristall::~***Cristall***()

{

canvas->deleteLater();

delete ui;

}

void Cristall::**slotAddPolygons**()

{

QString tmp = QFileDialog::getOpenFileName(0, "Choose file with polygons", "", "\*.txt");

if (tmp == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл!");

return;

}

else

{

pathPolygons = tmp;

ui->uiPathPolygons->setText(pathPolygons);

ui->uploadFiles->setText("Загрузить");

}

}

void Cristall::**slotAddPoints**()

{

QString tmp = QFileDialog::getOpenFileName(0, "Choose file with points", "", "\*.txt");

if (tmp == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл!");

return;

}

else

{

pathPoints = tmp;

ui->uiPathPoints->setText(pathPoints);

ui->uploadFiles->setText("Загрузить");

}

}

void Cristall::**slotUploadFiles**()

{

if (pathPolygons == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл полигонов!");

return;

}

if (pathPoints == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл вершин!");

return;

}

bool resultUpload = canvas->upload(pathPolygons, pathPoints);

if (resultUpload)

{

ui->uploadFiles->setText("Сбросить");

}

}

void Cristall::**slotExit**()

{

this->close();

}

void Cristall::**slotChangeAngle**()

{

canvas->setFiX(ui->FiX->text().toDouble());

canvas->setFiY(ui->FiY->text().toDouble());

canvas->setFiZ(ui->FiZ->text().toDouble());

}

void Cristall::**on\_goStepX\_clicked**()

{

canvas->rotateX(true);

}

void Cristall::**on\_goStepY\_clicked**()

{

canvas->rotateY(true);

}

void Cristall::**on\_goStepZ\_clicked**()

{

canvas->rotateZ("Right", true);

}

void Cristall::**on\_goScaleX\_clicked**()

{

canvas->scaleX(ui->scaleX->text().toDouble());

}

void Cristall::**on\_goScaleY\_clicked**()

{

canvas->scaleY(ui->scaleY->text().toDouble());

}

void Cristall::**on\_goScaleZ\_clicked**()

{

canvas->scaleZ(ui->scaleZ->text().toDouble());

}

void Cristall::**on\_goMove\_clicked**()

{

canvas->moveToCoord(ui->posX->text().toDouble(), ui->posY->text().toDouble(), ui->posZ->text().toDouble());

}

void Cristall::**on\_goReflectX\_clicked**()

{

canvas->reflectX();

}

void Cristall::**on\_goReflectY\_clicked**()

{

canvas->reflectY();

}

void Cristall::**on\_goReflectZ\_clicked**()

{

canvas->reflectZ();

}

### 2.3.3. Frame.cpp

#include "Frame.h"

Frame::**Frame**(QWidget \*parent) : QWidget(parent)

{

QPen pen(Qt::black, 1, Qt::DashLine, Qt::SquareCap, Qt::RoundJoin);

painter.setPen(pen);

screen = QImage(sizeCanvas - 1, sizeCanvas - 1, QImage::Format\_ARGB32);

}

/\* ------------------ Draw Figure ------------------ \*/

void Frame::***paintEvent***(QPaintEvent \*event)

{

Q\_UNUSED(event);

painter.begin(this);

drawFigure();

painter.end();

}

void Frame::**drawFigure**()

{

screen.fill(QColor(Qt::white).rgb());

for (uint x = 0; x < sizeCanvas; x++)

{

for (uint y = 0; y < sizeCanvas; y++)

{

buffFrame[x][y] = 0;

buffZ[x][y] = -1000;

}

}

for (int i = 0; i < dataPolygons.size(); i++)

{

QVector<intCoord> points;

for (int j = 0; j < dataPolygons[i].size(); j++)

{

points.push\_back

({

int(dataPoints[dataPolygons[i][j]].x + sizeCanvas / 2 + 0.5),

int(dataPoints[dataPolygons[i][j]].y + sizeCanvas / 2 + 0.5),

dataPoints[dataPolygons[i][j]].z

});

}

fillPolygon(i + 1, *points*);

}

painter.drawImage(1, 1, screen);

}

void Frame::**fillPolygon**(int idSegment, QVector<intCoord> &points)

{

QMap<int, QVector<test>> boundMap;

for (int i = 0; i < points.size() - 1; customLine(idSegment, *points[i]*, *points[i* *+* 1*]*, *boundMap*), i++);

customLine(idSegment, *points.last()*, *points[*0*]*, *boundMap*);

foreach (int key, boundMap.keys())

{

QVector<test> value = boundMap.value(key);

for (int i = value[0].y + 1; i < value[1].y - 1; i++)

{

if (buffFrame[key][i] != idSegment)

{

double tmp = value[0].z + double(value[1].z - value[0].z) \* double(double(i - value[0].y) / double(value[1].y - value[0].y));

if (tmp >= buffZ[key][i])

{

buffFrame[key][i] = 0;

screen.setPixel(key, i, 4294967295);

buffZ[key][i] = tmp;

}

}

}

}

}

void Frame::**customLine**(int idSegment, intCoord &p1, intCoord &p2, QMap<int, QVector<test>> &boundMap)

{

const int deltaX = abs(p2.x - p1.x);

const int deltaY = abs(p2.y - p1.y);

const int signX = p1.x < p2.x ? 1 : -1;

const int signY = p1.y < p2.y ? 1 : -1;

int error = deltaX - deltaY;

int x = p1.x,

y = p1.y;

double tmp;

while(x != p2.x || y != p2.y)

{

if (p1.y == p2.y)

{

if (p1.x > p2.x)

{

tmp = p1.z + double(p2.z - p1.z) \* double(double(x - p2.x) / double(p1.x - p2.x));

}else {

tmp = p1.z + double(p2.z - p1.z) \* double(double(x - p1.x) / double(p2.x - p1.x));

}

}

else

{

if (p1.y > p2.y)

{

tmp = p1.z + double(p2.z - p1.z) \* double(double(y - p2.y) / double(p1.y - p2.y));

}else {

tmp = p1.z + double(p2.z - p1.z) \* double(double(y - p1.y) / double(p2.y - p1.y));

}

}

if (tmp >= buffZ[x][y])

{

buffFrame[x][y] = idSegment;

screen.setPixel(x, y, 4278190080);

buffZ[x][y] = tmp;

}

if (boundMap.find(x) == boundMap.end())

{

test testTmp;

testTmp.y = y;

testTmp.z = tmp;

boundMap.insert(x, {testTmp, testTmp});

}else{

if (boundMap[x][0].y > y)

{

boundMap[x][0].y = y;

boundMap[x][0].z = tmp;

}else if(boundMap[x][1].y < y)

{

boundMap[x][1].y = y;

boundMap[x][1].z = tmp;

}

}

//Logical draw

const int error2 = error \* 2;

if(error2 > -deltaY)

{

error -= deltaY;

x += signX;

}

if(error2 < deltaX)

{

error += deltaX;

y += signY;

}

}

}

/\* ------------------ Figure operations------------------ \*/

void Frame::***mouseMoveEvent***(QMouseEvent \*event)

{

if (!\_p.isNull())

{

if (abs(rotationX - (event->x() - int(sizeCanvas / 2))) > 5 && abs(rotationY - (event->y() - int(sizeCanvas / 2))) > 5)

{

if (rotationX > (event->x() - int(sizeCanvas / 2)))

{

if (rotationY > (event->y() - int(sizeCanvas / 2)))

{

FiX = abs(FiX);

rotateX(false);

if ( FiY > 0) FiY = -FiY;

rotateY(false);

}else

{

if ( FiX > 0) FiX = -FiX;

rotateX(false);

if ( FiY > 0) FiY = -FiY;

rotateY(false);

}

}else

{

if (rotationY > (event->y() - int(sizeCanvas / 2)))

{

FiX = abs(FiX);

rotateX(false);

FiY = abs(FiY);

rotateY(false);

}else

{

if ( FiX > 0) FiX = -FiX;

rotateX(false);

FiY = abs(FiY);

rotateY(false);

}

}

rotationX = event->x() - int(sizeCanvas / 2);

rotationY = event->y() - int(sizeCanvas / 2);

}

}

repaint();

}

// Rotation

void Frame::**rotateZLeft**()

{

rotateZ("Left", true);

}

void Frame::**rotateZRight**()

{

rotateZ("Right", true);

}

void Frame::**rotateX**(bool b\_repaint)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, cos(FiX), sin(FiX), 0,

0, -sin(FiX), cos(FiX), 0,

0, 0, 0, 1

);

calculate(*R*);

if (b\_repaint) repaint();

}

void Frame::**rotateY**(bool b\_repaint)

{

QMatrix4x4 R

(

cos(FiY), 0, -sin(FiY), 0,

0, 1, 0, 0,

sin(FiY), 0, cos(FiY), 0,

0, 0, 0, 1

);

calculate(*R*);

if (b\_repaint) repaint();

}

void Frame::**rotateZ**(QString diraction, bool b\_repaint)

{

if (diraction == "Left")

{

FiZ = abs(FiZ);

}else if(diraction == "Right")

{

if ( FiZ > 0) FiZ = -FiZ;

}

QMatrix4x4 R

(

cos(FiZ), sin(FiZ), 0, 0,

-sin(FiZ), cos(FiZ), 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

if (b\_repaint) repaint();

}

// Scale

void Frame::**scaleX**(double scaleValue)

{

QMatrix4x4 R

(

scaleValue, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**scaleY**(double scaleValue)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, scaleValue, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**scaleZ**(double scaleValue)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, scaleValue, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

// Move

void Frame::**moveToCoord**(double posX, double posY, double posZ)

{

QMatrix4x4 R

(

1, 0, 0, posX,

0, 1, 0, posY,

0, 0, 1, posZ,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

// Reflect

void Frame::**reflectX**()

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, -1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**reflectY**()

{

QMatrix4x4 R

(

-1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**reflectZ**()

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, -1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**calculate**(QMatrix4x4 &R)

{

for (int i = 0; i < dataPoints.size(); i++)

{

QVector4D vectCoord(dataPoints[i].x, dataPoints[i].y, dataPoints[i].z, 1);

QVector4D result = R \* vectCoord;

dataPoints[i].x = result[0];

dataPoints[i].y = result[1];

dataPoints[i].z = result[2];

}

}

/\* ------------------ Upload data for figure ------------------ \*/

bool Frame::**upload**(QString initPathPolygons, QString initPathPoints)

{

pathPolygons = initPathPolygons;

pathPoints = initPathPoints;

bool resultFillPolygons = fillingDataPolygons();

bool resultFillPoints = fillingDataPoints();

repaint();

drawFigure();

if (resultFillPolygons && resultFillPoints) return true;

}

bool Frame::**fillingDataPolygons**()

{

dataPolygons.clear();

QFile file(pathPolygons);

if(!file.*open*(QIODevice::ReadOnly))

{

QMessageBox::warning(this, "Предупреждение!", "Файл " + pathPolygons + " не найден!");

return false;

}

while (!file.*atEnd*())

{

QStringList list;

QString tmpStr = file.readLine();

list = tmpStr.split(QRegExp(" "));

QVector<int> tmpVec;

for (int i = 1; i <= list.at(0).toInt(); i++)

{

tmpVec.push\_back(list.at(i).toInt());

}

dataPolygons.push\_back(tmpVec);

}

file.*close*();

return true;

}

bool Frame::**fillingDataPoints**()

{

dataPoints.clear();

QFile file(pathPoints);

if(!file.*open*(QIODevice::ReadOnly))

{

QMessageBox::warning(this, "Предупреждение!", "Файл " + pathPoints + " не найден!");

return false;

}

while (!file.*atEnd*())

{

QStringList list;

QString tmp = file.readLine();

list = tmp.split(QRegExp(" "));

dataPoints.push\_back({100 \* list[0].toDouble(), 100 \* list[1].toDouble(), 100 \* list[2].toDouble()});

}

\

file.*close*();

return true;

}

### 2.3.4. Результаты работы программы

Результаты работы программы приведены на рис. 2.1 и 2.2.

|  |  |
| --- | --- |
|  |  |
| *Рис. 2.1. Кристалл 1* | *Рис. 2.2. Кристалл 2* |

## 2.4. Контрольные вопросы

1. Какие алгоритмы удаления скрытых линий используют при работе в пространстве изображений?
2. Какая система координат использована в программной реализации алгоритма удаления скрытых линий?
3. Какую особенность имеет алгоритм метода Z-буфера для удаления скрытых линий в случае применения его для выпуклых многогранников?
4. Какие достоинства и недостатки имеет метод Z-буфера?
5. Для решения каких задач геометрического моделирования используется метод Z-буфера?

