Московский технологический университет
(МИРЭА/МГУПИ)

Физико-технологический институт

Кафедра аппаратного, программного и математического обеспечения вычислительных систем

Курсовая работа по объектно-ориентированному программированию

Проекция вращающихся 3D объектов на плоскость OXY

Группа: ТМБО-01-15
Студент: И. И. Аметов
Преподаватель: А. А. Кожевников

Москва, 2016

Содержание

Введение **2**

Постановка задачи **3**

Алгоритм программы **5**

Текст программы **7**

Литература **17**

Введение

Большую часть информации человек получает посредством зрения. По этой причине развитие разнообразных форм визуальной информации никогда не будет останавливаться. Примерами могут служить даже наскальные рисунки из времён первобытно-общинного строя. С развитием науки и техники также улучшались инструменты визуализации информации: изобретение печати, развитие живописи, возникновение театра, а в последствии и кино. С появлением электронно-вычислительных машин возникла возможность их применение в сфере графики. Очень быстро начала развиваться компьютерная графика. В наши дни компьютерная графика служит мощным инструментом как в сфере развлечений (компьютерные игры, спецэффекты и прочее), так и в научной сфере (визуализация поведения разнообразных реальных моделей, по разработанным математическим моделям).

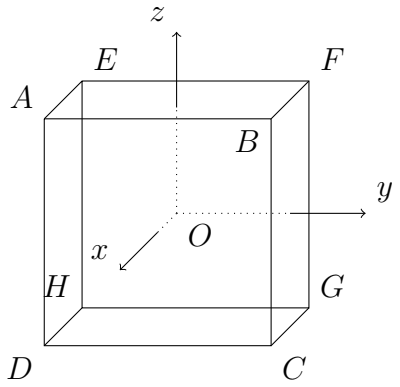
В качестве задачи курсовой работы была выбрана проблема моделирования поворота трёхмерных объектов вокруг осей координат с последующей проекцией полученного преобразования на плоскость OXY и отображением этой проекции на экран компьютера.

Такая постановка задачи позволяет в достаточно полной мере отразить полученные навыки работы с виджетами фреймворка Qt в объектно-ориентированном программировании. Помимо работы с виджетами в данной курсовой работе используется библиотека QPainter. Эта библиотека используется для рисования на некоторой области виджета. Дополнительно была рассмотрена возможность многопоточного программирования.

Постановка задачи

Требуется реализовать некоторую виртуальную модель какой-либо геометрической фигуры. Виртуальная модель должна описывать фигуру в трёхмерном пространстве, дополнительно на саму модель налагается задача преобразования координат в зависимости от углов поворота. Описание фигуры выполнено в виде двух массивов K и L. Массив K — двумерный массив размерности $n \times 3$, где n — количество точек, описывающие фигуру, а 3 — координаты X, Y и Z в декартовой системе координат. Массив L — плоский одномерный массив размерности $1 \times 2k$, где k — количество связей между какими-либо точками фигуры.

В качестве примера возьмём куб с вершинами A, B, C, D, E, F, G и H.



Тогда, если допустить что начало декартовой системы координат находится в центре масс куба, то координаты куба можно задать следующим образом: $A(t, -t, t)$, $B(t, t, t)$, $C(t, t, -t)$, $D(t, -t, -t)$, $E(-t, -t, t)$, $F(-t, t, t)$, $G(-t, t, -t)$ и $H(-t, -t, -t)$. В виде массива данные координаты можно записать так:

$$K = \begin{pmatrix} t & -t & t \\ t & t & t \\ t & t & -t \\ t & -t & -t \\ -t & -t & t \\ -t & t & t \\ -t & t & -t \\ -t & -t & -t \end{pmatrix}$$

В этой матрице строка 0 — это точка A, 1 — точка B, ..., 7 — точка H (отсчёт идёт с нуля).

