МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Компьютерная графика»

Тема: «Визуализация 3D объекта. Морфинг тетраэдр — куб»

Студентка гр. 8383 _____ Максимова А.А.

Студент гр. 8383 _____ Ларин А.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студенты: Максимова А.А., Ларин А.

Группа 8383

Тема работы: Визуализация 3D объекта. Морфинг тетраэдр — куб

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 08.05.2021

Дата сдачи реферата: 30.05.2021

Студентка гр. 8383	_____	Максимова А.А.
Студент гр. 8383	_____	Ларин А.
Преподаватель	_____	Герасимова Т.В.

Задание

1) Разработка демонстрационных примеров
2) Подбор материала по теме для обзора (1 стр), материал д.б. творчески переработан, дополнен примерами вашей реализации. Обязательны ссылки на литературу.
3) Отчет сдается в печатном виде, разработка + отчет в электронном виде.

Имеется куб и тетраэдр - Платоновы тела, написать программу возможных плавных переходов: Тетраэдр -> куб

Основные теоретические положения

Морфинг

Морфинг (англ. *morphing*, трансформация) — технология в компьютерной анимации, визуальный эффект, создающий впечатление плавной трансформации одного объекта в другой. Используется в игровом и телевизионном кино, в телевизионной рекламе. Встречается в трёхмерной и двухмерной (как растровой, так и векторной) графике. Для морфинга используется два объекта, которые являются ключевыми кадрами начала и конца морфинка соответственно, после чего вычисляется их форма в промежутках времени между ключевыми кадрами.
Морфинг также часто используется для создания анимации, когда не стоит задача добиться эффекта превращения одного объекта в другой, а требуется лишь выстроить промежуточные состояния между двумя (и более) ключевыми положениями анимируемого объекта.

Существует несколько видов морфинга.

1. Морфинг интерполяцией. При этом виде морфинга точки одного объекта меняют свое положения между начальным и конечным согласно функции интерполяции от времени. Такой вид морфинга сохраняет топологию объекта. В общем случае конфигурация вершин и ребер двух объектов могут отличаться, по этому может потребоваться подразбиение либо слияние граней. На топологию объекта это не повлияет.
2. Морфинг отсечением. Данный вид морфинга предполагает, что один объект может содержаться в другом, а потому отсекая части от одного ключевого объекта можно получить второй. Данный вид также предполагает выпуклость обеих фигур. Пример на рис1.

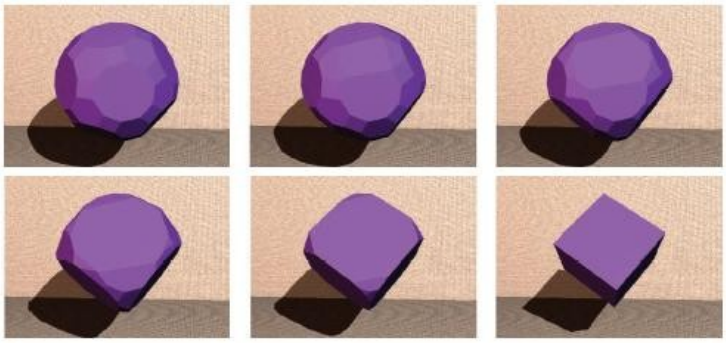

Рисунок 1 - Морфинг отсечением.

3. Морфирование системой частиц. Данный вид морфинга оперирует понятием частиц из которых состоят(или составимы) объекты. Оперрируя двумя множествами частиц можно превращ ать одну конфигурацию частиц пользуясь правилом интерполяции. Данный вид морфинга в основном используется как графический эффект и не зависим от топологии объекта. [4]

Платоновы тела - это выпуклые многогранники, состоящие из одинаковых правильных многоугольников и обладающие пространственной симметрией. Существует пять выпуклых платоновых тел без самопересечений.