# Работа 3. Удаления невидимых линий по алгоритму Вейлера‑Азертона

**Цель работы:** написать программу, обеспечивающую удаление скрытых рёбер многогранника по алгоритму Вейлера‑Азертона, работающего в пространстве объекта.

## 3.1. Порядок выполнения работы

1. Ознакомиться со способами удаления скрытых линий, работающих в пространстве объекта.
2. Написать программу удаления скрытых рёбер многогранника.
3. Выполнить удаление скрытых рёбер многогранников.
4. Составить отчёт, который должен содержать:

— задание и рисунки многогранников,

— описание алгоритма удаления скрытых линий по алгоритму Вейлера‑Азертона для многогранников,

— результаты работы программы удаления скрытых линий,

— выводы о проделанной работе.

— текст программы удаления скрытых линий,

1. Ответить устно на контрольные вопросы.

## 3.2. Методические указания

**Алгоритм удаления скрытых линий и поверхностей Вейлера‑Азертона**. Выходными данными этого алгоритма, который для достижения необходимой точности работает в пространстве объекта, служат многоугольники. Поскольку выходом являются многоугольники, то алгоритм можно легко использовать для удаления как невидимых линий, так и невидимых поверхностей. Алгоритм удаления невидимых поверхностей состоит из четырёх шагов.

1. Предварительная сортировка по глубине.
2. Отсечение по границе ближайшего к наблюдателю многоугольника, называемое сортировкой многоугольников на плоскости.
3. Удаление многоугольников, экранированных многоугольником, ближайшим к точке наблюдения.
4. Если требуется, то рекурсивное подразбиение и окончательная сортировка для устранения всех неопределенностей.

Предварительная сортировка по глубине нужна для формирования списка приблизительных приоритетов. Предположим, что точка наблюдения расположена в бесконечности на положительной полуоси z, тогда ближайшим к ней и первым в списке будет тот многоугольник, который обладает вершиной с максимальной координатой z.

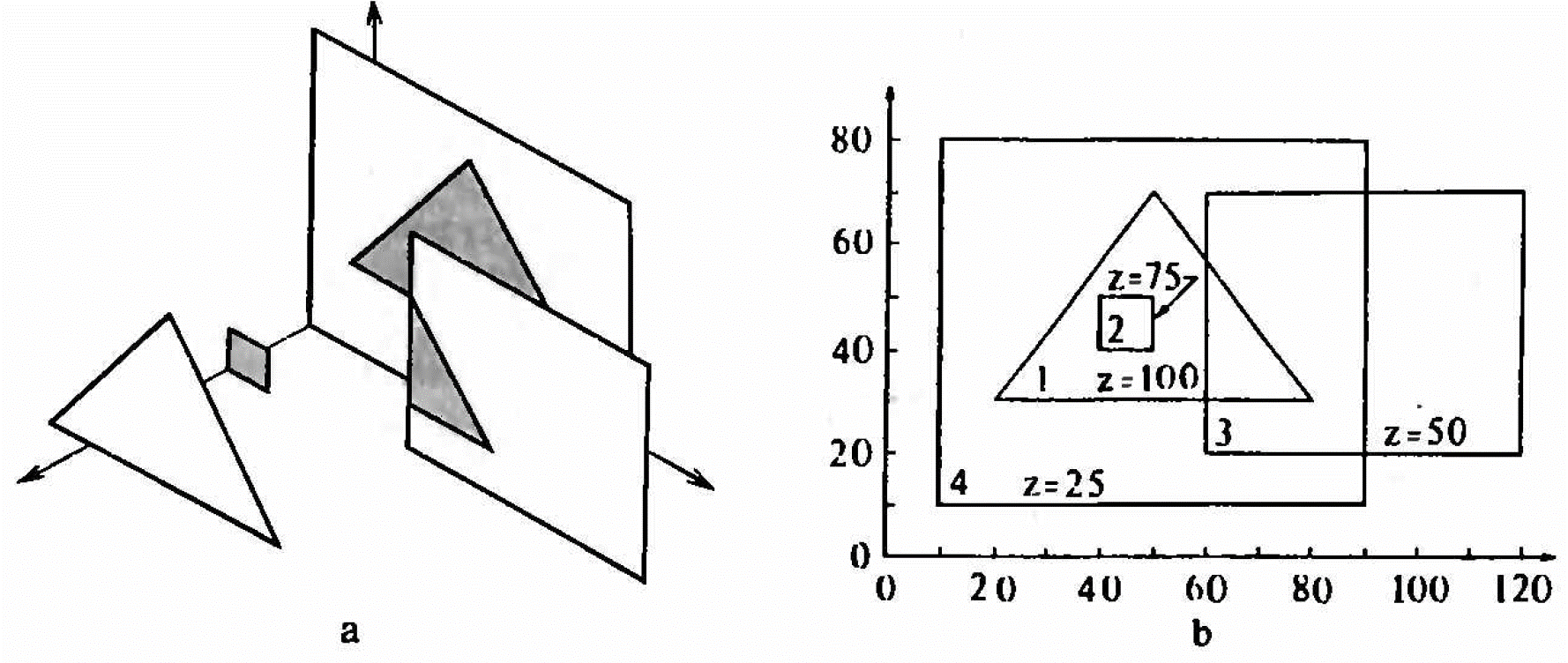


Рис. 3.1. Отсечение многоугольников по приоритетам для алгоритма удаления невидимых поверхностей Вейлера — Азертона.

В качестве отсекающего многоугольника используется копия первого многоугольника из предварительного списка приоритетов по глубине. Отсекаться будут остающиеся в этом списке многоугольники, включая и первый многоугольник. Вводятся два списка: внутренний и внешний. С помощью алгоритма отсечения Вейлера — Азертона все многоугольники отсекаются по границам отсекающего многоугольника. Фактически это двумерная операция отсечения проекций отсекающего и отсекаемого многоугольников. Та часть каждого отсекаемого многоугольника, которая оказывается внутри отсекающего, если она имеется, попадает во внутренний список. Оставшаяся часть, если таковая есть, попадает во внешний список. Этот этап алгоритма является сортировкой на плоскости или *ху*-сортировкой. Пример приведён на рис.3.1. На рис. 3.2. показаны внутренний и внешний списки для сцены на рис. 3.1. Теперь сравниваются глубины каждого многоугольника из внутреннего списка с глубиной отсекающего многоугольника. С использованием координат (*х*, *у*) вершин отсекаемых многоугольников и уравнений несущих плоскостей вычисляются глубины (координаты *z*) каждой вершины. Затем они сравниваются с минимальной координатой *z*(*z*c min) для отсекающего многоугольника. Если глубина ни одной из этих вершин не будет больше *z*c min, то все многоугольники из внутреннего списка экранируются отсекающим многоугольником (рис. 3.1), Эти многоугольники удаляются, и изображается внутренний список. Заметим, что во внутреннем списке остался лишь отсекающий многоугольник. Работа алгоритма затем продолжается с внешним списком.

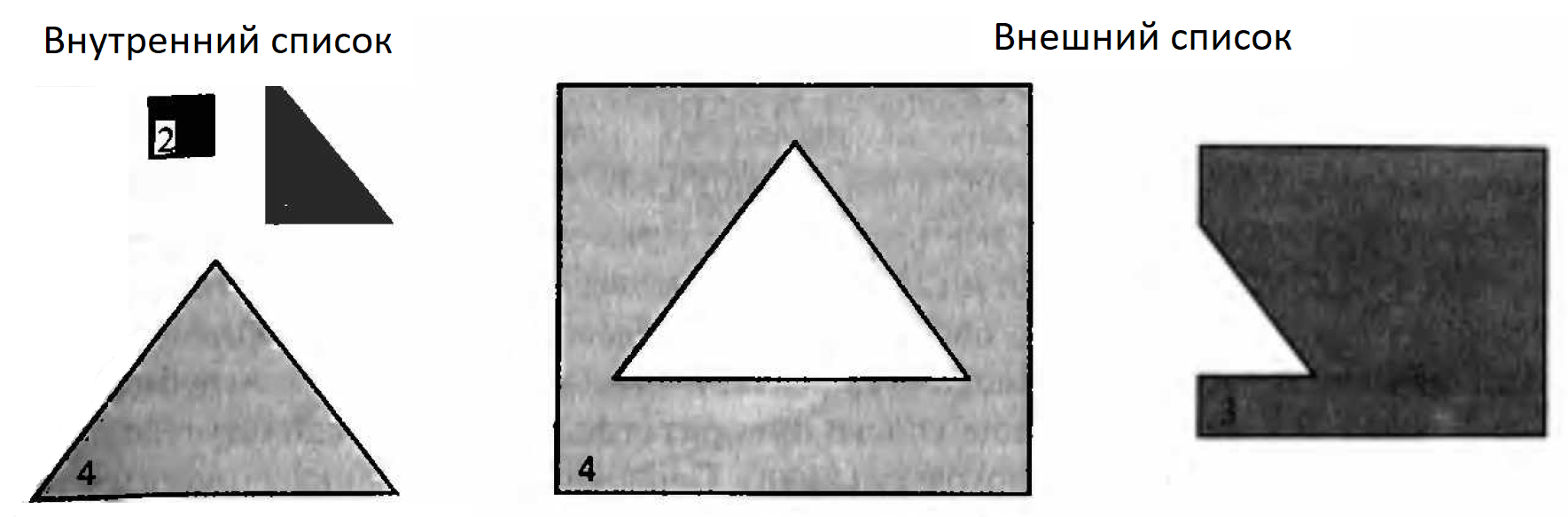


Рис. 3.2 Внутренний и внешний списки многоугольников.

Если координата z какого-либо многоугольника из внутреннего списка окажется больше, чем *z*c min, то такой многоугольник по крайней мере частично экранирует отсекающий многоугольник. На рис. 3.3 показано, как это может произойти. В подобном случае результат предварительной сортировки по глубине ошибочен. Поэтому алгоритм рекурсивно подразделяет плоскость (*х*, *у*), используя многоугольник, нарушивший порядок, в качестве нового отсекающего многоугольника. Отсечению подлежат многоугольники из внутреннего списка, причём старый отсекающий многоугольник теперь сам будет подвергнут отсечению новым отсекающим многоугольником. Подчеркнём, что новый отсекающий многоугольник является копией исходного многоугольника, а не его остатка после первого отсечения. Использование копии неотсечённого многоугольника позволяет минимизировать число разбиений.

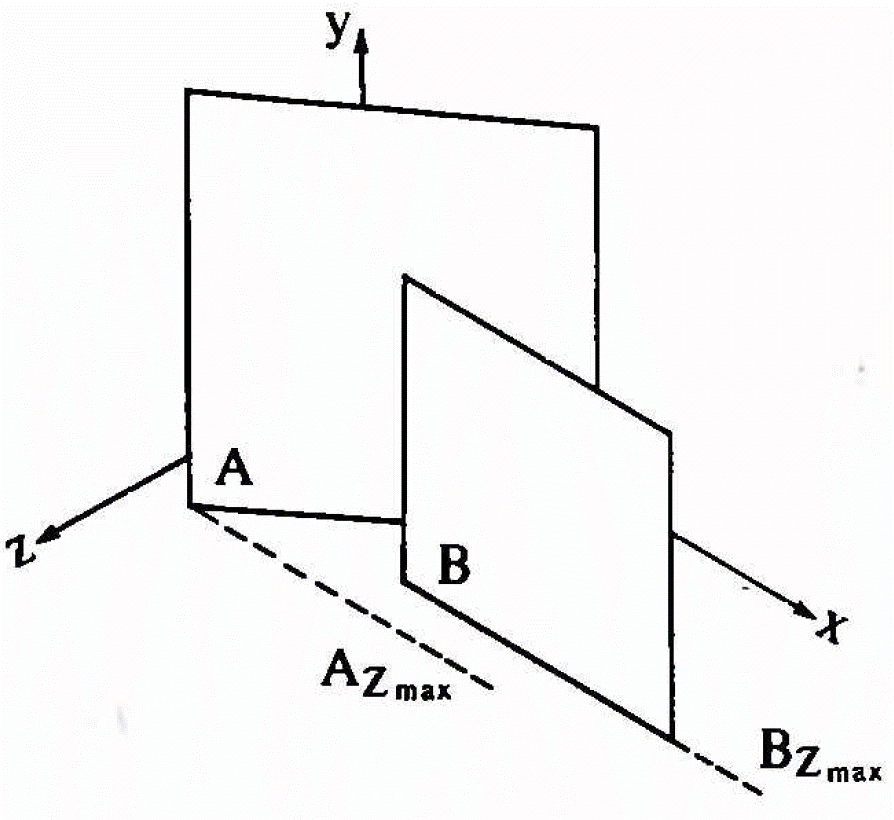


Рис. 3.3. Условие возникновения ошибочного результата в предварительной сортировке по *z*.