Связи между точками образуются следующими парами: (A, B), (A, E), (A, D), (B, F), (B, C), (C, G), (C, D), (D, H), (E, F), (E, H), (F, G) и (G, H). Причём пары (A, B) и (B, A) — это одна и та же пара. В виде матрицы-строки связи можно записать так:

$L = \big(0, 1, 0, 4, 0, 3, 1, 5, 1, 2, 2, 6, 2, 3, 3, 7, 4, 5, 4, 7, 5, 6, 6, 7\big)$

Если отсчитывать элементы индексов с нуля то образуются связи $(L(0), L(1)) = (A, B)$, $(L(2), L(3)) = (A, E)$, ..., $(L(6), L(7)) = (G, H)$. Поскольку в массиве L перечислены пары, то количество элементов в L может быть только чётным.

Помимо этого объект трёхмерной модели должен содержать сведения об углах поворота исходной системы координат. В данном случае этих углов три штуки: угол поворота

α вокруг оси OX, угол поворота β вокруг оси OY и угол поворота γ вокруг оси OZ. Положительным направлением поворота считается: для угла α — поворот оси OY к оси OZ, для угла β — поворот оси OZ к OX и для угла γ — поворот оси OX к оси OY.

Новая система координат (полученная после поворота старой системы координат) в старой системе координат будет выражаться так:

$$x' = (x\cos\gamma - y\sin\gamma)\cos\beta + z\sin\beta$$

$$y' = (y\cos\gamma + x\sin\gamma)\cos\alpha - z\sin\alpha$$

$$z' = (z\cos\alpha + y\sin\alpha)\cos\beta - x\sin\beta$$

где x', y' и z' — это новые координаты точки в старой системе координат полученные после поворота, x, y и z — это начальные координаты точки в старой системе координат.

Алгоритм программы

Всего в программе определено 4 новых класса:

- Object3D — реализация модели трёхмерного объекта
- widget — главный виджет графического интерфейса пользователя
- Autopilot — объект, отвечающий за режим автоматического вращения системы координат
- Sleeper — объект, предназначенный для реализации паузы в работе программы

Object3D

В этом объекте реализуется хранение, обработка и передача информации о трёхмерном объекте. Для хранения информации о точках фигуры используется двумерный массив objectDots с динамическим выделением памяти. Хранение информации о связях между точками осуществляется в одномерном массиве dotsLinks также с динамическим выделением памяти. Углы поворота вокруг осей OX, OY и OZ хранятся соответственно в переменных angleX, angleY, angleZ.

При создании объекта Object3D ему передаются массивы, содержащие данные о координатах точек и о связях между точками. Далее, в процессе работы объекту передаются сведения о новых углах поворотов с помощью методов setXAngle, setYAngle и setZAngle. Кроме того, реализован метод, предназначенный для смены объекта “на лету” — setObject. Всего в программе описаны три предустановленных объекта: куб, пирамида и звезда.

widget

Основной виджет графического интерфейса пользователя. Рабочая графическая часть программы состоит из двух окон: “Панели управления” и канвы отображения объекта.

Панель управления объектом состоит из трёх слайдеров отвечающих за задание углов поворота:

- aroundXSlider
- aroundYSlider
- aroundZSlider

Также на панели управления расположены клавиша включения автопилота и выпадающий список, позволяющий выбрать фигуру трёхмерного объекта.

Autopilot

Объект, реализующий автоматическую смену углов поворота по значениям угловых скоростей, сгенерированных датчиком случайных чисел. Основные элементы этого объекта:

- startXAngle, startYAngle и startZAngle — начальные значения углов поворота на момент включение автопилота.
- vX, vY и vZ — угловые скорости поворотов.
- Метод generateSpeed — генерация случайной угловой скорости.
- currentSeconds — переменная, хранящая время прошедшее с начала запуска автопилота.
- process — основная рабочая процедура автопилота

Работа процедуры process заключается в следующем: до тех пор, пока булево значение enabled установлено в истинное значение, повторять: приостановить работу на 100 миллисекунд, прирастить значение текущего времение currentSeconds на 0.1 секунду, породить сигнал newAngles содержащий значения новых углов поворота по формуле:

$$\text{новый угол} = \text{угловая скорость} \times \text{текущее время} + \text{начальное значение угла}$$

Если будет включена пауза в работе автопилота (происходит при смене фигуры при включённом автопилоте), то работа приостанавливается на 100 миллисекунд, после чего повторяется проверка на снятие автопилота с паузы. Если же пользователь нажал на клавишу останова автопилота, то переменная enabled устанавливается в ложное значение и цикл while прекращает работу с последующей генерацией сигнала finished().