- Тетраэдр является треугольной пирамидой при принятии любой из граней за основание. У тетраэдра 4 грани, 4 вершины и 6 рёбер.
- Куб (гексаэдр) - правильный многогранник, каждая грань которого представляет собой квадрат. Частный случай параллелепипеда и призмы. У гексаэдра 6 граней, 12 рёбер, 8 вершин
- Октаэдр - его грани — восемь равносторонних треугольников. У октаэдра 8 граней, 12 рёбер, 6 верши
- Додекаэдр составлен из двенадцати правильных пятиугольников, являющихся его гранями. Каждая вершина додекаэдра является вершиной трёх правильных пятиугольников. Таким образом, додекаэдр имеет 12 граней (пятиугольных), 30 рёбер и 20 вершин (в каждой сходятся 3 ребра).
- Икосаэдр — один из правильных многогранников, гранями которого являются 20 правильных треугольников. 12 вершин и 30 рёбер.

Создание трехмерного объекта

Основная задача, которую необходимо решить при выводе трехмерной графической информации, заключается в том, что объекты, описанные в мировых координатах, необходимо изобразить на плоской области вывода экрана, т.е. требуется преобразовать координаты точки из мировых координат (x,y,z) в оконные координаты (X,Y) ее центральной проекции. Это отображение выполняют в несколько этапов.

Первый этап – *видовое преобразование* – преобразование мировых координат в видовые (видовая матрица);

второй этап – *перспективное преобразование* – преобразование видовых координат в усеченные (матрица проекции).

Для того, чтобы активизировать какую-либо матрицу, надо установить текущий режим матрицы, для чего служит команда :

void glMatrixMode (GLenum mode).

Параметр mode определяет, с каким набором матриц будет выполняться последовательность операций, и может принимать одно из трех значений: GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE. Для определения элементов матрицы используются следующие команды: glLoadMatrix, glLoadIdentity.

Преобразование объектов выполняется при помощи следующих операций над матрицами.

void glRotate[f d]**(GLdouble angle, GLdouble x, GLdouble y, GLdouble z)**

Эта команда рассчитывает матрицу для выполнения вращения вектора против часовой стрелки на угол, определяемый параметром angle, осуществляемого относительно точки (x,y,z). После выполнения этой команды все объекты изображаются повернутыми.

void glTranslate[f d]**(GLdouble x, GLdouble y, GLdouble z);**

При помощи этой команды осуществляется перенос объекта на расстояние x по оси X, на расстояние y по оси У и на z по оси Z.

Проекции.

Несоответствие между пространственными объектами и плоским изображением устраняется путем введения проекций, которые отображают объекты на двумерной проекционной картинной плоскости. Очень важное значение имеет расстояние между наблюдателем и объектом, поскольку "эффект перспективы" обратно пропорционален этому расстоянию. Таким образом, вид проекции зависит от расстояния между наблюдателем и картинной плоскостью, в зависимости от которой различают два основных класса проекций: *параллельные* и *центральные*. В работе использовалась центральная проекция, а именно перспективная проекция. Для задания проекции использовалась команда

gluPerspective(Gldouble angley,Gldouble aspect,Gldouble znear,Gldouble zfar).

Предполагается, что точка схода имеет координаты (0, 0, 0) в видовой системе координат. Параметр angle задает угол видимости (в градусах) в направлении оси y. В направлении оси x угол видимости задается через отношение сторон aspect, которое обычно определяется отношением сторон области вывода. Два других параметра задают расстояние от наблюдателя (точки схода) до ближней и дальней плоскости отсечения.

Выполнение работы

Задание вполнено на языке C++ с использованием фреймворка Qt5.
Qt предоставляет возможность использование OpenGL посредством класса QGLWidget. От него отнаследован класс CustomGL.
`class CustomGL : public QGLWidget, в котором переопределены методы:`

- `void initializeGL();`
  Задает контекст рендеринга. Инициализирует шейдеры
- `void resizeGL(int nWidth, int nHeight);`
  `Для корректного масштабирования. Размер окна задается через glViewport.`
- `void paintGL();`
  Вызов этой функции происходит при рисовании.

В конструкторе класса CustomGL происходит создание шейдерной программы
В функции initializeGL происходит вызов initShaders() в которой загружается код шейдеров, затем линкуется и подгружается в шейдерную программу

shaderProgram->addShaderFromSourceCode(QGLShader::Fragment, _fsh.c_str()

Затем происходит первичная генерация многогранника.