Более полной иллюстрацией этого алгоритма послужит следующий простой пример.

Возьмём два прямоугольника (рис. 3.3). Вершинами прямоугольника *А* являются (5, 0, 25), (40, 0, 5), (40, 40, 5) и (5, 40, 25). Прямоугольник *В* имеет вершины (25, 0, 20), (55, 0, 20), (55, 30, 20) и (25, 30, 20). На рис. 3.4, а показана эта сцена до отсечения из точки наблюдения, расположенной в бесконечности на положительной полуоси *z*. Хотя *В* экранирует часть *А*, предварительная сортировка по глубине поставит *А* перед *В* в упорядоченном списке приоритетов. Копия *А* используется в качестве первого отсекающего многоугольника. Первоначальный список отсекаемых многоугольников содержит и *А*, и *В*, как показано в табл. 3.1. В табл. 3.1 и на рис. 3.4, b показан результат отсечения содержимого этого списка относительно многоугольника *А*. Теперь внутренний список содержит многоугольники *А* и *С*, а внешний список содержит *В*'. Сравнение глубин *А* и *С* с глубиной отсекающего многоугольника показывает, что *С* находится перед отсекающим многоугольником. Далее алгоритм рекурсивно разбивает плоскость, используя в качестве отсекающего многоугольник *В*, частью которого является *С*, а в качестве списка отсекаемых многоугольников — внутренний список. Результат показан на рис. 3.4, с и табл. 3.1. Участок, обозначенный *А*", отсекается и помешается во внешний список. Участок, обозначенный *D*, помещается во внутренний список. Сравнение многоугольников *С* и *D* из внутреннего списка с отсекающим многоугольником *В* показывает, что *D* экранирован. Следовательно, он удаляется. Поскольку *С* совпадает с отсекающим многоугольником *В*, он остаётся во внутреннем списке. Рекурсия не нужна. Многоугольник *С* изображается. Алгоритм продолжает работу с многоугольниками *В*' и *А*', извлекаемыми из внешнего списка. Подробности приведены в табл. 3.1. Окончательный результат показан на рис. 3.4, d.

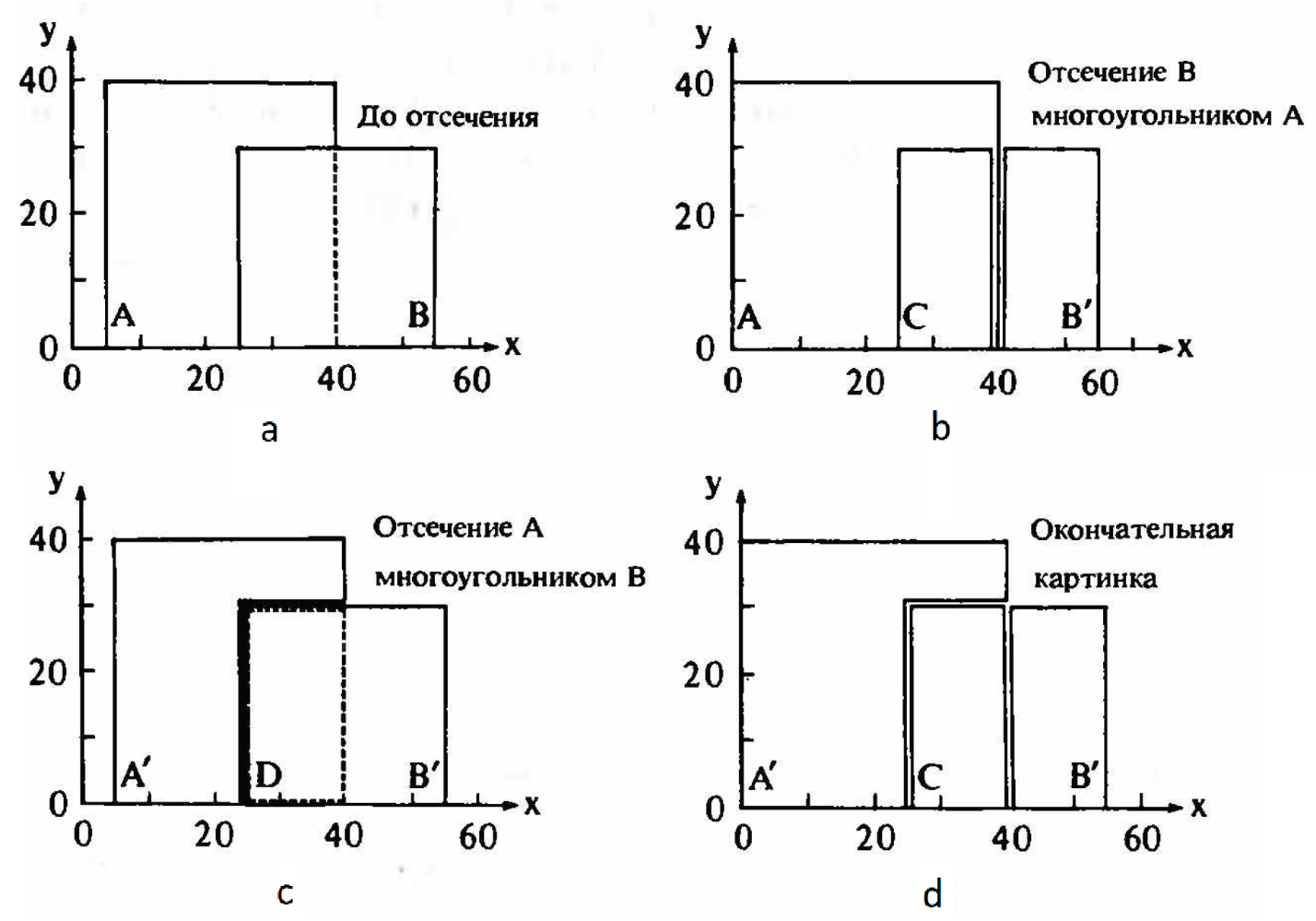
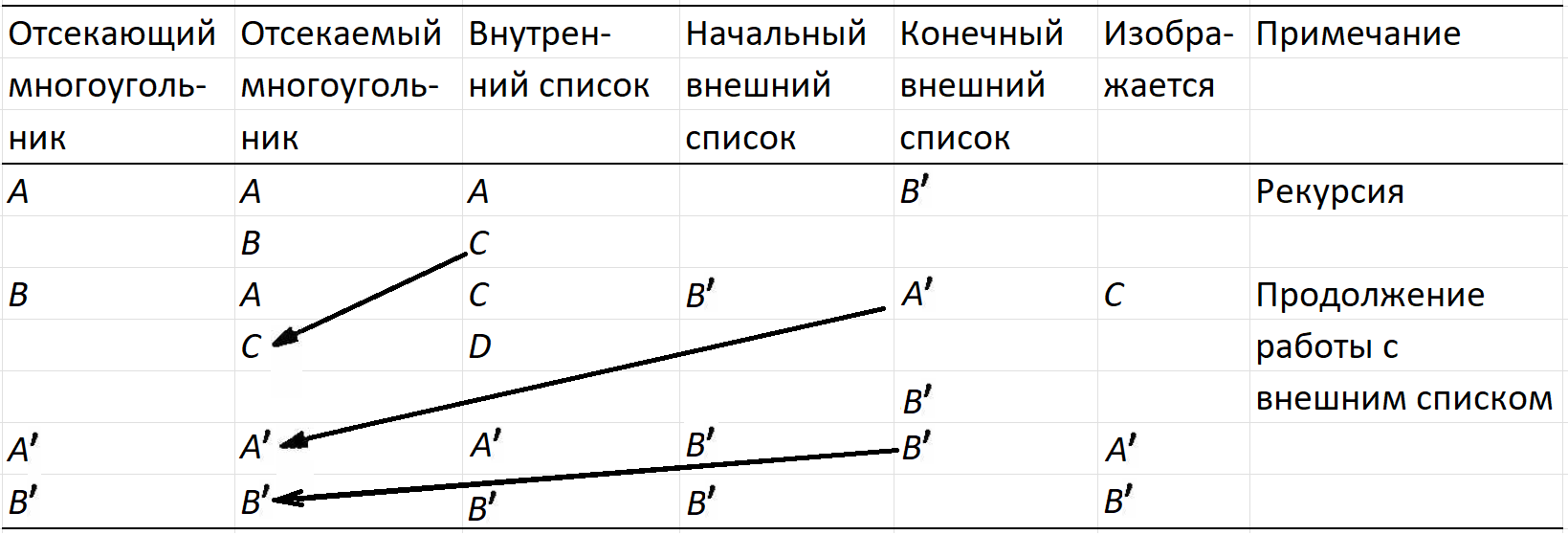


Рис. 3.4. Рекурсивное разбиение для алгоритма Вейлера — Азертона.

Таблица 3.1



Заслуживает внимание ещё одна дополнительная деталь этого алгоритма. Когда некоторый многоугольник циклически перекрывается с отсекающим, т. е. когда он лежит как впереди, так и позади отсекающего (рис. 3.5, а), то в рекурсивном разбиении необходимости нет. Дело в том, что все экранируемое циклическим многоугольником уже было удалено на предыдущем шаге отсечения. Необходимо лишь произвести отсечение исходного многоугольника по границам циклического многоугольника и изобразить результат. Ненужное рекурсивное разбиение можно предотвратить, если создать список многоугольников, которые уже использовались как отсекающие. Тогда, если при рекурсивном разбиении текущий отсекающий многоугольник появляется в этом списке, значит, обнаружен циклически перекрывающийся многоугольник. Следовательно, не требуется никакого дополнительного разбиения. Заметим, что данный алгоритм непосредственно обрабатывает случаи циклического перекрытия сразу нескольких многоугольников, как показано на рис. 3.5, b.

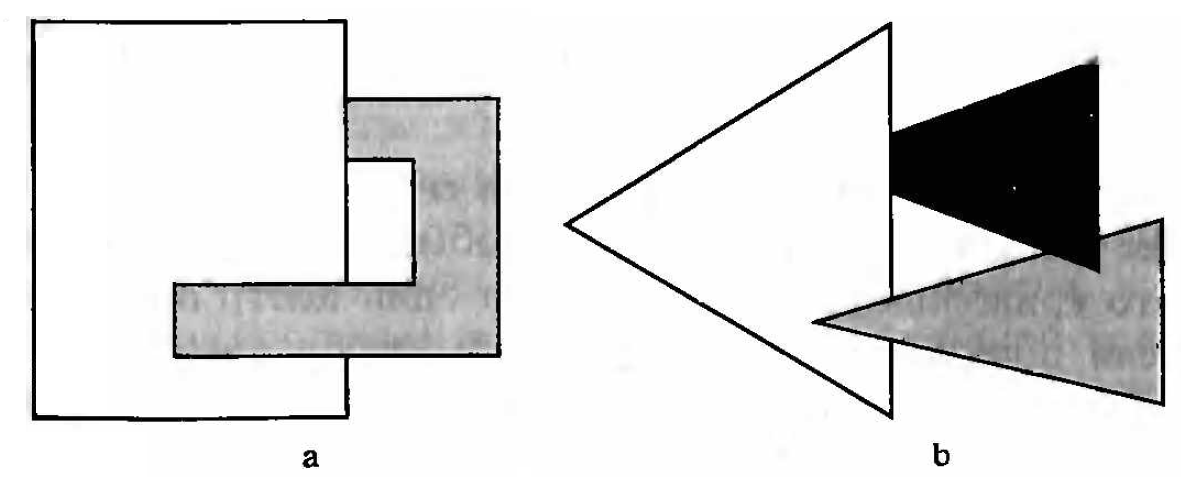


Рис. 4.48. Циклически перекрывающиеся многоугольники.