Sleeper

Весь функционал этого объекта заключён в паузе, формируемой с помощью класса QThread. В теле класса Sleeper осуществляется вызов `QThread::msleep(msecs)`, где `msecs` — время в миллисекундах.

Текст программы

Listing 1: widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QPainter>
#include <math.h>
#include <QPushButton>
#include <QVBoxLayout>
#include <QGroupBox>
#include <QSlider>
#include <QLabel>
#include "Object3D.h"
#include "Autopilot.h"
#include <QComboBox>

// Основной класс
class Widget : public QWidget
{
    Q_OBJECT
private:
    // Слайдер поворота вокруг оси Z
    QSlider * aroundZSlider;
    QLabel * zAngleLabel;

    // Слайдер поворота вокруг оси X
    QSlider * aroundXSlider;
    QLabel * xAngleLabel;

    // Слайдер поворота вокруг оси Y
    QSlider * aroundYSlider;
    QLabel * yAngleLabel;

    // Трёхмерный объект
    Object3D * object3d;

    // Клавиша запуска автопилота
    QPushButton * autoPilotButton;

    // Объект автопилота
    Autopilot * myAutopilot;

    // Тред для автопилота
    QThread * myThread;

    // Отображение включённости автопилота
    bool autopilotIsEnabled;

    // ComboBox с трёхмерными фигурами
    QComboBox * shapeComboBox;

    // Процедура установки нового объекта
    void setNewObject();
public:
    Widget(QWidget *parent = 0);
    void paintEvent(QPaintEvent *);
    ~Widget();
public  slots:

    // Реакция на изменение слайдера с углом Z
    void sliderZchanged(int value);

    // Реакция на изменение слайдера с углом X
    void sliderXchanged(int value);

    // Реакция на изменение слайдера с углом Y
    void sliderYchanged(int value);

    // Реакция на нажатие клавиши автопилота
    void buttonAutopilotPressed();

```

```
    // Реакция на сигнал с новыми углами поворотов от автопилота
    void autopilotDatas(qreal XAngle,
                        qreal YAngle,
                        qreal ZAngle);

    // Реакция на выбор фигуры из ComboBoxa—
    void selectShape();

    // Реакция на паузу автопилота
    void autopilotPaused();
};

#endif // WIDGET_H
```

Listing 2: widget.cpp

```
#include "widget.h"

// Конструктор главного объекта
Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    this—>setStyleSheet("background: black;");
    autopilotIsEnabled = false;
    QGroupBox * controlPanel = new QGroupBox(tr("Control panel"));
    QVBoxLayout * controlPanelLayout = new QVBoxLayout;

    // Слайдер для вращения вокруг оси Z
    aroundZSlider = new QSlider(Qt::Horizontal, controlPanel);
    aroundZSlider—>setMinimum(0);
    aroundZSlider—>setMaximum(360);
    aroundZSlider—>setValue(0);
    zAngleLabel = new QLabel("Z angle: " +
                             QString::number(aroundZSlider—>value()),
                             controlPanel);
    connect(aroundZSlider, SIGNAL(valueChanged(int)), this,
            SLOT(sliderZchanged(int)));

    // Слайдер для вращения вокруг оси X
    aroundXSlider = new QSlider(Qt::Horizontal, controlPanel);
    aroundXSlider—>setMinimum(0);
    aroundXSlider—>setMaximum(360);
    aroundXSlider—>setValue(0);
    xAngleLabel = new QLabel("X angle: " + QString::number(aroundXSlider—>value()),
                             controlPanel);
    connect(aroundXSlider, SIGNAL(valueChanged(int)), this,
            SLOT(sliderXchanged(int)));

