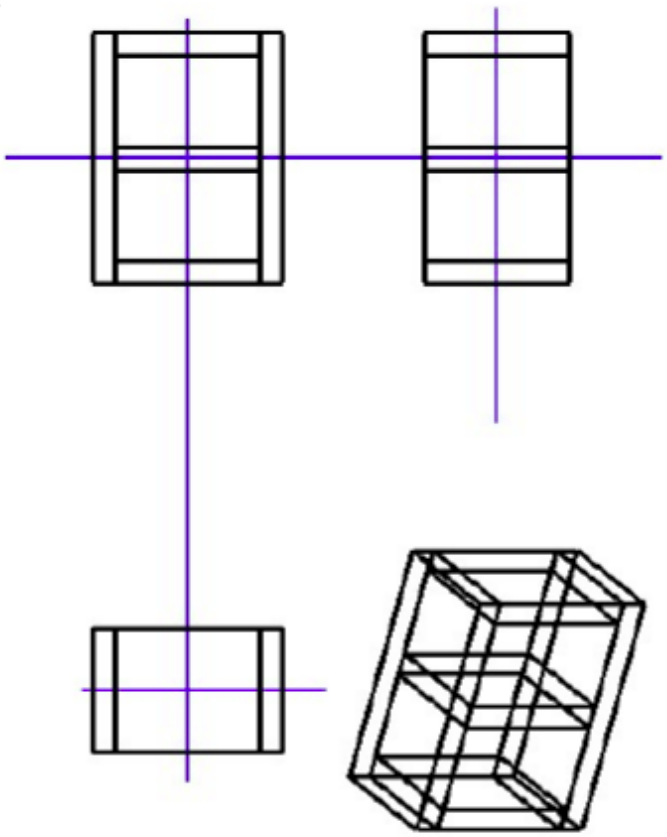
16		Ширина

		средней перегородки фигуры

4)Вывод необходимых формул для построения всех проекций. Указать какие матрицы используются для построения всех четырёх проекций изображений и в какой последовательности они умножаются.

Для построения используются матрицы Переноса и Поворота вокруг оси Y по сути по точкам строится только передняя часть тумбочки, а остальные 3 путем поворота уже имеющейся. При построении этой одной стороны используется матрица Переноса для отрисовки верхнего и нижнего яруса. Для построения проекций используется матрица Поворота вокруг Y(фронтальная проекция) и матрица Поворота вокруг X(вид сверху). Для отрисовки других проекций также используются матрицы Поворота вокруг осей X и Y с различным углом. Возможность уменьшать и увеличивать объект реализована через матрицу Масштабирования, а вращения через матрицу Поворота вокруг Y.

5)Текст программы.

Файл main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[]) {
  QApplication a(argc, argv);
  MainWindow w;
  w.show();