## 3.3. Пример программы удаления невидимых линий, реализующей алгоритм Вейлера — Азертона

Программа лабораторной работы №3 содержит описание классов: **Cristall** для описания главной экранной формы и **Frame** для описания кристалла**.** Файл «cristall.h» содержит описания структуры класса, файл «cristall.cpp» содержит описания функций обработки событий в окне.Файл «Frame.h» содержит описания структуры класса, файл «Frame.cpp» содержит описания функций обработки кристала

Ввод исходных данных кристаллов выполняется из текстовых файлов в структуру, описанную в классе Frame при помощи встроенной функции file.readLine().

Аффинные преобразования выполняются при помощи функций: Frame::rotateZLeft(), Frame::rotateZRight(),Frame::rotateX(),Frame::rotateY() – вращение; Frame::scaleX(double scaleValue), Frame::scaleY(double scaleValue), scaleZ(double scaleValue) – масштабирование, Frame::moveToCoord(double posX, double posY, double posZ) – перемещение; Frame::reflectX(),Frame::reflectY(),Frame::reflectZ() – отражение.

Формирование и вывод буфера выполняется функциями Frame::**drawFigureZBuffer**(),Frame::**drawFigureVeyler**(),Frame::**fillPolygon**(int idSegment, QVector<intCoord> &points), Frame::**customLine**(int idSegment, intCoord &p1, intCoord &p2, QMap<int, QVector<intCoord>> &boundMap)

Вывод на экран – встроенная функция repaint() и функция Frame::drawFigure() класса Frame

### 3.3.1. Cristall.h

#ifndef CRISTALL\_H

#define CRISTALL\_H

#include <QMainWindow>

#include <QFileDialog>

#include <QMessageBox>

#include <QString>

#include <Frame.h>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **Cristall**; }

QT\_END\_NAMESPACE

class **Cristall** : public QMainWindow

{

Q\_OBJECT

public:

**Cristall**(QWidget \*parent = nullptr);

~***Cristall***();

private slots:

void **slotAddPolygons**();

void **slotAddPoints**();

void **slotUploadFiles**();

void **slotExit**();

void **slotChangeAngle**();

void **on\_goStepX\_clicked**();

void **on\_goStepY\_clicked**();

void **on\_goStepZ\_clicked**();

void **on\_goScaleX\_clicked**();

void **on\_goScaleY\_clicked**();

void **on\_goScaleZ\_clicked**();

void **on\_goMove\_clicked**();

void **on\_goReflectX\_clicked**();

void **on\_goReflectY\_clicked**();

void **on\_goReflectZ\_clicked**();

void **slotZBuffer**(int);

void **slotVeyler**(int);

void **slotGuro**(int);

private:

Ui::Cristall \*ui;

QString pathPolygons;

QString pathPoints;

Frame \*canvas;

};

#endif // CRISTALL\_H

### 3.3.2. Frame.h

#ifndef FRAME\_H

#define FRAME\_H

#include <QWidget>

#include <QPainter>

#include <QVector>

#include <QFile>

#include <QMessageBox>

#include <QtMath>

#include <QDebug>

#include <QMatrix4x4>

#include <QMatrix>

#include <QMouseEvent>

#include <QWheelEvent>

class **Frame** : public QWidget

{

Q\_OBJECT

public:

**Frame**(QWidget \* parent = 0);

bool **upload**(QString, QString);

void **setFiX**(double initFiX){FiX = initFiX \* M\_PI / 180;};

void **setFiY**(double initFiY){FiY = initFiY \* M\_PI / 180;};

void **setFiZ**(double initFiZ){FiZ = initFiZ \* M\_PI / 180;};

// Rotation

void **rotateX**(bool b\_repaint);

void **rotateY**(bool b\_repaint);

void **rotateZ**(QString, bool b\_repaint);

// Scale

void **scaleX**(double);

void **scaleY**(double);

void **scaleZ**(double);

// Move

void **moveToCoord**(double, double, double);

// Reflect

void **reflectX**();

void **reflectY**();

void **reflectZ**();

// Set option draw

void **setOptionDraw**(int value){ optionDraw = value; };

void **setOptionFill**(bool value){ optionFill = value; };

protected:

void ***paintEvent***(QPaintEvent \*) override;

void ***mousePressEvent***(QMouseEvent \*event) override

{

if (event->button() == Qt::LeftButton)

{

\_p = event->pos();

rotationX = \_p.x() - 250;

rotationY = \_p.y() - 250;

}

}

void ***mouseMoveEvent***(QMouseEvent \*) override;

void ***wheelEvent***(QWheelEvent \*event) override

{

if (event->delta() > 0)

{

rotateZLeft();

}

else

{

rotateZRight();

}

};

private:

QString pathPolygons;

QString pathPoints;

QPainter painter;

int optionDraw = 0;

int optionFill = false;

static uint const sizeCanvas = 500;

QPoint \_p;

QImage screen;

int buffFrame[sizeCanvas][sizeCanvas];

double buffZ[sizeCanvas][sizeCanvas];

double FiX = 0.07;

double FiY = 0.07;

double FiZ = 0.07;

int rotationX;

int rotationY;

struct **intCoord**{

int x, y;

double z;

};

struct **coord**{

double x, y, z;

};

coord lightCoord;

QVector<QVector<int>> dataPolygons;

QVector<coord> dataPoints;

/\* --------- Functions --------- \*/

void **defaultDrawFigure**();

void **drawFigureZBuffer**();

void **drawFigureVeyler**();

bool **fillingDataPolygons**();

bool **fillingDataPoints**();

void **rotateZLeft**();

void **rotateZRight**();

void **calculate**(QMatrix4x4 &);

void **fillPolygon**(int, QVector<intCoord>&);

void **customLine**(int, intCoord&, intCoord&, QMap<int, QVector<intCoord>>&);

void **addInBuffFrame**(int, int, int);

};

#endif // FRAME\_H

Cristall.cpp

#include "cristall.h"

#include "ui\_cristall.h"

Cristall::**Cristall**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::Cristall)

{

ui->setupUi(this);

connect(ui->addPolygons, SIGNAL(triggered()), this, SLOT(slotAddPolygons()));

connect(ui->addPoints, SIGNAL(triggered()), this, SLOT(slotAddPoints()));

connect(ui->uploadFiles, SIGNAL(clicked()), this, SLOT(slotUploadFiles()));

connect(ui->exit, SIGNAL(triggered()), this, SLOT(slotExit()));

connect(ui->FiX, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

connect(ui->FiY, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

connect(ui->FiZ, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

connect(ui->checkBox\_1, SIGNAL(stateChanged(int)), this, SLOT(slotZBuffer(int)));

connect(ui->checkBox\_2, SIGNAL(stateChanged(int)), this, SLOT(slotVeyler(int)));

connect(ui->checkBox\_3, SIGNAL(stateChanged(int)), this, SLOT(slotGuro(int)));

canvas = new Frame(this);

ui->canvasLayout->addWidget(canvas);

}

Cristall::~***Cristall***()

{

canvas->deleteLater();

delete ui;

}

void Cristall::**slotAddPolygons**()

{

QString tmp = QFileDialog::getOpenFileName(0, "Choose file with polygons", "", "\*.txt");

if (tmp == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл!");

return;

}

else

{

pathPolygons = tmp;

ui->uiPathPolygons->setText(pathPolygons);

ui->uploadFiles->setText("Загрузить");

}

}

void Cristall::**slotAddPoints**()

{

QString tmp = QFileDialog::getOpenFileName(0, "Choose file with points", "", "\*.txt");

if (tmp == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл!");

return;

}

else

{

pathPoints = tmp;

ui->uiPathPoints->setText(pathPoints);

ui->uploadFiles->setText("Загрузить");

}

}

void Cristall::**slotUploadFiles**()

{

if (pathPolygons == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл полигонов!");

return;

}

if (pathPoints == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл вершин!");

return;

}

bool resultUpload = canvas->upload(pathPolygons, pathPoints);

if (resultUpload)

{

ui->uploadFiles->setText("Сбросить");

}

}

void Cristall::**slotExit**()

{

this->close();

}

void Cristall::**slotChangeAngle**()

{

canvas->setFiX(ui->FiX->text().toDouble());

canvas->setFiY(ui->FiY->text().toDouble());

canvas->setFiZ(ui->FiZ->text().toDouble());

}

void Cristall::**on\_goStepX\_clicked**()

{

canvas->rotateX(true);

}

void Cristall::**on\_goStepY\_clicked**()

{

canvas->rotateY(true);

}

void Cristall::**on\_goStepZ\_clicked**()

{

canvas->rotateZ("Right", true);

}

void Cristall::**on\_goScaleX\_clicked**()

{

canvas->scaleX(ui->scaleX->text().toDouble());

}

void Cristall::**on\_goScaleY\_clicked**()

{

canvas->scaleY(ui->scaleY->text().toDouble());

}

void Cristall::**on\_goScaleZ\_clicked**()

{

canvas->scaleZ(ui->scaleZ->text().toDouble());

}

void Cristall::**on\_goMove\_clicked**()

{

canvas->moveToCoord(ui->posX->text().toDouble(), ui->posY->text().toDouble(), ui->posZ->text().toDouble());

}

void Cristall::**on\_goReflectX\_clicked**()

{

canvas->reflectX();

}

void Cristall::**on\_goReflectY\_clicked**()

{

canvas->reflectY();

}

void Cristall::**on\_goReflectZ\_clicked**()

{

canvas->reflectZ();

}

void Cristall::**slotZBuffer**(int check)

{

if (check == 2)

{

canvas->setOptionDraw(1);

ui->checkBox\_1->setCheckState(Qt::Checked);

ui->checkBox\_2->setCheckState(Qt::Unchecked);

}else{

if (!ui->checkBox\_2->isChecked())

{

canvas->setOptionDraw(0);

}

}

canvas->update();

}

void Cristall::**slotVeyler**(int check)

{

if (check == 2)

{

canvas->setOptionDraw(2);

ui->checkBox\_1->setCheckState(Qt::Unchecked);

ui->checkBox\_2->setCheckState(Qt::Checked);

}else{

if (!ui->checkBox\_1->isChecked())

{

canvas->setOptionDraw(0);

}

}

canvas->update();

}

void Cristall::**slotGuro**(int check)

{

if (check == 2)

{

canvas->setOptionFill(true);

}else{

canvas->setOptionFill(false);

}

canvas->update();

}

### 3.3.3. Frame.cpp

#include "Frame.h"

Frame::**Frame**(QWidget \*parent) : QWidget(parent)

{

QPen pen(Qt::black, 1, Qt::DashLine, Qt::SquareCap, Qt::RoundJoin);

painter.setPen(pen);

screen = QImage(sizeCanvas - 1, sizeCanvas - 1, QImage::Format\_ARGB32);

lightCoord.x = 200;

lightCoord.y = 300;

lightCoord.x = 50;

}

/\* ------------------ Draw Figure ------------------ \*/

void Frame::***paintEvent***(QPaintEvent \*event)

{

Q\_UNUSED(event);

painter.begin(this);

switch(optionDraw)

{

case 0:

defaultDrawFigure();

break;

case 1:

drawFigureZBuffer();

break;

case 2:

drawFigureVeyler();

break;

}

painter.end();

}

void Frame::**defaultDrawFigure**()

{

for (int i = 0; i < dataPolygons.size(); i++)

{

QPointF points[dataPolygons[i].size()];

for (int j = 0; j < dataPolygons[i].size(); j++)

{

points[j] = QPointF(dataPoints[dataPolygons[i][j]].x + 250, dataPoints[dataPolygons[i][j]].y + 250);

}

painter.drawPolygon(points, dataPolygons[i].size());

}

}

void Frame::**drawFigureZBuffer**()

{

screen.fill(QColor(Qt::white).rgb());

for (uint x = 0; x < sizeCanvas; x++)

{

for (uint y = 0; y < sizeCanvas; y++)

{

buffFrame[x][y] = 0;

buffZ[x][y] = -1000;

}

}

for (int i = 0; i < dataPolygons.size(); i++)