    // Слайдер для вращения вокруг оси Y
    aroundYSlider = new QSlider(Qt::Horizontal, controlPanel);
    aroundYSlider—>setMinimum(0);
    aroundYSlider—>setMaximum(360);
    aroundYSlider—>setValue(0);
    yAngleLabel = new QLabel("Y angle: " + QString::number(aroundYSlider—>value()),
                             controlPanel);
    connect(aroundYSlider, SIGNAL(valueChanged(int)), this,
            SLOT(sliderYchanged(int)));

    // Клавиша автопилота
    autoPilotButton = new QPushButton("Enable autopilot", controlPanel);
    connect(autoPilotButton, SIGNAL(clicked()), SLOT(buttonAutopilotPressed()));

    // Задание выпадающего списка с фигурами
    shapeComboBox = new QComboBox;
    shapeComboBox—>addItem("Cube");
    shapeComboBox—>addItem("Pyramid");
    shapeComboBox—>addItem("Star");
    connect(shapeComboBox, SIGNAL(currentIndexChanged(int)), SLOT(selectShape()));

    // Задание фигуры
    qreal cubeDots[8][3] = {{50, —50, —50}, {50, —50, 50},
                            {50, 50, 50}, {50, 50, —50},
                            {—50, —50, —50}, {—50, —50, 50},
                            {—50, 50, 50}, {—50, 50, —50}};
    int cubeLinks[] = {0, 1, 0, 4, 0, 3, 1, 2, 1, 5, 2, 6, 2, 3, 3, 7, 5, 6, 5, 4,
```

```
                      6, 7, 4, 7};
    object3d = new Object3D(&cubeDots[0][0], 8, &cubeLinks[0], 24);

    // Добавление виджетов на слой компоновки
    controlPanelLayout->addWidget(zAngleLabel);
    controlPanelLayout->addWidget(aroundZSlider);

    controlPanelLayout->addWidget(xAngleLabel);
    controlPanelLayout->addWidget(aroundXSlider);

    controlPanelLayout->addWidget(yAngleLabel);
    controlPanelLayout->addWidget(aroundYSlider);

    controlPanelLayout->addWidget(autoPilotButton);

    controlPanelLayout->addWidget(shapeComboBox);
    controlPanel->setLayout(controlPanelLayout);
    controlPanel->show();
}

void Widget::buttonAutopilotPressed(){
    // Реакция на нажатие клавиши автопилота
    if (autopilotIsEnabled) {
        // Если автопилот уже был запущен
        myAutopilot->stop();
        aroundXSlider->setEnabled(true);
        aroundYSlider->setEnabled(true);
        aroundZSlider->setEnabled(true);
        autoPilotButton->setText("Enable autopilot");
        autopilotIsEnabled = false;
    }
    else {
        // Если автопилот не был запущен
        myThread = new QThread;
        myAutopilot = new Autopilot(object3d->getAngleX(),
                                    object3d->getAngleY(),
                                    object3d->getAngleZ());
        myAutopilot->moveToThread(myThread);
        connect(myAutopilot, SIGNAL(newAngles(qreal,qreal,qreal)),
                SLOT(autopilotDatas(qreal,qreal,qreal)));
        connect(myAutopilot, SIGNAL(pausedSignal()),
                SLOT(autopilotPaused()));
        connect(myThread, SIGNAL(started()), myAutopilot,
                SLOT(start()));
        connect(myAutopilot, SIGNAL(finished()), myThread,
                SLOT(quit()));
        connect(myAutopilot, SIGNAL(finished()), myAutopilot,
                SLOT(deleteLater()));
        connect(myThread, SIGNAL(finished()), myThread,
                SLOT(deleteLater()));
        aroundXSlider->setEnabled(false);
        aroundYSlider->setEnabled(false);
        aroundZSlider->setEnabled(false);

        // Смена раскраски слайдеров
        QPalette disablePalette = aroundXSlider->palette();
        disablePalette.setColor(QPalette::Disabled,
                                QPalette::Light,
                                QColor(128, 128, 128));
        aroundXSlider->setPalette(disablePalette);
        aroundYSlider->setPalette(disablePalette);
        aroundZSlider->setPalette(disablePalette);
        autoPilotButton->setText("Disable autopilot");
        myThread->start();
        autopilotIsEnabled=true;
    }
}

void Widget::paintEvent(QPaintEvent *){
    // Процедура отрисовки окна с изображением
    this->setWindowTitle("Width: " +
                         QString::number(this->width()) +
                         " Height: " +
                         QString::number(this->height()));
    QPainter painter(this);
    painter.setPen(Qt::blue);
    qreal windowHalfHeight = this->height()/2,
```