Генерация морфирующего многогранника

Многогранник должен морфировать между кубом и тетраэдром, рис. 2.

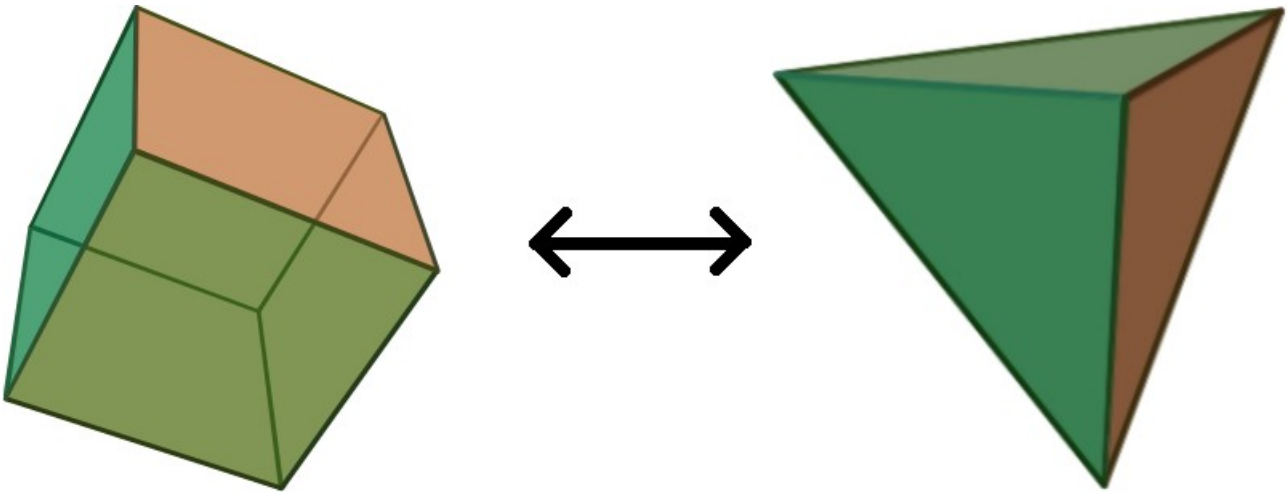

Рисунок 2 — Куб и тетраэдр, между которыми должен происходить морфинг.

Тетраэдр обладает очень полезным для нас свойством — он может быть построен на вершинах куба. Если взять три вершины инцидентные одной, и четвертую не инцидентную ни одной из первых трех, то многогранник построенный на них будет являться тетраэдром. Т.о. генерация многогранника и его морфинг сводится к генерации куба и проекцией вершин, не принадлежащим вершинам тетраэдра на плоскость тетраэдра либо на сферу. Данная фигура представлена на рис. 3

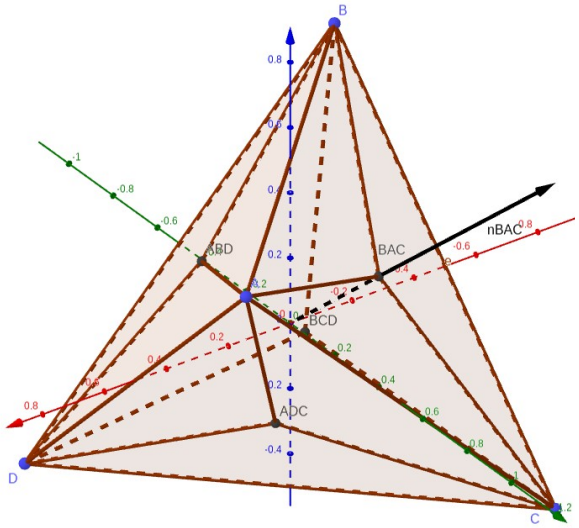

Рисунок 3 — Геометрия фигуры, морфируемой между тетраэдром и кубом
Обратите внимание на точку «BAC» и проходящий через нее вектор из начала координат, оканчивающийся в точке «nBAC». Положение точки BAC соответствует середине грани тетраэдра. При ее перемещении вдоль вектора в точку nBAC Она станет соответствовать вершине куба на верхней грани, которой также будут принадлежать точки A (ближняя к наблюдателю), B и «nABD» аналочично nBAC полученой проекцией ABD на сферу радиуса OA(= OB = OC).