  return a.exec();
}
```

Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
      QMainWindow(parent),
      ui(new Ui::MainWindow),
      dilatation(DilatationM(1, 1, 1)),
      rotate(RotateMY(0)),
      flag(0),
      sizeCenter(15){
  ui->setupUi(this);
}

MainWindow::~MainWindow() {
  delete ui;
}

void MainWindow::paintEvent(QPaintEvent *p) {
  QWidget::paintEvent(p);
  QPainter painter(this);
  painter.setBrush(QBrush(Qt::white));
  painter.setPen(QPen(Qt::black, 5));
```

```
    // Радиус большей окружности
    float radius;
    // Вычисляем радиус окружности
    if (width() > height()) // Если ширина окна больше высоты
        radius = (height() - 20) / 2; // Отступ от краёв - 10 пикселей
    else radius = (width() - 20) / 2;

    Object staticTumba = Tumba(radius / 290., sizeCenter);

    // Координаты центра всего
    QPointF center = QPointF(width() / 2, height() / 2);

    painter.translate(center.x(), center.y());

    painter.translate(-radius / 2., -radius / 2.);
    staticTumba.drawFront(painter);

    painter.translate(radius, 0);
    staticTumba.drawPorf(painter);

    painter.translate(-radius, radius);
    staticTumba.drawUp(painter);

    painter.translate(radius, 0);
    drawDinamic(painter, radius);

    painter.translate(-radius / 2., -radius / 2.);
    painter.setPen(Qt::blue);
    drawAxis(painter, radius);

}

void MainWindow::wheelEvent(QWheelEvent *wheelevent) {
    double k = 1. + wheelevent->delta() / 10000.;
    Matr dilatationMnog = DilatationM(k, k, k);
```

```
    dilatation = dilatationMnog * dilatation;
    repaint();
}

void MainWindow::drawAxis(QPainter &painter, double radius) {
    painter.drawLine(-radius / 2., -radius, -radius / 2., radius);
    painter.drawLine(-radius, -radius / 2., radius, -radius / 2.);
}

void MainWindow::drawDinamic(QPainter &painter, double radius) {
    Object tumba = Tumba(radius / 290., sizeCenter);
    tumba = dilatation * tumba;
    tumba = rotate * tumba;

    Matr proect = RotateMY(45 * RAD);
    if (flag == 1) {
        //косоугольная кабинетная
        Matr rt_x = RotateMX(45 * RAD);
        proect = (rt_x * proect);
    } else if (flag == 2) {
        //косоугольная свободная
        Matr rt_x = RotateMX(30 * RAD);
        proect = (rt_x * proect);
    } else if (flag == 3) {
        //прямоугольная диметрия
        Matr rt_x = RotateMX((7 + 1. / 6.) * RAD);
        Matr rt_y = RotateMY((41 + 25./60.) * RAD);
        proect = (rt_x * rt_y * proect);
    } else if (flag == 4) {
        //прямоугольная изометрия
        Matr rt_x = RotateMX(30 * RAD);
        Matr rt_y = RotateMY(30 * RAD);
        proect = (rt_x * rt_y * proect);
    }
```

```
    tumba = proect * tumba;
    tumba.draw(painter);
}

void MainWindow::mouseMoveEvent(QMouseEvent *mouseEvent) {
    if (!oldXpos || !oldYpos) {
        oldXpos = cursor().pos().x();
        oldYpos = cursor().pos().y();
    }else {
        double newXPos = cursor().pos().x();
        double ky = 1. + (newXPos - oldXpos) / 10000.;

//      double newYPos = cursor().pos().y();
//      double kx = 1. + (newYPos - oldYpos) / 10000.;

        int mnogY = 1;
        if (newXPos < oldXpos)
            mnogY = -1;
        oldXpos = newXPos;

//      int mnogX = 1;
//      if (newYPos > oldYpos)
//          mnogX = -1;
//      oldYpos = newYPos;

        Matr rotateMnogY = RotateMY(mnogY * ky * RAD);
//      Matr rotateMnogX = RotateMZ(mnogX * kx * RAD);

        rotate = rotateMnogY * rotate;
//      rotate = rotateMnogX * rotate;
        repaint();
    }
}


void MainWindow::on_comboBox_currentIndexChanged(int index) {
```

```
    flag = index;
    repaint();
}

void MainWindow::on_textEdit_selectionChanged()
{
    sizeCenter = ui->textEdit->toPlainText().toInt();
}
```

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <random>

#include <QMainWindow>
#include <QPainter>
#include <QWheelEvent>

#include "Headers/model3d.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_comboBox_currentIndexChanged(int index);
```

```
    void on_textEdit_selectionChanged();

private:
    double oldXpos;
    double oldYpos;
    int flag;
    double sizeCenter;

    Matr dilatation;
    Matr rotate;

    Ui::MainWindow *ui;
    void paintEvent(QPaintEvent *p) override;
    // Обработчик события прокрутки колеса мыши
    void wheelEvent(QWheelEvent* ) override;
    void mouseMoveEvent(QMouseEvent* );

    void drawAxis(QPainter &painter, double radius);
    void drawDinamic(QPainter &painter, double radius);
};

#endif // MAINWINDOW_H
```

Файл matr.h

```
#ifndef LAB1_MATR_H
#define LAB1_MATR_H

#include <QPointF>

#include <utility>
#include <vector>

using namespace std;

typedef vector<double> type_arr;
typedef vector<type_arr> type_m;
```

```
class Matr{
private:
   type_m m;

public:
   explicit Matr(type_m m): m(std::move(m)){};
   int get_n();
   int get_m();
   friend Matr operator *(Matr m_l, Matr m_r);
   friend type_arr operator *(Matr &m, type_arr &arr);
   friend QPointF operator *(Matr &m, QPointF &p);
   type_arr & operator [](int index);
};

#endif //LAB1_MATR_H
```

Файл matr.cpp

```
//
// Created by ouroboros on 20.09.2020.
//
#include "../Headers/matr.h"