```
            windowHalfWidth = this->width()/2;
    int firstDot, secondDot;

    // Отрисовка трёхмерного объекта
    for (int i = 0; i < object3d->getLinksCount(); i++){
        firstDot = object3d->getLinkFirstDot(i);
        secondDot = object3d->getLinkSecondDot(i);
        painter.drawLine(QPoint(object3d->getY(firstDot) +
                                windowHalfWidth,
                                object3d->getX(firstDot) +
                                windowHalfHeight),
                         QPoint(object3d->getY(secondDot) +
                                windowHalfWidth,
                                object3d->getX(secondDot) +
                                windowHalfHeight));
    }
}

Widget::~Widget()
{
    // Деструктор виджета
}

void Widget::sliderZchanged(int value){
    // Реакция на изменение слайдера с углом Z
    zAngleLabel->setText("Z angle: " +
                         QString::number(value));
    object3d->setZAngle(value*3.14/180);
    update();
}

void Widget::sliderXchanged(int value){
    // Реакция на изменение слайдера с углом X
    xAngleLabel->setText("X angle: " +
                         QString::number(value));
    object3d->setXAngle(value*3.14/180);
    update();
}

void Widget::sliderYchanged(int value){
    // Реакция на изменение слайдера с углом Y
    yAngleLabel->setText("Y angle: " +
                         QString::number(value));
    object3d->setYAngle(value*3.14/180);
    update();
}

void Widget::autopilotDatas(qreal XAngle, qreal YAngle, qreal ZAngle){
    // Реакция на сигнал с новыми углами поворотов от автопилота
    object3d->setXAngle(XAngle);
    object3d->setYAngle(YAngle);
    object3d->setZAngle(ZAngle);
    aroundXSlider->setValue((((int)(XAngle * 180 / 3.14) % 360)+360)%360);
    aroundYSlider->setValue((((int)(YAngle * 180 / 3.14) % 360)+360)%360);
    aroundZSlider->setValue((((int)(ZAngle * 180 / 3.14) % 360)+360)%360);
    update();
}

void Widget::selectShape(){
    // Реакция на выбор фигуры из ComboBoxa
    if (!autopilotIsEnabled){
        setNewObject();
    }
    else myAutopilot->pauseOnOff();
}

void Widget::setNewObject(){
    // Процедура установки нового объекта в myObject
    switch (shapeComboBox->currentIndex()) {
    case 0:
    {
        // Реализация трёхмерного объекта "Куб"
        qreal cubeDots[8][3] = {{50, -50, -50}, {50, -50, 50}, {50, 50, 50},
                                {50, 50, -50}, {-50, -50, -50}, {-50, -50, 50},
                                {-50, 50, 50}, {-50, 50, -50}};
        int cubeLinks[] = {0, 1, 0, 4, 0, 3, 1, 2, 1, 5, 2, 6,
                           2, 3, 3, 7, 5, 6, 5, 4, 6, 7, 4, 7};
```

```
        object3d->setObject(&cubeDots[0][0], 8, &cubeLinks[0], 24);
        update();
    }
        break;
    case 1:
    {
        // Реализация трёхмерного объекта "Пирамида"
        qreal tetraDots[4][3] = {{0, 0, 100}, {0, -100, 0}, {-50, 50, 0},
                                 {50, 50, 0}};