{

QVector<intCoord> points;

for (int j = 0; j < dataPolygons[i].size(); j++)

{

points.push\_back

({

int(dataPoints[dataPolygons[i][j]].x + 250 + 0.5),

int(dataPoints[dataPolygons[i][j]].y + 250 + 0.5),

dataPoints[dataPolygons[i][j]].z

});

}

fillPolygon(i + 1, *points*);

}

painter.drawImage(1, 1, screen);

}

void Frame::**drawFigureVeyler**()

{

drawFigureZBuffer();

}

void Frame::**fillPolygon**(int idSegment, QVector<intCoord> &points)

{

QMap<int, QVector<intCoord>> boundMap;

for (int i = 0; i < points.size() - 1; customLine(idSegment, *points[i]*, *points[i* *+* 1*]*, *boundMap*), i++);

customLine(idSegment, *points.last()*, *points[*0*]*, *boundMap*);

int distance = sqrt(pow((lightCoord.x - 250), 2) + pow((lightCoord.y - 250), 2) + pow(lightCoord.z, 2));

foreach (int key, boundMap.keys())

{

QVector<intCoord> value = boundMap.value(key);

for (int i = value[0].y; i < value[1].y; i++)

{

if (buffFrame[key][i] != idSegment)

{

double tmp = value[0].z + double(value[1].z - value[0].z) \* double(double(i - value[0].y) / double(value[1].y - value[0].y));

if (tmp >= buffZ[key][i])

{

buffFrame[key][i] = 0;

int tmpDistance = sqrt(pow((lightCoord.x - key), 2) + pow((lightCoord.y - i), 2) + pow((lightCoord.z - tmp), 2));

double betweenDistance = distance - tmpDistance \* 0.8;

int alpha = 255.0 \* (1.0 - betweenDistance / 100.0) < 255 ? 255.0 \* (1.0 - betweenDistance / 100.0) : 255;

alpha = alpha < 0 ? 0 : alpha;

switch(optionDraw)

{

case 1:

optionFill ?

screen.setPixelColor(key, i, QColor(0, 20, 255, alpha)):

screen.setPixelColor(key, i, 4294967295);

break;

case 2:

optionFill ?

screen.setPixelColor(key, i, QColor(0, 20, 255, alpha)):

screen.setPixelColor(key, i, 4294967295);

break;

}

buffZ[key][i] = tmp;

}

}

}

}

}

void Frame::**customLine**(int idSegment, intCoord &p1, intCoord &p2, QMap<int, QVector<intCoord>> &boundMap)

{

const int deltaX = abs(p2.x - p1.x);

const int deltaY = abs(p2.y - p1.y);

const int signX = p1.x < p2.x ? 1 : -1;

const int signY = p1.y < p2.y ? 1 : -1;

int error = deltaX - deltaY;

int x = p1.x,

y = p1.y;

double tmp;

while(x != p2.x || y != p2.y)

{

if (p1.x == p2.x)

{

tmp = p1.z + double(p2.z - p1.z) \* double(double(y - p1.y) / double(p2.y - p1.y));

}else

{

tmp = p1.z + double(p2.z - p1.z) \* double(double(x - p1.x) / double(p2.x - p1.x));

}

if (tmp >= buffZ[x][y])

{

buffFrame[x][y] = idSegment;

screen.setPixelColor(x, y, 4278190080); // Black

buffZ[x][y] = tmp;

}

if (boundMap.find(x) == boundMap.end())

{

intCoord boundCoord;

boundCoord.y = y;

boundCoord.z = tmp;

boundMap.insert(x, {boundCoord, boundCoord});

}else{

if (boundMap[x][0].y > y)

{

boundMap[x][0].y = y;

boundMap[x][0].z = tmp;

}else if(boundMap[x][1].y < y)

{

boundMap[x][1].y = y;

boundMap[x][1].z = tmp;

}

}

//Logical draw

const int error2 = error \* 2;

if(error2 > -deltaY)

{

error -= deltaY;

x += signX;

}

if(error2 < deltaX)

{

error += deltaX;

y += signY;

}

}

}

/\* ------------------ Figure operations------------------ \*/

void Frame::***mouseMoveEvent***(QMouseEvent \*event)

{

if (!\_p.isNull())

{

if (abs(rotationX - (event->x() - int(sizeCanvas / 2))) > 5 && abs(rotationY - (event->y() - int(sizeCanvas / 2))) > 5)

{

if (rotationX > (event->x() - int(sizeCanvas / 2)))

{

if (rotationY > (event->y() - int(sizeCanvas / 2)))

{

if ( FiX > 0) FiX = -FiX;

rotateX(false);

FiY = abs(FiY);

rotateY(false);

}else

{

FiX = abs(FiX);

rotateX(false);

FiY = abs(FiY);

rotateY(false);

}

}else

{

if (rotationY > (event->y() - int(sizeCanvas / 2)))

{

if ( FiX > 0) FiX = -FiX;

rotateX(false);

if ( FiY > 0) FiY = -FiY;

rotateY(false);

}else

{

FiX = abs(FiX);

rotateX(false);

if ( FiY > 0) FiY = -FiY;

rotateY(false);

}

}

rotationX = event->x() - int(sizeCanvas / 2);

rotationY = event->y() - int(sizeCanvas / 2);

}

}

repaint();

}

// Rotation

void Frame::**rotateZLeft**()

{

rotateZ("Left", true);

}

void Frame::**rotateZRight**()

{

rotateZ("Right", true);

}

void Frame::**rotateX**(bool b\_repaint)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, cos(FiX), sin(FiX), 0,

0, -sin(FiX), cos(FiX), 0,

0, 0, 0, 1

);

calculate(*R*);

if (b\_repaint) repaint();

}

void Frame::**rotateY**(bool b\_repaint)

{

QMatrix4x4 R

(

cos(FiY), 0, -sin(FiY), 0,

0, 1, 0, 0,

sin(FiY), 0, cos(FiY), 0,

0, 0, 0, 1

);

calculate(*R*);

if (b\_repaint) repaint();

}

void Frame::**rotateZ**(QString diraction, bool b\_repaint)

{

if (diraction == "Left")

{

FiZ = abs(FiZ);

}else if(diraction == "Right")

{

if ( FiZ > 0) FiZ = -FiZ;

}

QMatrix4x4 R

(

cos(FiZ), sin(FiZ), 0, 0,

-sin(FiZ), cos(FiZ), 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

if (b\_repaint) repaint();

}

// Scale

void Frame::**scaleX**(double scaleValue)

{

QMatrix4x4 R

(

scaleValue, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**scaleY**(double scaleValue)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, scaleValue, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**scaleZ**(double scaleValue)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, scaleValue, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

// Move

void Frame::**moveToCoord**(double posX, double posY, double posZ)

{

QMatrix4x4 R

(

1, 0, 0, posX,

0, 1, 0, posY,

0, 0, 1, posZ,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

// Reflect

void Frame::**reflectX**()

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, -1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**reflectY**()

{

QMatrix4x4 R

(

-1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**reflectZ**()

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, -1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**calculate**(QMatrix4x4 &R)

{

for (int i = 0; i < dataPoints.size(); i++)

{

QVector4D vectCoord(dataPoints[i].x, dataPoints[i].y, dataPoints[i].z, 1);

QVector4D result = R \* vectCoord;

dataPoints[i].x = result[0];

dataPoints[i].y = result[1];

dataPoints[i].z = result[2];

}

}

/\* ------------------ Upload data for figure ------------------ \*/

bool Frame::**upload**(QString initPathPolygons, QString initPathPoints)

{

pathPolygons = initPathPolygons;

pathPoints = initPathPoints;

bool resultFillPolygons = fillingDataPolygons();

bool resultFillPoints = fillingDataPoints();

repaint();

if (resultFillPolygons && resultFillPoints) return true;

}

bool Frame::**fillingDataPolygons**()

{

dataPolygons.clear();

QFile file(pathPolygons);

if(!file.*open*(QIODevice::ReadOnly))

{

QMessageBox::warning(this, "Предупреждение!", "Файл " + pathPolygons + " не найден!");

return false;

}

while (!file.*atEnd*())

{

QStringList list;

QString tmpStr = file.readLine();

list = tmpStr.split(QRegExp(" "));

QVector<int> tmpVec;

for (int i = 1; i <= list.at(0).toInt(); i++)

{

tmpVec.push\_back(list.at(i).toInt());

}

dataPolygons.push\_back(tmpVec);

}

file.*close*();

return true;

}

bool Frame::**fillingDataPoints**()

{

dataPoints.clear();

QFile file(pathPoints);

if(!file.*open*(QIODevice::ReadOnly))

{

QMessageBox::warning(this, "Предупреждение!", "Файл " + pathPoints + " не найден!");

return false;

}

while (!file.*atEnd*())

{

QStringList list;

QString tmp = file.readLine();

list = tmp.split(QRegExp(" "));

dataPoints.push\_back({100 \* list[0].toDouble(), 100 \* list[1].toDouble(), 100 \* list[2].toDouble()});

}

\

file.*close*();

return true;

}

### 3.3.4. Результаты работы программы

Результаты работы программы приведены на рис. 3.1 и 3.2.

|  |  |
| --- | --- |
|  |  |
| *Рис. 3.1. Кристалл 1* | *Рис. 3.2. Кристалл 2* |

## 3.4. Контрольные вопросы

1. Какие алгоритмы удаления скрытых линий используют при работе в пространстве объекта?
2. Как в вашей программе реализовано информационное обеспечение работы алгоритма Вейлера‑Азертона?
3. Какую особенность имеет алгоритм Вейлера‑Азертона для удаления скрытых линий в случае применения его для выпуклых многогранников?
4. Какие достоинства и недостатки имеет применение алгоритма Вейлера‑Азертона?
5. Для решения каких задач геометрического моделирования используется алгоритм Вейлера‑Азертона?

# Работа 4. Закраска поверхностей тел методом Гуро

**Цель работы:** написать программу, реализующую закраску поверхности многогранника методом Гуро.

## 4.1. Порядок выполнения работы

1. Ознакомиться со способами закраски полигональных сеток.
2. Написать программу закраски поверхности многогранника.
3. Выполнить закраску поверхности многогранников при различных положениях источника света.
4. Составить отчёт, который должен содержать:

— задание и рисунки многогранников,

— описание алгоритма закраски поверхностей методом Гуро для многогранников,

— результаты работы программы закраски поверхностей тел методом Гуро,

— выводы о проделанной работе.

— текст программы закраски поверхностей тел методом Гуро.

1. Ответить устно на контрольные вопросы.

## 4.2. Методические указания

Объект, представленный или аппроксимированный гранями, с учётом освещения точечным источником света может быть закрашен разными способами. Существуют три способа закраски объектов, заданных полигональными сетками:

□ однотонная закраска;

□ интерполяция интенсивностей (метод Гуро);

□ интерполяция векторов нормали (метод Фонга).

### 4.2.1. Метод Гуро

**Метод Гуро** позволяет получать сглаженный объект на этапе визуализации без внесения изменений в геометрическую модель (полигональные сетки). Полосы Маха значительно уменьшаются. Происходит сглаживание рёбер. Поэтому данный метод используется для объектов, аппроксимированных гранями.

Процесс закраски осуществляется в четыре этапа.

1. Вычисляются нормали к поверхностям

2. Определяются нормали в вершинах путём усреднения нормалей по всем граням, которым принадлежит вершина (рис. 4.1):

3. Используя нормали в вершинах и применяя произвольный метод закраски, вычисляются значения интенсивности в вершинах.

4. Каждый многоугольник закрашивается путём линейной интерполяции значений интенсивности в вершинах сначала вдоль каждого ребра, а затем между рёбрами вдоль каждой сканирующей строки (рис. 4.2):

Изображение выглядит как текст, антенна, день

Автоматически созданное описание

Рис. 4.1. Определение нормалей

Изображение выглядит как антенна

Автоматически созданное описание

Рис. 4.2. Определение значений интенсивности путём линейной интерполяции

Интерполяция вдоль рёбер легко объединяется с алгоритмом удаления скрытых поверхностей, построенным на принципе построчного сканирования. Для всех рёбер запоминается начальное значение интенсивности, а также изменение интенсивности при каждом единичном шаге по координате у. Заполнение видимого интервала на сканирующей строке производится путём интерполяции между значениями интенсивности на двух рёбрах, ограничивающих интервал. Для цветных объектов отдельно интерполируется каждая из компонент цвета.