Опишем генерацию данной фигуры на языке C++. Пусть функция построения многогранника принимает вещественное число p от 0 до 1, где 0 будет соответствовать тетраэдр а 1 — куб.
Определен массив вершин _r, который заполнен вершинами восемью вершинами: 4 соответствующие тетраэдру

```
std::vector<QVector3D> _r; // Morph vertexes
    _r.push_back({.5,.5,.5});
    _r.push_back({-.5,-.5,.5});
    _r.push_back({-.5,.5,-.5});
    _r.push_back({.5,-.5,-.5});
```

Далее определим положение остальных четырех вершин. При p==0 они должны лежать в серединах граней тетраэдра, при p==1 они должны быть спроецированы на сферу радиусом, равным расстоянию от центра фигуры до неподвижных вершин.

```
    double l0 = (_r[1]+_r[0]+_r[2]).length()/3;
    double l1 = _r[0].length();
    double l = (l0 + ((l1 - l0) * p)) / (l0*3);

    _r.push_back((_r[1]+_r[0]+_r[2])*l);
    _r.push_back((_r[0]+_r[1]+_r[3])*l);
    _r.push_back((_r[0]+_r[3]+_r[2])*l);
    _r.push_back((_r[1]+_r[2]+_r[3])*l);
```

Здесь `l0` – расстояние до центра граней, `l1` – расстояние до неподвижных вершин, а `l` драсчитывается как линейная интерполяция между ними с параметром p. Переменная l также домножена на нормировочный коэффициент (1/(l0*3)), что позволяет домножать его на сумму трех вершин соотв. граней тетраэдра без нормировки.

Далее по вершинам составлены грани. Вершины в гранях перечисляются по индексам против часовой стрелки. Для единообразия подвижные вершины стоят в центре.

```
// BAC
    faces.push_back({0,4,1});
    faces.push_back({1,4,2});
    faces.push_back({2,4,0});
    // ABD
    faces.push_back({1,5,0});
    faces.push_back({0,5,3});
    faces.push_back({3,5,1});
    // ADC
    faces.push_back({0,6,2});
    faces.push_back({2,6,3});
    faces.push_back({3,6,0});
    // BCD
    faces.push_back({1,7,3});
    faces.push_back({3,7,2});
    faces.push_back({2,7,1});
```

Здесь буквенные обозначения в комментариях к коду соответвуют центральным вершинам указанным на рис.3.

Для работы с OpenGL формируется массив вершин по тройкам. Для этого производится итерация по граням и добавление вершин по индексам в массив. Также в соотв. массивы добавляются цвета и нормали для наглядности отображения и просчета освещения. Нормали расчитываются как векторное произведение векторов построенных на вершинах треугольных полигонов.

```
    for (int i = 0; i < _r.size(); i++) {
        _colors.push_back({(GLfloat)(i % 3 == 0), (GLfloat)(i % 3 == 1),
(GLfloat)(i % 3 == 2)});
    }

    for (auto face: faces) {
        QVector3D v0 = _r[std::get<0>(face)];
        QVector3D v1 = _r[std::get<1>(face)];
        QVector3D v2 = _r[std::get<2>(face)];

        auto u = v1 - v0;
        auto v = v2 - v0;
        auto n = QVector3D::crossProduct(v, u).normalized();

        vertexes.push_back({v0[0], v0[1], v0[2]});
        prime_colors.push_back(_colors[std::get<0>(face)]);
        normals.push_back({n.x(), n.y(), n.z()});

        vertexes.push_back({v1[0], v1[1], v1[2]});
        prime_colors.push_back(_colors[std::get<1>(face)]);
        normals.push_back({n.x(), n.y(), n.z()});
```

```
        vertexes.push_back({v2[0], v2[1], v2[2]});
        prime_colors.push_back(_colors[std::get<2>(face)]);
        normals.push_back({n.x(), n.y(), n.z()});
    }
```