        int tetraLinks[] = {0, 1, 0, 2, 0, 3, 1, 0, 1, 2, 1, 3, 2, 0, 2, 1,
                            2, 3, 3, 0, 3, 1, 3, 2};
        object3d->setObject(&tetraDots[0][0], 4, &tetraLinks[0], 24);
        update();
    }
        break;
    case 2:
    {
        qreal starDots[12][3] = {{200, 0, 0}, {60, 190, 0}, {-160, 116, 0},
                                 {-160, -116, 0}, {60, -190, 0}, {60, 44, 0},
                                 {-22, 72, 0}, {-76, 0, 0}, {-22, -72, 0},
                                 {60, -44, 0}, {0, 0, 25}, {0, 0, -25}};
        int starLinks[] = {0, 5, 0, 10, 0, 11, 0, 9, 1, 5, 1, 6, 1, 11,
                           1, 10, 2, 6, 2, 7, 2, 11, 2, 10, 3, 7, 3, 8,
                           3, 11, 3, 10, 4, 8, 4, 9, 4, 11, 4, 10, 5, 11,
                           5, 10, 6, 11, 6, 10, 7, 11, 7, 10, 8, 11, 8, 10,
                           9, 11, 9, 10};
        object3d->setObject(&starDots[0][0], 12, &starLinks[0], 60);
        update();

    }
        break;
    default:
        break;
    }
}

void Widget::autopilotPaused(){
    // Смена объекта при паузе автопилота
    setNewObject();
    myAutopilot->pauseOnOff();
}
```

Listing 3: Object3D.h

```
#ifndef OBJECT3D_H
#define OBJECT3D_H
#include <QtGlobal>
#include <cmath>

const int three = 3, two = 2, zero = 0;

// Класс трёхмерного объекта
class Object3D{
private:
    qreal ** objectDots; // Координаты (x, y, z)
    int dotsCount; // Количество точек
    int *dotsLinks; // Взаимосвязи точек (1 соединяется с 2 и тд..)
    int linksCount; // Количество связей
    qreal angleZ; // Поворот вокруг оси Z радианы()
    qreal angleX; // Поворот вокруг оси X радианы()
    qreal angleY; // Поворот вокруг оси Y радианы()
    qreal maxD; // Дистанция от начала координат то самой удалённой точки

    // Расстояние от центра до самой удалённой точки Д3 объекта
    qreal maxDistance();
    // Сброс данных при( смене фигуры), углы сохраняются
    void resetObject();

public:
    // Конструктор, dots — массив точек, r — количество точек
    // links — связи между точками, linksCount — количество связей
    Object3D(qreal *dots, int r, int * links, int totalLinksCount);

    // Возврат координаты X какойлибо— точки. Отсчёт от 0.
```

```
    qreal getX(int dotNumber);

    // Возврат координаты Y какойлибо— точки. Отсчёт от 0.
    qreal getY(int dotNumber);

    // Возврат координаты Z какойлибо— точки. Отсчёт от 0.
    qreal getZ(int dotNumber);

    // Вернуть количество связей. Отсчёт от 1.
    int getLinksCount();

    // Вернуть количество точек. Отсчёт от 1.
    int getDotsCount();

    // Вернуть номер первой точки из связи
    int getLinkFirstDot(int linkNumber);

    // Вернуть номер второй точки из связи
    int getLinkSecondDot(int linkNumber);

    // Установить поворот вокруг оси Z в( радианах)
    void setZAngle(qreal angle);

    // Установить поворот вокруг оси X в( радианах)
    void setXAngle(qreal angle);

    // Установить поворот вокруг оси Y в( радианах)
    void setYAngle(qreal angle);

    // Возращение углов поворота вокруг осей
    qreal getAngleZ();
    qreal getAngleX();
    qreal getAngleY();

    // Возвращает максимальную дистанцию
    qreal getMaxDistance();

    // Установка нового объекта
    void setObject(qreal *, int, int *, int);
};

#endif // OBJECT3D_H
```

Listing 4: Object3D.cpp

```
#include "Object3D.h"

Object3D::Object3D(qreal *dots, int r, int * links,
                   int totalLinksCount){
    // Конструктор объекта
    // Загружаем координаты точек объекта
    objectDots = new qreal * [r];
    dotsCount = r;
    for (int j = zero; j < r; j++){
        objectDots[j] = new qreal [three];
        for (int i = zero; i < three; i++)
            objectDots[j][i] = dots[j*three + i];
    }

    // Загружаем связи точек
    dotsLinks = new int [totalLinksCount];
    linksCount = totalLinksCount / 2;
    for (int j = zero; j < totalLinksCount; j++){
        dotsLinks[j] = links[j];
    }