Matr operator*(Matr m_l, Matr m_r) {
   type_m result;
   for (int i = 0; i < m_l.get_n(); ++i) {
      result.push_back({});
      for (int j = 0; j < m_r.get_m(); ++j) {
         double i_j_el = 0;
         for (int k = 0; k < m_r.get_n(); ++k) {
            i_j_el += m_l[i][k] * m_r[k][j];
         }
         result[i].push_back(i_j_el);
      }
   }
   return Matr(result);
}
```

```
type_arr operator*(Matr &m, type_arr &arr) {
  type_arr result;

  for (int i = 0; i < m.get_n(); ++i) {
     double i_j_el = 0;
     for (int j = 0; j < arr.size(); ++j)
        i_j_el += m[i][j] * arr[j];

     result.push_back(i_j_el);
  }

  return result;
}

type_arr &Matr::operator[](int index) {
  return m[index];
}

int Matr::get_n() {
  return m.size();
}

int Matr::get_m() {
  return m[0].size();
}

QPointF operator*(Matr &m, QPointF &p) {
  type_arr mult_arr({p.x(), p.y()});
  for (int i = 2; i < m.get_n(); ++i)
     mult_arr.push_back(0);
  type_arr tranfrom_arr = m*mult_arr;
  return {tranfrom_arr[0], tranfrom_arr[1]};
}
```

Файл afin3d.h

```
//
```

```
// Created by ouroboros on 20.09.2020.
//

#ifndef LAB1_AFIN3D_H
#define LAB1_AFIN3D_H

#include "matr.h"
#include "math.h"

class RotateMX : public Matr {
public:
  explicit RotateMX(double alfa) :
        Matr({
                {1, 0,        0,         0},
                {0, cos(alfa), -sin(alfa), 0},
                {0, sin(alfa), cos(alfa),  0},
                {0, 0,        0,         1}
            }) {};
};

class RotateMY : public Matr {
public:
  explicit RotateMY(double alfa) :
        Matr({
                {cos(alfa), 0, -sin(alfa), 0},
                {0,        1, 0,         0},
                {sin(alfa), 0, cos(alfa),  0},
                {0,        0, 0,         1}
            }) {};
};

class RotateMZ : public Matr {
public:
  explicit RotateMZ(double alfa) :
        Matr({
                {cos(alfa),  sin(alfa), 0, 0},
```

```
                {-sin(alfa), cos(alfa), 0, 0},
                {0,          0,         1, 0},
                {0,          0,         0, 1}
            }) {};
};

class DilatationM : public Matr {
public:
  explicit DilatationM(double kx, double ky, double kz) :
        Matr({
                {kx, 0,  0,  0},
                {0,  ky, 0,  0},
                {0,  0,  kz, 0},
                {0,  0,  0,  1}
            }) {};
};

class TranslateM : public Matr {
public:
  explicit TranslateM(double x, double y, double z) :
        Matr({
                {1, 0, 0, x},
                {0, 1, 0, y},
                {0, 0, 1, z},
                {0, 0, 0, 1}
            }) {};
};

class CabinetProec : public Matr {
public:
  explicit CabinetProec(double l, double alfa) :
        Matr({
                {1,              0,              0, 0},
                {0,              1,              0, 0},
                {l * cos(alfa), l * sin(alfa), 0, 0},
                {0,              0,              0, 1}
```

```
            }) {};
};


#endif //LAB1_AFIN3D_H
```

Файл model3d.h

```
//
// Created by ouroboros on 02.10.2020.
//

#ifndef LAB1_MODEL3D_H
#define LAB1_MODEL3D_H

#include <QPainter>
#include <iostream>
#include "afin3d.h"


#define LEN_X 70*k
#define LEN_Y LEN_X/12.
#define LEN_Z LEN_X

#define RAD 3.14/180.

class Point3D {
public:
  Point3D(double x, double y, double z) : point({x, y, z, 1}) {};