Графическое отображение фигуры

Перемещение камеры происходит при помощи мыши

В функции paintGL происходит задание типа прорисовки и теста глубины, после чего управление передается в функцию scene.
В ней происходит настройка матрицы трансформации(тип проекции, положение вращение и пр.), после чего последовательно отрисовываются оси координат и сам многогранник. Для их отрисовки используется шейдер.
В шейдер передаются вершины, нормали, цвета, матрица трансформации и источник света. Интенсивность света высчитывается с учетом направления нормали. На источник ярче, от источника тусклее. В коде шейдера это выглядит так:

```
gl_FragColor = vec4(vColor.xyz,1.0f) + vec4(val,val,val,1)*
dot(normalize(lightsource.xyz-position),normal);
```

Здесь:
`vColor` — цвет вершины
`val` — интенсивность света с учетом расстояния и коэфициента
`position` — положение точки
`lightsource.xyz` — положение источника (это vec4, в четвертой компоненте передана интенсивность для val)
normal — нормаль плоскости, закладывается в каждую вершину

Реализована анимация фигуры при отрисовки. Для этого происходит обновление по таймеру, увеличивающее счетчик. Его значение затем используется для задания степени морфинга (параметр `p` генерации от 0 до 1). и поворота. Т.о. фигура плавно сорфирует и вращается.

Анимацию можно отключить. Тогда фигуру можно будет вращать мышкой, а для морфинга добавлен бегунок.

Реализована возможность рисования сетки ребер без граней(wireframe), для наблюдения за морфингом. Это реализовано установкой режима:

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

Включение и отключение режима wireframe происходит нажатием клавиши «пробел»

Демонстрация работы

Вид интерфейса призапуске представлен на рис.4. Вверху слева виден небольшой chechBox, отвачающий за включение и отключение анимации. Внизу виден бегунов для управление морфингом. Его крайнее левое положение соответствует `p==0`, т.е. тетраэдру

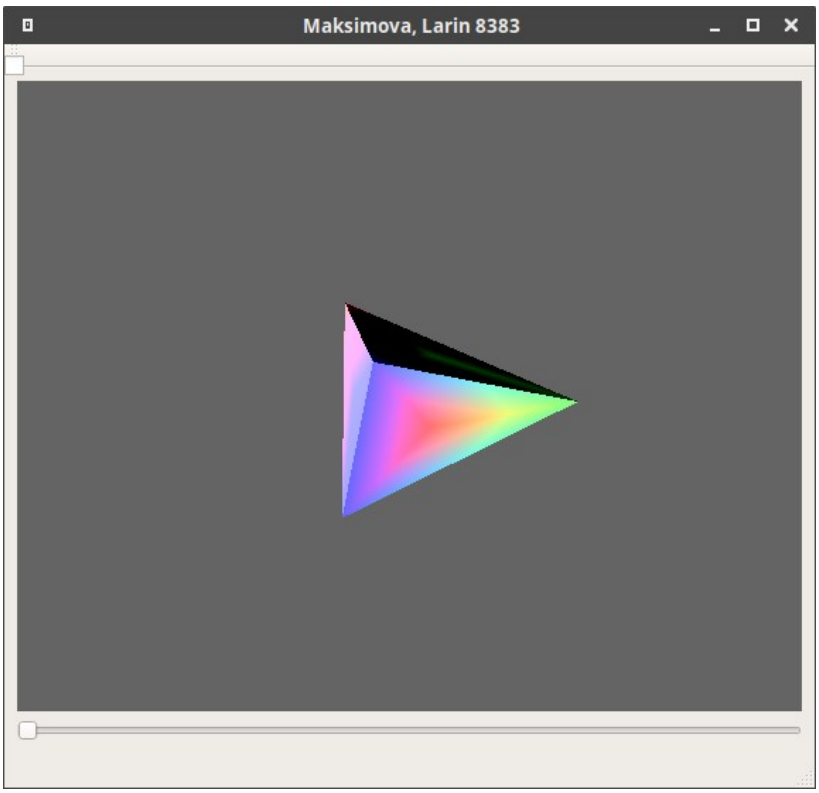

Рисунок 4 — Вид интерфейса и фигуры в виде тетраэдра

Бегунов выкручен на середину, пол пути между тетраэдром и кубом. Результат на рис.5. Фигура в виде куба (`p==1`) на рис.6