    // Выставление начальных углов
    setXAngle(0);
    setYAngle(0);
    setZAngle(0);

    // Дистанция от начала координат до дальней точки
    maxD = maxDistance();
}

```

```
qreal Object3D::getX(int dotNumber){
    // Возврат координаты X какойлибо— точки
    if (dotNumber <= dotsCount — 1)
        return (objectDots[dotNumber][0] * cos (angleZ) —
                objectDots[dotNumber][1] * sin(angleZ)) * cos (angleY) +
                objectDots[dotNumber][2] * sin(angleY);
    else
        return zero;
}

qreal Object3D::getY(int dotNumber){
    // Возврат координаты Y какойлибо— точки
    if (dotNumber <= dotsCount — 1)
        return (objectDots[dotNumber][1] * cos(angleZ) +
                objectDots[dotNumber][0] * sin(angleZ)) * cos(angleX) —
                objectDots[dotNumber][2] * sin(angleX);
    else
        return zero;
}

qreal Object3D::getZ(int dotNumber){
    // Возврат координаты Z какойлибо— точки
    if (dotNumber <= dotsCount — 1)
        return (objectDots[dotNumber][2] * cos(angleX) +
                objectDots[dotNumber][1] * sin(angleX)) * cos(angleY) —
                objectDots[dotNumber][0] * sin(angleY);
    else
        return zero;
}

int Object3D::getLinksCount(){
    // Возврат количества связей
    return linksCount;
}

int Object3D::getDotsCount(){
    // Возврат количества точек
    return dotsCount;
}

int Object3D::getLinkFirstDot(int linkNumber){
    // Возврат номера первой точки из связи
    if (linkNumber <= linksCount — 1 && linkNumber >= 0)
        return dotsLinks[linkNumber * 2];
    else
        return —1;
}

int Object3D::getLinkSecondDot(int linkNumber){
    // Возврат номера второй точки из связи
    if (linkNumber <= linksCount — 1 && linkNumber >= 0)
        return dotsLinks[linkNumber * 2 + 1];
    else
        return —1;
}

void Object3D::setZAngle(qreal angle){
    // Установка угла поворота вокруг оси Z
    angleZ = angle;
}

void Object3D::setXAngle(qreal angle){
    // Установка угла поворота вокруг оси X
    angleX = angle;
}

void Object3D::setYAngle(qreal angle){
    // Установка угла поворота вокруг оси Y
    angleY = angle;
}

qreal Object3D::maxDistance()
{
    // Вычисление дистанции от начала координат до дальней точки
    qreal distance = 0, temp;

    for (int i = 0; i < dotsCount; i++){
```

```
        temp = sqrt(getX(i)*getX(i) + getY(i)*getY(i) + getZ(i)*getZ(i));
        if (temp > distance) distance = temp;
    }
    return distance;
}

qreal Object3D::getMaxDistance()
{
    // Возврат дистанции от начала координат до дальней точки
    return maxD;
}

qreal Object3D::getAngleX(){
    // Возврат угла поворота вокруг оси X
    return angleX;}

qreal Object3D::getAngleY(){
    // Возврат угла поворота вокруг оси Y
    return angleY;}

qreal Object3D::getAngleZ(){
    // Возврат угла поворота вокруг оси Z
    return angleZ;}

void Object3D::resetObject(){
    // Сброс данных объекта
    delete [] objectDots;
    dotsCount = 0;
    delete [] dotsLinks;
    linksCount = 0;
    maxD = 0;
}

void Object3D::setObject(qreal * dots, int r, int * links, int totalLinksCount){
    // Установка нового объекта
    // Загружаем координаты точек объекта
    objectDots = new qreal * [r];
    dotsCount = r;
    for (int j = zero; j < r; j++){
        objectDots[j] = new qreal [three];
        for (int i = zero; i < three; i++)
            objectDots[j][i] = dots[j*three + i];
    }

    // Загружаем связи точек
    dotsLinks = new int [totalLinksCount];
    linksCount = totalLinksCount / 2;
    for (int j = zero; j < totalLinksCount; j++){
        dotsLinks[j] = links[j];
    }
    maxD = maxDistance();
}
```

Listing 5: Autopilot.h