  Point3D(type_arr vec) : point(vec) {};
  type_arr point;

  double x();

  double y();

  double z();
```

```
};

class Model {
private:
  vector<Point3D> vectorPoints;
public:
  explicit Model(vector<Point3D> vector) :
vectorPoints(std::move(vector)) {};

  friend Model operator*(Matr &, Model &);

  void draw(QPainter &painter);
};

class Object {
protected:
  vector<Model> vectorModels;
public:
  explicit Object(vector<Model> vector) :
vectorModels(std::move(vector)) {};

  friend Object operator*(Matr, Object &);

  void draw(QPainter &painter);
  void drawFront(QPainter &painter);
  void drawPorf(QPainter &painter);
  void drawUp(QPainter &painter);
};

class Tumba : public Object {
public:
  Tumba(double k, double sizeH) :
        Object({}) {
     Model inSquare({Point3D(-LEN_X, -LEN_Y, LEN_Z),
                     Point3D(LEN_X, -LEN_Y, LEN_Z),
                     Point3D(LEN_X, LEN_Y, LEN_Z),
```

```
                Point3D(-LEN_X, LEN_Y, LEN_Z)});
    Model centerSquare({Point3D(-LEN_X, -sizeH*k, LEN_Z),
                Point3D(LEN_X, -sizeH*k, LEN_Z),
                Point3D(LEN_X, sizeH*k, LEN_Z),
                Point3D(-LEN_X, sizeH*k, LEN_Z)});
    Matr rt_y_90 = RotateMY(90.*RAD);
    Matr t_up = TranslateM(0, -20*LEN_Y, 0);
    Matr t_down = TranslateM(0, 20*LEN_Y, 0);
    for (int i = 0; i < 4; ++i) {
      vectorModels.push_back(centerSquare);
      vectorModels.push_back(t_up*inSquare);
      vectorModels.push_back(t_down*inSquare);
      inSquare = rt_y_90*inSquare;
      centerSquare = rt_y_90*centerSquare;
    }

    Model outSquareL({Point3D(-LEN_X-2, -21*LEN_Y, LEN_Z),
                Point3D(-(LEN_X+LEN_Y*2), -21*LEN_Y,
LEN_Z),
                Point3D(-(LEN_X+LEN_Y*2), 21*LEN_Y,
LEN_Z),
                Point3D(-LEN_X-2, 21*LEN_Y, LEN_Z)});
    Matr t_back = TranslateM(0, 0, -2*LEN_Z);
    Matr t_rigth = TranslateM(2*(LEN_X+LEN_Y), 0, 0);
    for (int i = 0; i < 2; ++i) {
      vectorModels.push_back(outSquareL);
      vectorModels.push_back(t_rigth*outSquareL);
      outSquareL = t_back*outSquareL;
    }

    Model outSquareS({Point3D(-LEN_X-2, 21*LEN_Y, -LEN_Z),
                Point3D(-(LEN_X+LEN_Y*2), 21*LEN_Y, -
LEN_Z),
                Point3D(-(LEN_X+LEN_Y*2), 21*LEN_Y,
LEN_Z),
                Point3D(-LEN_X-2, 21*LEN_Y, LEN_Z)});
```

```
        Matr t_up_s = TranslateM(0, -42*LEN_Y, 0);
        for (int i = 0; i < 2; ++i) {
            vectorModels.push_back(outSquareS);
            vectorModels.push_back(t_up_s*outSquareS);
            outSquareS = t_rigth*outSquareS;
        }




    };
};