### 4.2.2. Определение вектора отражения

Для модели освещения чрезвычайно важно правильно задавать направление векторов отражения. В данном разделе рассматриваются три наиболее общих метода. По закону отражения вектор падающего света, нормаль к поверхности и вектор отражения лежат в одной плоскости, причём на этой плоскости угол падения равен углу отражения (рис. 4.3, а). Фонг вывел отсюда простое решение для случая, когда свет падает вдоль оси *z*. Это предположение удобно для модели освещения с одним точечным источником. Если начало системы координат перенести в точку поверхности, то проекции нормали и вектора отражения на плоскость *ху* будут лежать на одной прямой (рис. 4.3. b).

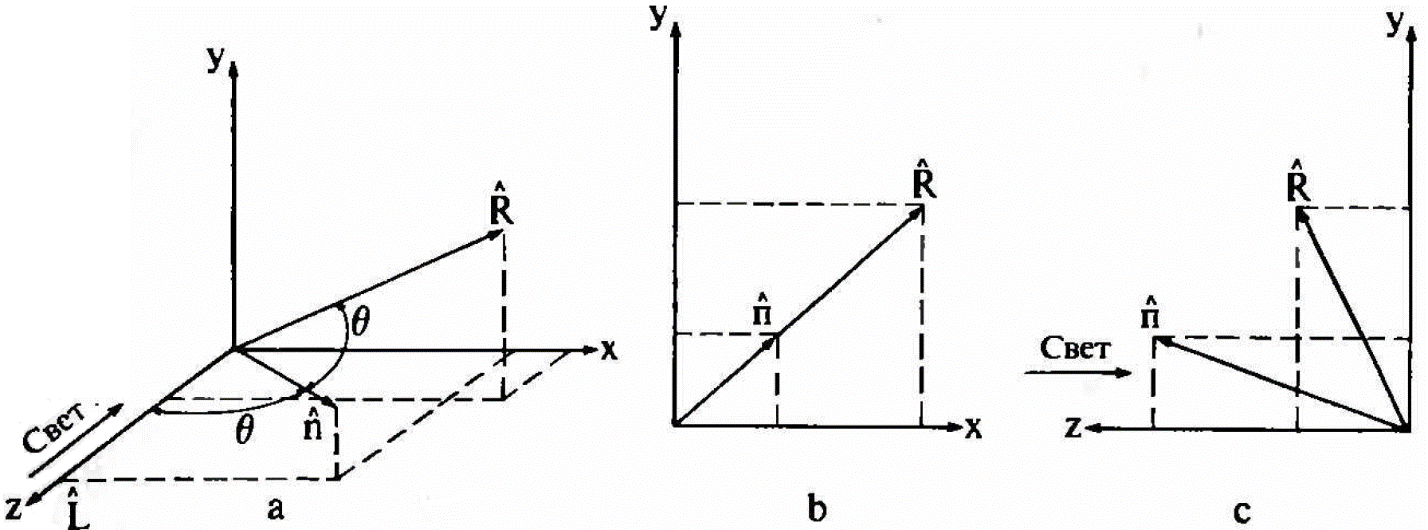


Рис. 4.3. Расчёт направления отражения.

Таким образом,

где — суть *x*- и *y*-составляющие единичных векторов соответственно отражения и нормали.

Обозначим угол между единичным вектором нормали и осью *z* через *θ*. Тогда составляющая вектора нормали по оси *z* есть

Аналогично, угол между единичным вектором отражения и осью *z* равен 2*θ*, поэтому

Известно, что

и

или

Используя отношение *х*- и *y*-составляющих векторов отражения и нормали (4.1), и соотношение

получаем

Перепишем правую часть:

или

Из соотношения (4.1) имеем

Когда свет падает не по оси *z* (например, когда источников несколько), таким методом воспользоваться нельзя. Можно перенести и повернуть каждый источник так, чтобы свет падал вдоль оси *z*; однако проще перенести и повернуть нормаль так, чтобы она была параллельна оси *z*, а точку *Р* поверхности принять за начало координат. Тогда плоскость *ху* будет касательной к поверхности, а *х*- и *y*-составляющие единичных векторов падения и отражения будут иметь разные знаки; *z*-составляющие этих векторов, будут, конечно, равны. Для того чтобы получить результаты в первоначальной ориентации, надо выполнить обратное преобразование. В перемещённой и повёрнутой системе координат

Этот метод особенно удобен, когда преобразования реализованы аппаратно или микропрограммно. В третьем методе условие того, что единичная нормаль, единичный вектор падения и единичный вектор отражения лежат в одной плоскости, записывается при помощи их векторных произведений. Равенство углов падения и отражения выражается через скалярные произведения этих векторов. Из данных условий получаем

**n ´**×**L** = **R ´** **n**

или

Векторные произведения совпадают, если равны их *х*-, *у*- и *z*‑составляющие:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  | (4.5) |
|  |  |  |  |  |

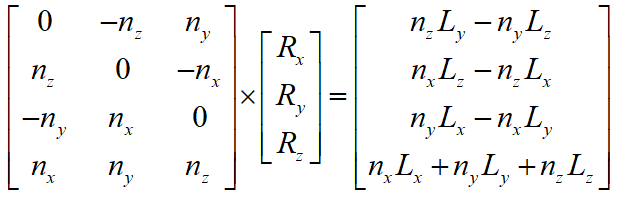
На первый взгляд может показаться, что вектор отражения найден. К сожалению, одно из этих уравнений всегда является линейно зависимым.

Запишем равенство углов падения и отражения:

**n •** **L** = **R** • **n**

или

что даёт необходимое добавочное условие. Матричный вид для всех уравнений с тремя неизвестными *Rx*, *Ry* и *Rz* таков:



или

Поскольку матрица [*N*] не является квадратной, для решения системы нужны особые методы. В частности,

## 4.3. Пример программы закраски поверхностей тел методом Гуро

Программа лабораторной работы №4 содержит описание классов: **Cristall** для описания главной экранной формы и **Frame** для описания кристалла**.** Файл «cristall.h» содержит описания структуры класса, файл «cristall.cpp» содержит описания функций обработки событий в окне.Файл «Frame.h» содержит описания структуры класса, файл «Frame.cpp» содержит описания функций обработки кристала

Ввод исходных данных кристаллов выполняется из текстовых файлов в структуру, описанную в классе Frame при помощи встроенной функции file.readLine().

Аффинные преобразования выполняются при помощи функций: Frame::rotateZLeft(), Frame::rotateZRight(),Frame::rotateX(),Frame::rotateY() – вращение; Frame::scaleX(double scaleValue), Frame::scaleY(double scaleValue), scaleZ(double scaleValue) – масштабирование, Frame::moveToCoord(double posX, double posY, double posZ) – перемещение; Frame::reflectX(),Frame::reflectY(),Frame::reflectZ() – отражение.

Формирование и вывод буфера выполняется функциями Frame::**drawFigureZBuffer**(),Frame::**drawFigureVeyler**(),Frame::**fillPolygon**(int idSegment, QVector<intCoord> &points), Frame::**customLine**(int idSegment, intCoord &p1, intCoord &p2, QMap<int, QVector<intCoord>> &boundMap)

Вывод на экран – встроенная функция repaint() и функция Frame::drawFigure() класса Frame

### 4.4.1. Cristall.h

#ifndef CRISTALL\_H

#define CRISTALL\_H

#include <QMainWindow>

#include <QFileDialog>

#include <QMessageBox>

#include <QString>

#include <Frame.h>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **Cristall**; }

QT\_END\_NAMESPACE

class **Cristall** : public QMainWindow

{

Q\_OBJECT

public:

**Cristall**(QWidget \*parent = nullptr);

~***Cristall***();

private slots:

void **slotAddPolygons**();

void **slotAddPoints**();

void **slotUploadFiles**();

void **slotExit**();

void **slotChangeAngle**();

void **on\_goStepX\_clicked**();

void **on\_goStepY\_clicked**();

void **on\_goStepZ\_clicked**();

void **on\_goScaleX\_clicked**();

void **on\_goScaleY\_clicked**();

void **on\_goScaleZ\_clicked**();

void **on\_goMove\_clicked**();

void **on\_goReflectX\_clicked**();

void **on\_goReflectY\_clicked**();

void **on\_goReflectZ\_clicked**();

void **slotZBuffer**(int);

void **slotVeyler**(int);

void **slotGuro**(int);

private:

Ui::Cristall \*ui;

QString pathPolygons;

QString pathPoints;

Frame \*canvas;

};

#endif // CRISTALL\_H

### 4.4.2. Frame.h

#ifndef FRAME\_H

#define FRAME\_H

#include <QWidget>

#include <QPainter>

#include <QVector>

#include <QFile>

#include <QMessageBox>

#include <QtMath>

#include <QDebug>

#include <QMatrix4x4>

#include <QMatrix>

#include <QMouseEvent>

#include <QWheelEvent>

class **Frame** : public QWidget

{

Q\_OBJECT

public:

**Frame**(QWidget \* parent = 0);

bool **upload**(QString, QString);

void **setFiX**(double initFiX){FiX = initFiX \* M\_PI / 180;};

void **setFiY**(double initFiY){FiY = initFiY \* M\_PI / 180;};

void **setFiZ**(double initFiZ){FiZ = initFiZ \* M\_PI / 180;};

// Rotation

void **rotateX**(bool b\_repaint);

void **rotateY**(bool b\_repaint);

void **rotateZ**(QString, bool b\_repaint);

// Scale

void **scaleX**(double);

void **scaleY**(double);

void **scaleZ**(double);

// Move

void **moveToCoord**(double, double, double);

// Reflect

void **reflectX**();

void **reflectY**();

void **reflectZ**();

// Set option draw

void **setOptionDraw**(int value){ optionDraw = value; };

void **setOptionFill**(bool value){ optionFill = value; };

protected:

void ***paintEvent***(QPaintEvent \*) override;

void ***mousePressEvent***(QMouseEvent \*event) override

{

if (event->button() == Qt::LeftButton)

{

\_p = event->pos();

rotationX = \_p.x() - 250;

rotationY = \_p.y() - 250;

}

}

void ***mouseMoveEvent***(QMouseEvent \*) override;

void ***wheelEvent***(QWheelEvent \*event) override

{

if (event->delta() > 0)

{

rotateZLeft();

}

else

{

rotateZRight();

}

};

private:

QString pathPolygons;

QString pathPoints;

QPainter painter;

int optionDraw = 0;

int optionFill = false;

static uint const sizeCanvas = 500;

QPoint \_p;

QImage screen;

int buffFrame[sizeCanvas][sizeCanvas];

double buffZ[sizeCanvas][sizeCanvas];

double FiX = 0.07;

double FiY = 0.07;

double FiZ = 0.07;

int rotationX;

int rotationY;

struct **intCoord**{

int x, y;

double z;

};

struct **coord**{

double x, y, z;

};

coord lightCoord;

QVector<QVector<int>> dataPolygons;

QVector<coord> dataPoints;

/\* --------- Functions --------- \*/

void **defaultDrawFigure**();

void **drawFigureZBuffer**();

void **drawFigureVeyler**();

bool **fillingDataPolygons**();

bool **fillingDataPoints**();

void **rotateZLeft**();

void **rotateZRight**();

void **calculate**(QMatrix4x4 &);

void **fillPolygon**(int, QVector<intCoord>&);

void **customLine**(int, intCoord&, intCoord&, QMap<int, QVector<intCoord>>&);

void **addInBuffFrame**(int, int, int);

};

#endif // FRAME\_H

Cristall.cpp

#include "cristall.h"

#include "ui\_cristall.h"

Cristall::**Cristall**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::Cristall)