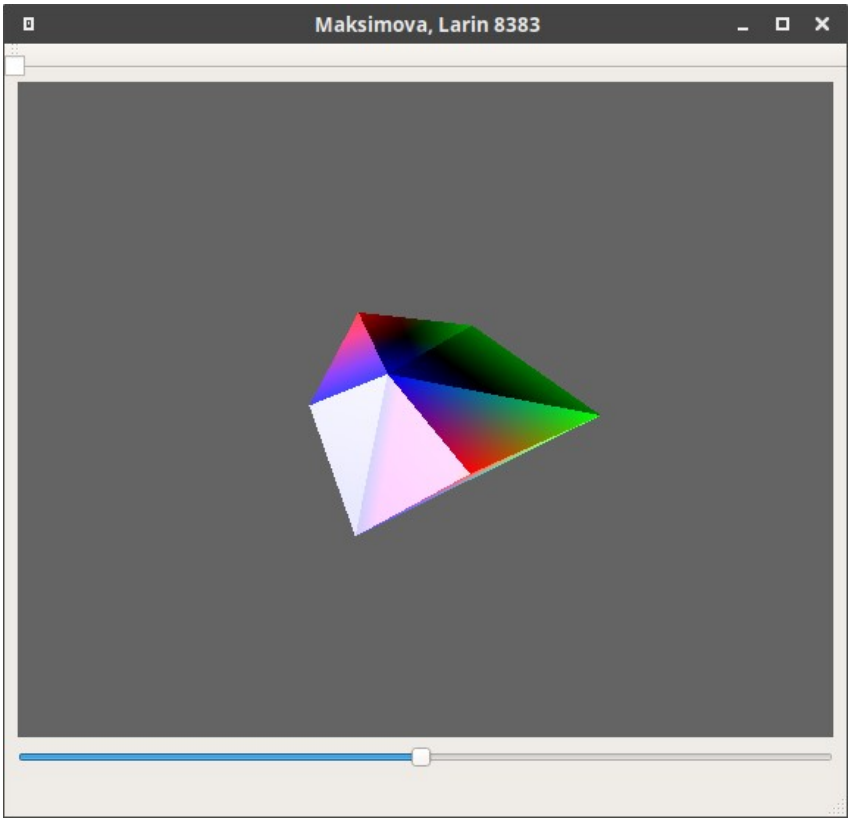

Рисунок 5 — Фигура на пол-пути между тетраэдром и кубом

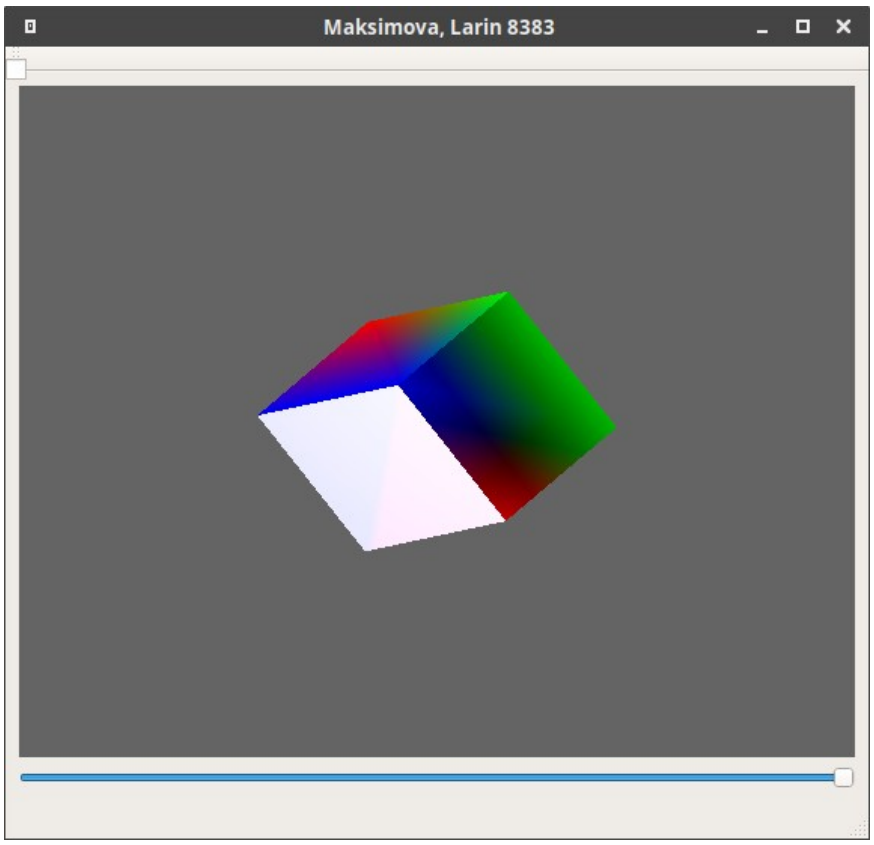

Рисунок 6 — Фигура в виде куба

На рис.7-9 представлены фигуры wireframe-ы фигур на рис.4-6 соответственно

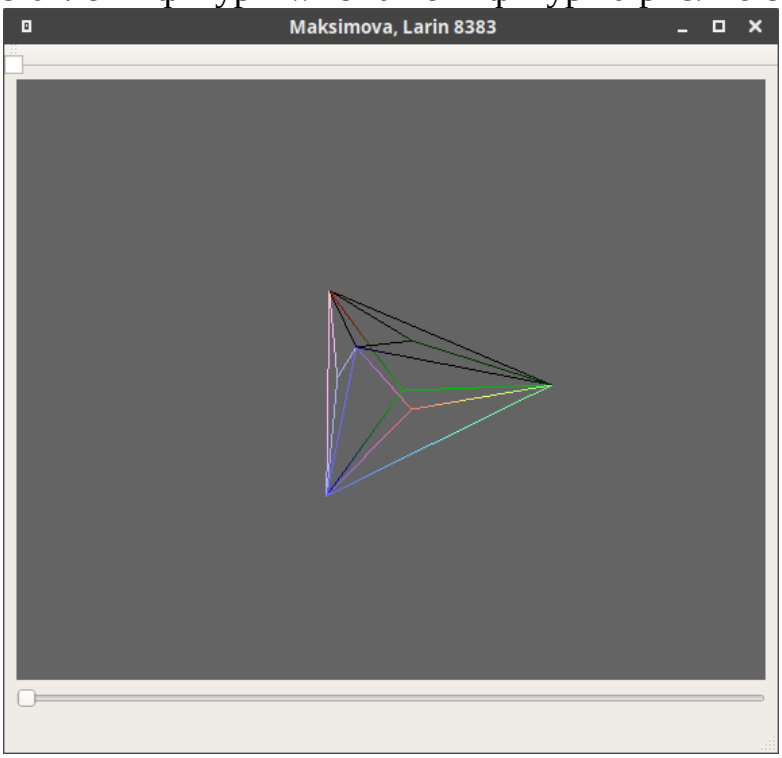

Рисунок 7 — Ребра фигуры в режиме тетраэдра

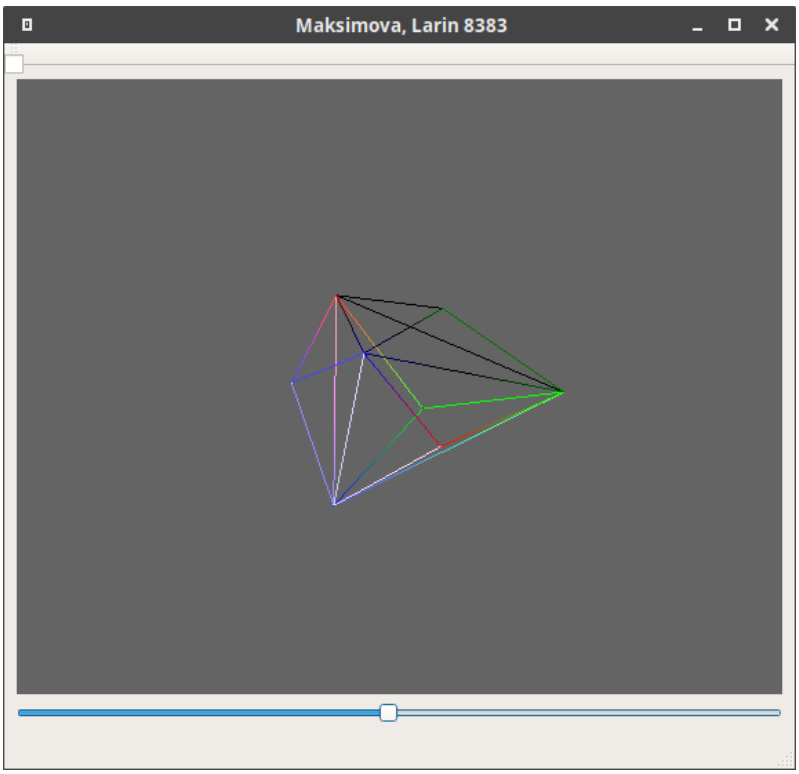

Рисунок 8 — Ребра фигуры на пол-пути между тетраэдром и кубом

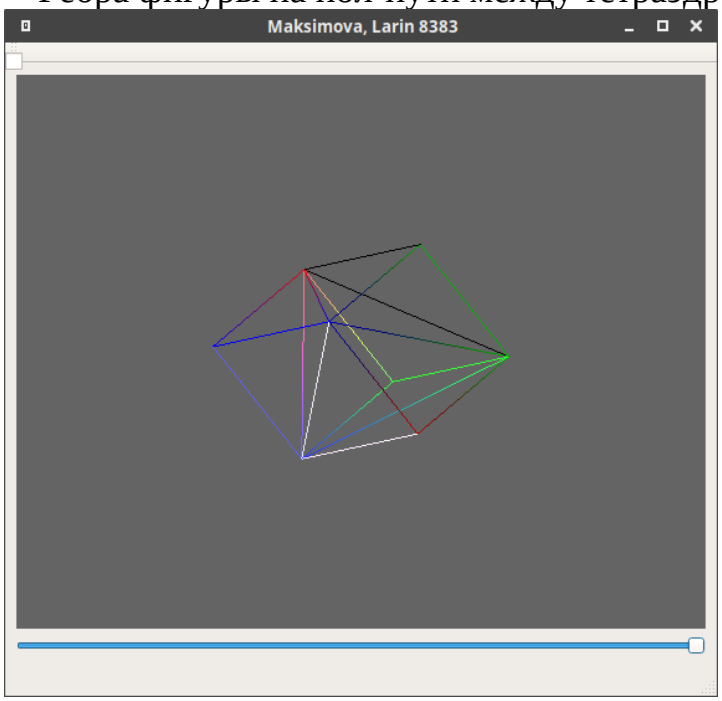

Рисунок 9 — Ребра фигуры в режиме куба

ЗАКЛЮЧЕНИЕ

Были изучены характеристики платоновых тел, с которыми предстояло провести работы. Была изучена проблема морфинга тел друг в друга. Проанализировано множество источников о существующих способов морфинга. Построена модель, описывающая конфигурацию морфирующей фигуры в зависимости от парамета. Модель была реализована в программе в виде параметрической функции генерации фигуры. Была имплементирована интерактивная визуализация фигуры с возможностью ручного управления и автоматической анимации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Dissecting a Cube into a Tetrahedron and a Square Pyramid // Wolfram. URL: https://demonstrations.wolfram.com/DissectingACubeIntoATetrahedronAndASquarePyramid
2. *Morph tetrahedron into cube // Wolfram. URL: https://codepen.io/thebabydino/details/sGfkb*
3. *Polyhedra Gallery // Virtual Math Museum. URL: http://virtualmathmuseum.org/Polyhedra/Tetrahedron/index.html#tetrahedron_cube_morph_001*
4. *Морфинг по Вороному // 3dcenter. URL: http://3dcenter.ru/tutors/read.php?sname=maxscript&articlealias=maxscript_voronoi*