#endif //LAB1_MODEL3D_H
```

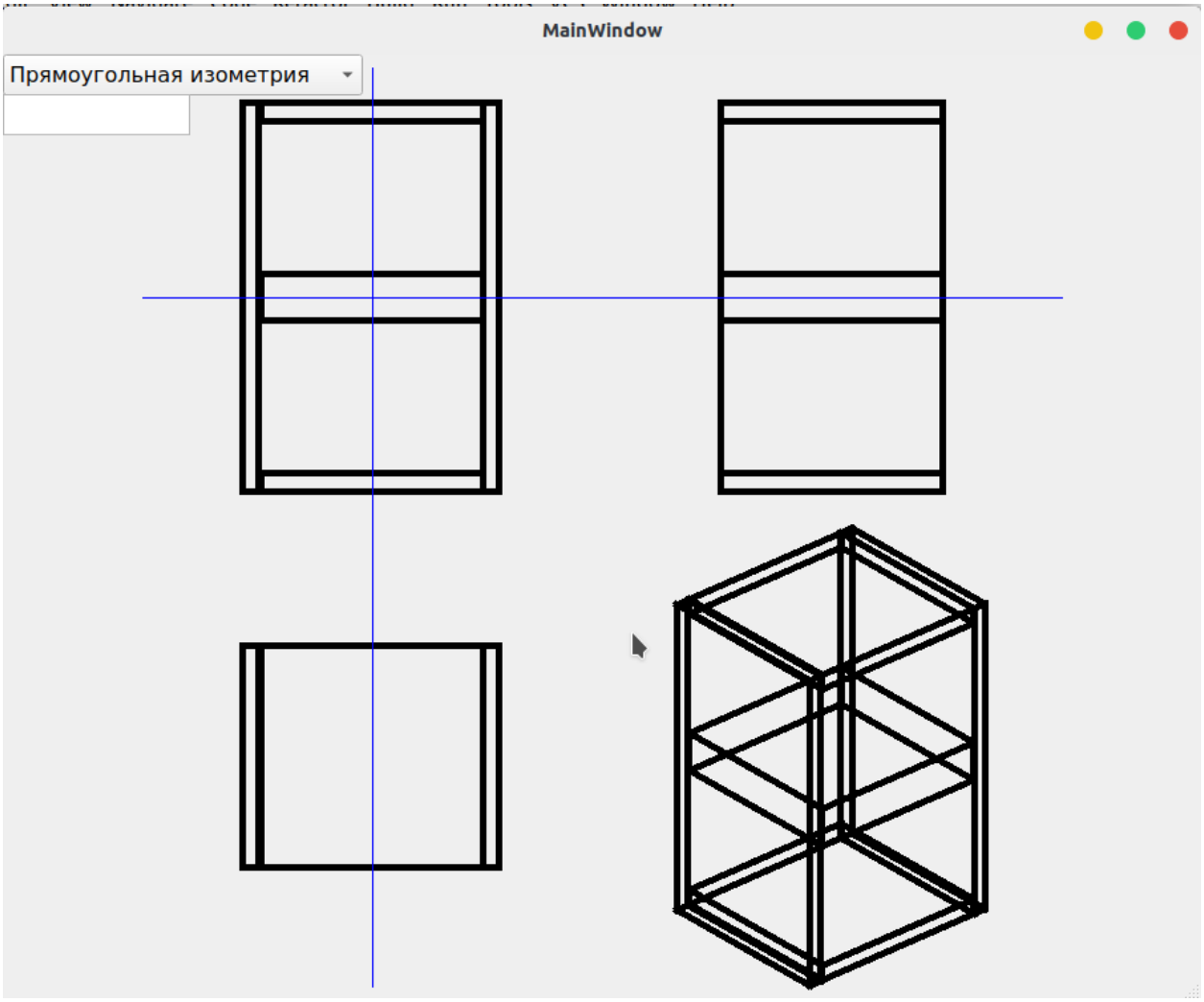
Файл model3d.cpp

```
//
// Created by ouroboros on 02.10.2020.
//
#include "../Headers/model3d.h"

Model operator*(Matr &matr, Model &model) {
    vector<Point3D> newVec;
    for (auto &vec: model.vectorPoints)
        newVec.emplace_back(matr * vec.point);
    return Model(newVec);
}

void Model::draw(QPainter &painter) {
    size_t size = vectorPoints.size() - 1;
    for (int i = 0; i < size; ++i)
        painter.drawLine(QPointF(vectorPoints[i].x(),
vectorPoints[i].y()),
                         QPointF(vectorPoints[i + 1].x(), vectorPoints[i +
1].y()));
    painter.drawLine(QPointF(vectorPoints[size].x(),
vectorPoints[size].y()),
```

```
                QPointF(vectorPoints[0].x(), vectorPoints[0].y()));
}

Object operator*(Matr matr, Object &object) {
  vector<Model> newVec;
  for (auto &model: object.vectorModels)
      newVec.push_back(matr * model);
  return Object(newVec);
}

void Object::draw(QPainter &painter) {
  for (auto &model: vectorModels)
      model.draw(painter);
}

void Object::drawFront(QPainter &painter) {
  draw(painter);
}

void Object::drawPorf(QPainter &painter) {
  Matr rt = RotateMY(90*RAD);
  (rt*(*this)).draw(painter);
}

void Object::drawUp(QPainter &painter) {
  Matr rt = RotateMX(90*RAD);
  (rt*(*this)).draw(painter);
}

double Point3D::x() {
  return point[0];
}

double Point3D::y() {
  return point[1];
```

```
}

double Point3D::z() {
  return point[2];
}
```

6)Результат работы программы (снимки экрана).



Тут изменен размер средней перекладины на 40 а сама проекция чуть уменьшена и повернута