```
#ifndef AUTOPILOT_H
#define AUTOPILOT_H
#include <QObject>
#include <ctime>
#include <QThread>
#include "Sleeper.h"

// Класс автопилота
class Autopilot : public QObject{
    Q_OBJECT
private:
    qreal startXAngle, startYAngle, startZAngle; // Начальные углы по осям
    qreal vX, vY, vZ; // Скорость изменения угла в секунду
    bool enabled; // Запущен ли процесс
    bool paused; // Приостановлен ли процесс
    qreal generateSpeed(); // Генерация угловых скоростей
    qreal currentSeconds; // Текущее время от начала работы
    void process(); // Основной рабочий процесс автопилота
public:
    // Конструктор
```

```
    Autopilot(qreal XAngle, qreal YAngle, qreal ZAngle);

    // Деструктор
    ~Autopilot();

public slots:
    // Слот запуска процесса
    void start();

    // Слот останова процесса
    void stop();

    // Слот паузы вклвыкл/
    void pauseOnOff();
signals:

    // Сигнал с новыми углами
    void newAngles(qreal XAngle, qreal YAngle, qreal ZAngle);

    // Сигнал завершения работы автопилота
    void finished();

    // Сигнал о вставании на паузу
    void pausedSignal();
};
#endif // AUTOPILOT_H
```

Listing 6: Autopilot.cpp

```
#include "Autopilot.h"

Autopilot::Autopilot(qreal XAngle,
                     qreal YAngle,
                     qreal ZAngle){
    // Конструктор автопилота
    std::srand(std::time(0) * 29);
    startXAngle = XAngle;
    startYAngle = YAngle;
    startZAngle = ZAngle;
    vX = generateSpeed();
    vY = generateSpeed();
    vZ = generateSpeed();
    currentSeconds = 0;
    enabled = false;
    paused = false;
}

Autopilot::~Autopilot(){
    // Деструктор автопилота
}

qreal Autopilot::generateSpeed(){
    // Генерация новой скорости
    return std::rand() * 3.14 / RAND_MAX—1.57;
}

void Autopilot::start(){
    // Запуск автопилота
    enabled = true;
    process();
}

void Autopilot::stop(){
    // Останов автопилота
    enabled = false;
}

void Autopilot::pauseOnOff(){
    // Пауза автопилота вклвыкл/
    if (paused) paused = false;
    else {
        paused = true;
        emit pausedSignal();
    }
}
```

```
47
48  void Autopilot::process(){
49      // Основной процесс автопилота
50      while (enabled){
51          if (!paused){
52              Sleeper::msleep(100);
53              currentSeconds += 0.1;
54              emit newAngles(vX*currentSeconds + startXAngle,
55                             vY*currentSeconds + startYAngle,
56                             vZ*currentSeconds + startZAngle);}
57          else Sleeper::msleep(100);
58      }
59      emit finished();
60  }
```

Listing 7: Sleeper.h

```
1   #ifndef SLEEPER_H
2   #define SLEEPER_H
3   #include <QThread>
4
5   // Объект паузы в процессе работы треда
6   class Sleeper : public QThread
7   {
8   public:
9       static void usleep(unsigned long usecs){QThread::usleep(usecs);}
10      static void msleep(unsigned long msecs){QThread::msleep(msecs);}
11      static void sleep(unsigned long secs){QThread::sleep(secs);}
12  };
13  #endif // SLEEPER_H
```

Listing 8: main.cpp

```
1   #include "widget.h"
2   #include "Object3D.h"
3   #include <QApplication>
4
5   int main(int argc, char *argv[])
6   {
7       QApplication a(argc, argv);
8
9       Widget w;
10      w.show();
11      return a.exec();
12  }
```

Литература

В процессе написания курсовой работы использовалась следующая литература:

1. Qt 4.8 Профессиональное программирование на C++. Макс Шлее. “БХВ-Петербург”. 2012.
2. Лекции по аналитической геометрии, пополненные необходимыми сведениями из алгебры. П.С. Александров. Издательство “Наука”. 1968.