{

ui->setupUi(this);

connect(ui->addPolygons, SIGNAL(triggered()), this, SLOT(slotAddPolygons()));

connect(ui->addPoints, SIGNAL(triggered()), this, SLOT(slotAddPoints()));

connect(ui->uploadFiles, SIGNAL(clicked()), this, SLOT(slotUploadFiles()));

connect(ui->exit, SIGNAL(triggered()), this, SLOT(slotExit()));

connect(ui->FiX, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

connect(ui->FiY, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

connect(ui->FiZ, SIGNAL(textChanged(QString)), this, SLOT(slotChangeAngle()));

connect(ui->checkBox\_1, SIGNAL(stateChanged(int)), this, SLOT(slotZBuffer(int)));

connect(ui->checkBox\_2, SIGNAL(stateChanged(int)), this, SLOT(slotVeyler(int)));

connect(ui->checkBox\_3, SIGNAL(stateChanged(int)), this, SLOT(slotGuro(int)));

canvas = new Frame(this);

ui->canvasLayout->addWidget(canvas);

}

Cristall::~***Cristall***()

{

canvas->deleteLater();

delete ui;

}

void Cristall::**slotAddPolygons**()

{

QString tmp = QFileDialog::getOpenFileName(0, "Choose file with polygons", "", "\*.txt");

if (tmp == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл!");

return;

}

else

{

pathPolygons = tmp;

ui->uiPathPolygons->setText(pathPolygons);

ui->uploadFiles->setText("Загрузить");

}

}

void Cristall::**slotAddPoints**()

{

QString tmp = QFileDialog::getOpenFileName(0, "Choose file with points", "", "\*.txt");

if (tmp == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл!");

return;

}

else

{

pathPoints = tmp;

ui->uiPathPoints->setText(pathPoints);

ui->uploadFiles->setText("Загрузить");

}

}

void Cristall::**slotUploadFiles**()

{

if (pathPolygons == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл полигонов!");

return;

}

if (pathPoints == 0)

{

QMessageBox::warning(this, "Внимание!", "Вы не выбрали файл вершин!");

return;

}

bool resultUpload = canvas->upload(pathPolygons, pathPoints);

if (resultUpload)

{

ui->uploadFiles->setText("Сбросить");

}

}

void Cristall::**slotExit**()

{

this->close();

}

void Cristall::**slotChangeAngle**()

{

canvas->setFiX(ui->FiX->text().toDouble());

canvas->setFiY(ui->FiY->text().toDouble());

canvas->setFiZ(ui->FiZ->text().toDouble());

}

void Cristall::**on\_goStepX\_clicked**()

{

canvas->rotateX(true);

}

void Cristall::**on\_goStepY\_clicked**()

{

canvas->rotateY(true);

}

void Cristall::**on\_goStepZ\_clicked**()

{

canvas->rotateZ("Right", true);

}

void Cristall::**on\_goScaleX\_clicked**()

{

canvas->scaleX(ui->scaleX->text().toDouble());

}

void Cristall::**on\_goScaleY\_clicked**()

{

canvas->scaleY(ui->scaleY->text().toDouble());

}

void Cristall::**on\_goScaleZ\_clicked**()

{

canvas->scaleZ(ui->scaleZ->text().toDouble());

}

void Cristall::**on\_goMove\_clicked**()

{

canvas->moveToCoord(ui->posX->text().toDouble(), ui->posY->text().toDouble(), ui->posZ->text().toDouble());

}

void Cristall::**on\_goReflectX\_clicked**()

{

canvas->reflectX();

}

void Cristall::**on\_goReflectY\_clicked**()

{

canvas->reflectY();

}

void Cristall::**on\_goReflectZ\_clicked**()

{

canvas->reflectZ();

}

void Cristall::**slotZBuffer**(int check)

{

if (check == 2)

{

canvas->setOptionDraw(1);

ui->checkBox\_1->setCheckState(Qt::Checked);

ui->checkBox\_2->setCheckState(Qt::Unchecked);

}else{

if (!ui->checkBox\_2->isChecked())

{

canvas->setOptionDraw(0);

}

}

canvas->update();

}

void Cristall::**slotVeyler**(int check)

{

if (check == 2)

{

canvas->setOptionDraw(2);

ui->checkBox\_1->setCheckState(Qt::Unchecked);

ui->checkBox\_2->setCheckState(Qt::Checked);

}else{

if (!ui->checkBox\_1->isChecked())

{

canvas->setOptionDraw(0);

}

}

canvas->update();

}

void Cristall::**slotGuro**(int check)

{

if (check == 2)

{

canvas->setOptionFill(true);

}else{

canvas->setOptionFill(false);

}

canvas->update();

}

### 4.4.3. Frame.cpp

#include "Frame.h"

Frame::**Frame**(QWidget \*parent) : QWidget(parent)

{

QPen pen(Qt::black, 1, Qt::DashLine, Qt::SquareCap, Qt::RoundJoin);

painter.setPen(pen);

screen = QImage(sizeCanvas - 1, sizeCanvas - 1, QImage::Format\_ARGB32);

lightCoord.x = 200;

lightCoord.y = 300;

lightCoord.x = 50;

}

/\* ------------------ Draw Figure ------------------ \*/

void Frame::***paintEvent***(QPaintEvent \*event)

{

Q\_UNUSED(event);

painter.begin(this);

switch(optionDraw)

{

case 0:

defaultDrawFigure();

break;

case 1:

drawFigureZBuffer();

break;

case 2:

drawFigureVeyler();

break;

}

painter.end();

}

void Frame::**defaultDrawFigure**()

{

for (int i = 0; i < dataPolygons.size(); i++)

{

QPointF points[dataPolygons[i].size()];

for (int j = 0; j < dataPolygons[i].size(); j++)

{

points[j] = QPointF(dataPoints[dataPolygons[i][j]].x + 250, dataPoints[dataPolygons[i][j]].y + 250);

}

painter.drawPolygon(points, dataPolygons[i].size());

}

}

void Frame::**drawFigureZBuffer**()

{

screen.fill(QColor(Qt::white).rgb());

for (uint x = 0; x < sizeCanvas; x++)

{

for (uint y = 0; y < sizeCanvas; y++)

{

buffFrame[x][y] = 0;

buffZ[x][y] = -1000;

}

}

for (int i = 0; i < dataPolygons.size(); i++)

{

QVector<intCoord> points;

for (int j = 0; j < dataPolygons[i].size(); j++)

{

points.push\_back

({

int(dataPoints[dataPolygons[i][j]].x + 250 + 0.5),

int(dataPoints[dataPolygons[i][j]].y + 250 + 0.5),

dataPoints[dataPolygons[i][j]].z

});

}

fillPolygon(i + 1, *points*);

}

painter.drawImage(1, 1, screen);

}

void Frame::**drawFigureVeyler**()

{

drawFigureZBuffer();

}

void Frame::**fillPolygon**(int idSegment, QVector<intCoord> &points)

{

QMap<int, QVector<intCoord>> boundMap;

for (int i = 0; i < points.size() - 1; customLine(idSegment, *points[i]*, *points[i* *+* 1*]*, *boundMap*), i++);

customLine(idSegment, *points.last()*, *points[*0*]*, *boundMap*);

int distance = sqrt(pow((lightCoord.x - 250), 2) + pow((lightCoord.y - 250), 2) + pow(lightCoord.z, 2));

foreach (int key, boundMap.keys())

{

QVector<intCoord> value = boundMap.value(key);

for (int i = value[0].y; i < value[1].y; i++)

{

if (buffFrame[key][i] != idSegment)

{

double tmp = value[0].z + double(value[1].z - value[0].z) \* double(double(i - value[0].y) / double(value[1].y - value[0].y));

if (tmp >= buffZ[key][i])

{

buffFrame[key][i] = 0;

int tmpDistance = sqrt(pow((lightCoord.x - key), 2) + pow((lightCoord.y - i), 2) + pow((lightCoord.z - tmp), 2));

double betweenDistance = distance - tmpDistance \* 0.8;

int alpha = 255.0 \* (1.0 - betweenDistance / 100.0) < 255 ? 255.0 \* (1.0 - betweenDistance / 100.0) : 255;

alpha = alpha < 0 ? 0 : alpha;

switch(optionDraw)

{

case 1:

optionFill ?

screen.setPixelColor(key, i, QColor(0, 20, 255, alpha)):

screen.setPixelColor(key, i, 4294967295);

break;

case 2:

optionFill ?

screen.setPixelColor(key, i, QColor(0, 20, 255, alpha)):

screen.setPixelColor(key, i, 4294967295);

break;

}

buffZ[key][i] = tmp;

}

}

}

}

}

void Frame::**customLine**(int idSegment, intCoord &p1, intCoord &p2, QMap<int, QVector<intCoord>> &boundMap)

{

const int deltaX = abs(p2.x - p1.x);

const int deltaY = abs(p2.y - p1.y);

const int signX = p1.x < p2.x ? 1 : -1;

const int signY = p1.y < p2.y ? 1 : -1;

int error = deltaX - deltaY;

int x = p1.x,

y = p1.y;

double tmp;

while(x != p2.x || y != p2.y)

{

if (p1.x == p2.x)

{

tmp = p1.z + double(p2.z - p1.z) \* double(double(y - p1.y) / double(p2.y - p1.y));

}else

{

tmp = p1.z + double(p2.z - p1.z) \* double(double(x - p1.x) / double(p2.x - p1.x));

}

if (tmp >= buffZ[x][y])

{

buffFrame[x][y] = idSegment;

screen.setPixelColor(x, y, 4278190080); // Black

buffZ[x][y] = tmp;

}

if (boundMap.find(x) == boundMap.end())

{

intCoord boundCoord;

boundCoord.y = y;

boundCoord.z = tmp;

boundMap.insert(x, {boundCoord, boundCoord});

}else{

if (boundMap[x][0].y > y)

{

boundMap[x][0].y = y;

boundMap[x][0].z = tmp;

}else if(boundMap[x][1].y < y)

{

boundMap[x][1].y = y;

boundMap[x][1].z = tmp;

}

}

//Logical draw

const int error2 = error \* 2;

if(error2 > -deltaY)

{

error -= deltaY;

x += signX;

}

if(error2 < deltaX)

{

error += deltaX;

y += signY;

}

}

}

/\* ------------------ Figure operations------------------ \*/

void Frame::***mouseMoveEvent***(QMouseEvent \*event)

{

if (!\_p.isNull())

{

if (abs(rotationX - (event->x() - int(sizeCanvas / 2))) > 5 && abs(rotationY - (event->y() - int(sizeCanvas / 2))) > 5)

{

if (rotationX > (event->x() - int(sizeCanvas / 2)))

{

if (rotationY > (event->y() - int(sizeCanvas / 2)))

{

if ( FiX > 0) FiX = -FiX;

rotateX(false);

FiY = abs(FiY);

rotateY(false);

}else

{

FiX = abs(FiX);

rotateX(false);

FiY = abs(FiY);

rotateY(false);

}

}else

{

if (rotationY > (event->y() - int(sizeCanvas / 2)))

{

if ( FiX > 0) FiX = -FiX;

rotateX(false);

if ( FiY > 0) FiY = -FiY;

rotateY(false);

}else

{

FiX = abs(FiX);

rotateX(false);

if ( FiY > 0) FiY = -FiY;

rotateY(false);

}

}

rotationX = event->x() - int(sizeCanvas / 2);

rotationY = event->y() - int(sizeCanvas / 2);

}

}

repaint();

}

// Rotation

void Frame::**rotateZLeft**()

{

rotateZ("Left", true);

}

void Frame::**rotateZRight**()

{

rotateZ("Right", true);

}

void Frame::**rotateX**(bool b\_repaint)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, cos(FiX), sin(FiX), 0,

0, -sin(FiX), cos(FiX), 0,

0, 0, 0, 1

);

calculate(*R*);

if (b\_repaint) repaint();

}

void Frame::**rotateY**(bool b\_repaint)

{

QMatrix4x4 R

(

cos(FiY), 0, -sin(FiY), 0,

0, 1, 0, 0,

sin(FiY), 0, cos(FiY), 0,

0, 0, 0, 1

);

calculate(*R*);

if (b\_repaint) repaint();

}

void Frame::**rotateZ**(QString diraction, bool b\_repaint)

{

if (diraction == "Left")

{

FiZ = abs(FiZ);

}else if(diraction == "Right")

{

if ( FiZ > 0) FiZ = -FiZ;

}

QMatrix4x4 R

(

cos(FiZ), sin(FiZ), 0, 0,

-sin(FiZ), cos(FiZ), 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

if (b\_repaint) repaint();

}

// Scale

void Frame::**scaleX**(double scaleValue)

{

QMatrix4x4 R

(

scaleValue, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**scaleY**(double scaleValue)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, scaleValue, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**scaleZ**(double scaleValue)

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, scaleValue, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

// Move

void Frame::**moveToCoord**(double posX, double posY, double posZ)

{

QMatrix4x4 R

(

1, 0, 0, posX,

0, 1, 0, posY,

0, 0, 1, posZ,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

// Reflect

void Frame::**reflectX**()

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, -1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**reflectY**()

{

QMatrix4x4 R

(

-1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**reflectZ**()

{

QMatrix4x4 R

(

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, -1, 0,

0, 0, 0, 1

);

calculate(*R*);

repaint();

}

void Frame::**calculate**(QMatrix4x4 &R)

{

for (int i = 0; i < dataPoints.size(); i++)

{

QVector4D vectCoord(dataPoints[i].x, dataPoints[i].y, dataPoints[i].z, 1);

QVector4D result = R \* vectCoord;

dataPoints[i].x = result[0];

dataPoints[i].y = result[1];

dataPoints[i].z = result[2];

}

}

/\* ------------------ Upload data for figure ------------------ \*/

bool Frame::**upload**(QString initPathPolygons, QString initPathPoints)

{

pathPolygons = initPathPolygons;

pathPoints = initPathPoints;

bool resultFillPolygons = fillingDataPolygons();

bool resultFillPoints = fillingDataPoints();

repaint();

if (resultFillPolygons && resultFillPoints) return true;

}

bool Frame::**fillingDataPolygons**()

{

dataPolygons.clear();

QFile file(pathPolygons);

if(!file.*open*(QIODevice::ReadOnly))

{

QMessageBox::warning(this, "Предупреждение!", "Файл " + pathPolygons + " не найден!");

return false;

}

while (!file.*atEnd*())

{

QStringList list;

QString tmpStr = file.readLine();

list = tmpStr.split(QRegExp(" "));

QVector<int> tmpVec;

for (int i = 1; i <= list.at(0).toInt(); i++)

{

tmpVec.push\_back(list.at(i).toInt());

}

dataPolygons.push\_back(tmpVec);

}

file.*close*();

return true;

}

bool Frame::**fillingDataPoints**()

{

dataPoints.clear();

QFile file(pathPoints);

if(!file.*open*(QIODevice::ReadOnly))

{

QMessageBox::warning(this, "Предупреждение!", "Файл " + pathPoints + " не найден!");

return false;

}

while (!file.*atEnd*())

{

QStringList list;

QString tmp = file.readLine();

list = tmp.split(QRegExp(" "));

dataPoints.push\_back({100 \* list[0].toDouble(), 100 \* list[1].toDouble(), 100 \* list[2].toDouble()});

}

\

file.*close*();

return true;

}

### 4.4.4. Результаты работы программы

Результаты работы программы приведены на рис. 4.1 и 4.2.

|  |  |
| --- | --- |
|  |  |
| *Рис. 4.1. Кристалл 1* | *Рис. 4.2. Кристалл 2* |

## 4.4. Контрольные вопросы

1. Какой алгоритм управления положением источника света использован в вашей программе?
2. Как в вашей программе реализовано информационное обеспечение работы алгоритма закраски поверхностей тел методом Гуро?
3. Какие особенности имеет алгоритм закраски поверхностей тел методом Гуро для создания фотореалистических?
4. Какие достоинства и недостатки имеет применение алгоритма закраски поверхностей тел методом Гуро?
5. Для решения каких задач геометрического моделирования используется алгоритм закраски поверхностей тел методом Гуро?

# Список литературы

1. Роджерс Д. Алгоритмические основы машинной графики: Пер.с англ. М.: Мир, 1989. – 512с.
2. Шаскольская М.П. Кристаллография. М.: Высш.шк., 1984. – 376с.
3. Сиденко Л. А. Компьютерная графика и геометрическое моделирование: Учебное пособие. СПб.: Питер, 2009. — 224 с.: ил.

# Приложения

## Варианты заданий по лабораторным работам

Таблица П1

Варианты заданий

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N/N  п/п | Кристалл | | N/N  п/п | Кристалл | | N/N  п/п | Кристалл | | N/N  п/п | Кристалл | |
| №1 | №2 | №1 | №2 | №1 | №2 | №1 | №2 |
| 1 | 1 | 1 | 24 | 23 | 9 | 47 | 23 | 23 | 70 | 21 | 9 |
| 2 | 2 | 2 | 25 | 21 | 8 | 48 | 22 | 22 | 71 | 23 | 8 |
| 3 | 3 | 3 | 26 | 20 | 7 | 49 | 21 | 21 | 72 | 20 | 7 |
| 4 | 4 | 4 | 27 | 19 | 6 | 50 | 20 | 20 | 73 | 19 | 6 |
| 5 | 5 | 5 | 28 | 18 | 22 | 51 | 19 | 19 | 74 | 18 | 22 |
| 6 | 6 | 6 | 29 | 17 | 21 | 52 | 18 | 18 | 75 | 12 | 21 |
| 7 | 7 | 7 | 30 | 16 | 20 | 53 | 17 | 17 | 76 | 13 | 20 |
| 8 | 8 | 8 | 31 | 15 | 19 | 54 | 16 | 16 | 77 | 14 | 19 |
| 9 | 9 | 9 | 32 | 14 | 18 | 55 | 15 | 15 | 78 | 15 | 18 |
| 10 | 10 | 10 | 33 | 13 | 17 | 56 | 14 | 14 | 79 | 16 | 17 |
| 11 | 11 | 11 | 34 | 12 | 16 | 57 | 13 | 13 | 80 | 17 | 16 |
| 12 | 12 | 12 | 35 | 11 | 15 | 58 | 12 | 12 | 81 | 22 | 15 |
| 13 | 13 | 13 | 36 | 10 | 14 | 59 | 11 | 11 | 82 | 11 | 14 |
| 14 | 14 | 14 | 37 | 9 | 13 | 60 | 10 | 10 | 83 | 10 | 13 |
| 15 | 15 | 15 | 38 | 8 | 12 | 61 | 9 | 9 | 84 | 8 | 12 |
| 16 | 16 | 16 | 39 | 7 | 3 | 62 | 8 | 8 | 85 | 9 | 3 |
| 17 | 17 | 17 | 40 | 6 | 23 | 63 | 7 | 7 | 86 | 1 | 23 |
| 18 | 18 | 18 | 41 | 5 | 4 | 64 | 6 | 6 | 87 | 7 | 4 |
| 19 | 19 | 19 | 42 | 4 | 5 | 65 | 5 | 5 | 88 | 2 | 5 |
| 20 | 20 | 20 | 43 | 3 | 11 | 66 | 4 | 4 | 89 | 3 | 11 |
| 21 | 21 | 21 | 44 | 2 | 1 | 67 | 3 | 3 | 90 | 4 | 1 |
| 22 | 22 | 22 | 45 | 1 | 2 | 68 | 2 | 2 | 91 | 5 | 2 |
| 23 | 23 | 23 | 46 | 22 | 10 | 69 | 1 | 1 | 92 | 6 | 10 |

Варианты кристаллов приведены на рис.П1 и рис. П2.

|  |  |
| --- | --- |
|  | Рис. П1. Варианты заданий кристалла №1 |

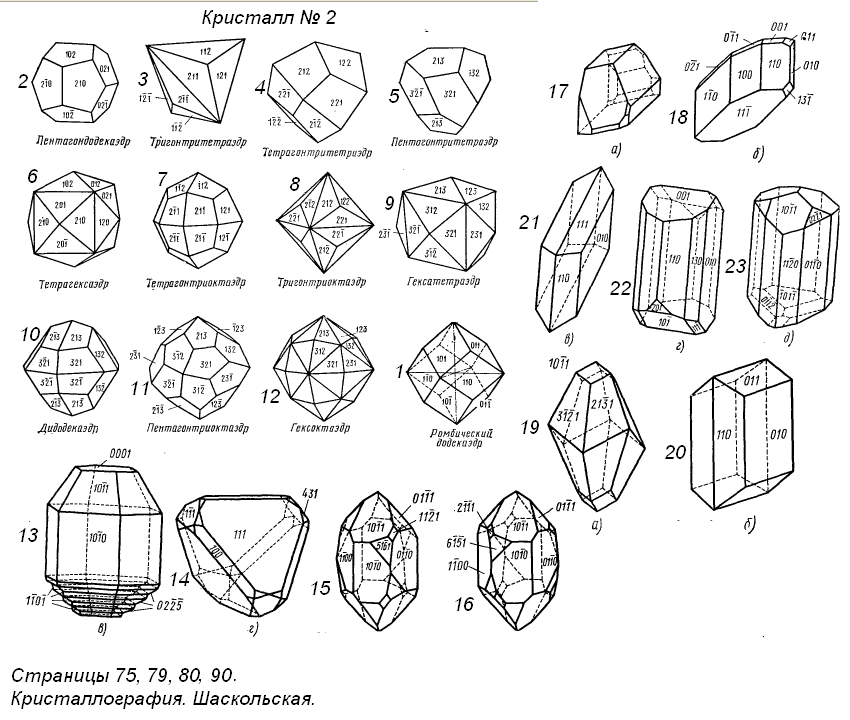


Рис. П2. Варианты заданий кристалла №2

# Содержание

[Предисловие 3](#_Toc96795655)

[Работа 1. Аффинные преобразования 4](#_Toc96795656)

[1.1. Порядок выполнения работы 4](#_Toc96795657)

[1.2. Методические указания 4](#_Toc96795658)

[1.3. Пример программы аффинных преобразований вершин кристаллов 10](#_Toc96795659)

[1.3.1. Файл «cristall.cpp» 11](#_Toc96795660)

[1.3.2. Файл «Frame.cpp» 14](#_Toc96795661)

[1.3.3. Файл «cristall.h» 20](#_Toc96795662)

[1.3.4. Файл «Frame.h» 21](#_Toc96795663)

[1.3.5. Результаты работы программы. 23](#_Toc96795664)

[1.4. Контрольные вопросы 24](#_Toc96795665)

[Работа 2. Удаления невидимых линий методом Z‑буфера для многогранников 25](#_Toc96795666)

[2.1. Порядок выполнения работы 25](#_Toc96795667)

[2.2. Методические указания 25](#_Toc96795668)

[2.3. Пример программы удаления невидимых линий методом Z‑буфера 27](#_Toc96795669)

[2.3.1. Cristall.h 28](#_Toc96795670)

[2.3.2. Frame.h 29](#_Toc96795671)

[2.3.3. Frame.cpp 33](#_Toc96795672)

[2.3.4. Результаты работы программы 40](#_Toc96795673)

[2.4. Контрольные вопросы 41](#_Toc96795674)

[Работа 3. Удаления невидимых линий по алгоритму Вейлера‑Азертона 41](#_Toc96795675)

[3.1. Порядок выполнения работы 41](#_Toc96795676)

[3.2. Методические указания 41](#_Toc96795677)

[3.3. Пример программы удаления невидимых линий, реализующей алгоритм Вейлера — Азертона 45](#_Toc96795678)

[3.3.1. Cristall.h 46](#_Toc96795679)

[3.3.2. Frame.h 47](#_Toc96795680)

[3.3.3. Frame.cpp 52](#_Toc96795681)

[3.3.4. Результаты работы программы 64](#_Toc96795682)

[3.4. Контрольные вопросы 64](#_Toc96795683)

[Работа 4. Закраска поверхностей тел методом Гуро 64](#_Toc96795684)

[4.1. Порядок выполнения работы 64](#_Toc96795685)

[4.2. Методические указания 65](#_Toc96795686)

[4.2.1. Метод Гуро 65](#_Toc96795687)

[4.2.2. Определение вектора отражения 66](#_Toc96795688)

[4.3. Пример программы закраски поверхностей тел методом Гуро 69](#_Toc96795689)

[4.4.1. Cristall.h 69](#_Toc96795690)

[4.4.2. Frame.h 70](#_Toc96795691)

[4.4.3. Frame.cpp 78](#_Toc96795692)

[4.4.4. Результаты работы программы 90](#_Toc96795693)

[4.4. Контрольные вопросы 90](#_Toc96795694)

[Список литературы 91](#_Toc96795695)

[Приложения 92](#_Toc96795696)

[Варианты заданий по лабораторным работам 92](#_Toc96795697)