ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»

(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА ПРИКЛАДНОЙ ИНФОРМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

РАЗРАБОТКА МОДУЛЯ ВИЗУАЛИЗАЦИИ ОРИЕНТИРОВАННЫХ 3D ГРАФОВ СРЕДСТВАМИ GLSCENCE

Выпускная квалификационная работа
обучающегося по направлению подготовки 010300.62(02.03.02)
«Фундаментальная информатика и информационные технологии»
заочной формы обучения, группы 07001360
Бекирова Владислава Руслановича

Научный руководитель:
доц. Васильев П.В.

БЕЛГОРОД 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Появление первых компьютеров в середине прошлого столетия привело к развитию вычислительных методов в области естественных наук, инженерных дисциплинах, управлении. Что заметно ускорило процессы разработки новых алгоритмов и математических моделей. Бурное развитие вычислительной техники, создание многопроцессорных систем, и появление суперкомпьютеров, позволило успешно производить сложные расчеты и выводить их в виде графики на мониторы, бумагу и т.д. В связи с этим разработка и оптимизация, направленная на снижение вычислительной трудоемкости, и быстрой обработкой графики является одной из важных задач в программировании.

Компьютерная графика в настоящее время уже сформировалась в отдельную отрасль науки. Трудно представить любой вид работы без применения компьютерной трехмерной графики. Образование, медицина, промышленность, телевидение, все эти отрасли используют трехмерную графику. Использование трехмерной графики в вычислительных системах, помогает людям наглядно увидеть результаты тех или иных вычислений в виде графиков или иных объектов вычисления. Актуальной задачей программирования с использованием трехмерной графики, является написание такой программы, которая была бы интуитивно понятна большему числу пользователей, при хорошей функциональности и быстрым вычислением, так как компьютерная трехмерная графика является ресурсозависимой от технических характеристик компьютера.

Объектом исследования выпускной квалификационной работы является ориентированный граф.

Целью является изучение теории графов и применения практических навыков программирования для написания модуля.

Основной задачей выпускной квалификационной работы является разработка модуля для визуализации 3D графа, на основании введенных данных с использованием графического движка GLScene.

Практической значимостью работы является создание простого приложения, с изменяемым набором данных для универсальности программы, и показом разных проекций ориентированного графа.

Во введении ВКР описывается актуальность данной работы и описание целей и задач для выполнения работы. В первом разделе описываются теоретические данные о теории графов, компьютерной графике, ее разновидностях и проблемах, а также общие сведения по теме. Второй раздел включает в себя выбор среды программирования и реализацию самого алгоритма модуля визуализации. Тестирование программы описывается в третьем разделе. Тестирование проводится по методу белого ящика.

Данная ВКР содержит 1 таблицу, 10 рисунков , 8 листингов кода и 1 приложение.

1. ИЗУЧЕНИЕ ОБЪЕКТА ИССЛЕДОВАНИЯ И МЕТОДОВ ВИЗУАЛИЗАЦИИ

1.1 ТЕОРИЯ ГРАФОВ

Теория графов является одним из важных разделов дискретной математики, который исследует свойства множеств с заданными отношениями между элементами этих множеств. Особенностью теории графов является геометрический подход к изучению объектов внешнего мира. Понятие «граф» ввел в 1936 году венгерский математик Денеш Кёниг. Первая работа по теории графов была написана еще в 1736 году Леонардом Эйлером, в которой он решил «задачу о Кёнигсбергских мостах». Сутью этой задачи является то, что есть часть города, которая включает в себя два берега реки, два острова в и семь соединяющих их мостов. Задачей требуется обойти четыре части суши, пройдя по каждому мосту всего один раз, и вернуться в начальную точку.

Теория графов даёт удобный аппарат для моделирования структурных свойств различных отношений между объектами разной природы. Поэтому она широко используется как в самой математике, так и ее приложениях в самых разнообразных областях техники, науки и практической деятельности. В частности, теория графов находит свое применение в информатике и программировании, химии, экономике, логистике, в коммуникационных и транспортных системах, схемотехнике. Нужно понимать, что для понятия «граф» нет общего единого определения.

Графом $G(V, E)$ называется совокупность двух множеств — непустого множества V (множества вершин), множества E неупорядоченных пар различных элементов множества V (E — множество ребер).

Соединения между узлами графа называются ребрами. Если узлы графа не нумерованы, то его ребра являются неориентированными. У графа с нумерованными узлами ребра ориентированы. Ребрам графа могут быть присвоены определенные веса или метки. Более простым определением графа является - совокупность точек и линий, в которых каждая линия соединяет две точки. Для ориентированного графа $E \subseteq Vx$ это конечный набор ориентированных ребер.

Ребром может быть прямая или кривая линия. Ребра не могут иметь общих точек кроме вершин (узлов) графа. Замкнутая кривая в E может иметь только одну точку из множества V, а каждая незамкнутая кривая в E имеет ровно две точки множества V. Если V и E конечные множества, то и граф соответствующий им называется конечным . Граф называется вырожденным , если он не имеет ребер. Параллельными ребрами графа называются такие, которые имеют общие узлы начала и конца.

Если ребро соединяет две вершины, то говорят, что оно им инцидентно; вершины, соединенные ребром называются смежными. Две вершины, соединенные ребром, могут совпадать; такое ребро называется петлей. Число ребер, инцидентных вершине, называется степенью вершины. Если степень вершины равна 0, то получается изолированная графа. Если два ребра инцидентны одной и той же паре вершин, они называются кратными; граф, содержащий кратные ребра, называется мультиграфом.

Часто рассматриваются следующие родственные графам объекты:

- Если элементы множества E являются упорядоченными парами, то граф называется ориентированным (или орграфом). В этом случае элементы множества V называются узлами, а элементы множества E — дугами.
- Если элементом множества E может быть пара одинаковых (не различных)элементов V , то такой элемент множества E называется петлей, а граф называется графом с петлями (или псевдографом).

- Если E является не множеством, а набором, содержащим несколько одинаковых элементов, то эти элементы называются кратными ребрами, а граф называется мулътиграфом.
- Если элементами множества E являются не обязательно двухэлементные, а любые подмножества множества V, то такие элементы множества E называются гипердугами, а граф называется гиперграфом. Графы отображаются на плоскости набором точек и соединяющих их линий или векторов. При этом грани могут отображаться и кривыми линиями, а их длина не играет никакой роли.

Граф G называется плоским , если его можно отобразить в плоскости без пересечения его граней. Очертанием графа считается любая топологически связанная область, ограниченная ребрами графа. Неориентированный граф G = <V,E> называется связанным , если для любых двух узлов x,y O V существует последовательность ребер из набора E, соединяющий x и y. Граф G связан тогда и только тогда, когда множество его вершин нельзя разбить на два непустых подмножества V1 и V2 так, чтобы обе граничные точки каждого ребра находились в одном и том же подмножестве. Граф G называется k-связным (k i 1), если не существует набора из k-1 или меньшего числа узлов V`H V, такого, что удаление всех узлов V` и сопряженных с ними ребер, сделают граф G несвязанным,[8].

Способы численного представления графа:

- Матричный способ (с помощью матрицы смежности). Матрица смежности имеет m– строк и n– столбцов, где m– количество вершин графа.

Элементами матрицы смежности являются 0 и 1, Если вершины соединены, то ставится 1 и наоборот.

Таблица 1.1. Матрица смежности для графа указанного на рисунке 1

	a	b	c	d	e
a	0	0	1	1	0
b	0	0	1	1	1
c	1	1	0	0	0
d	1	1	0	0	0
e	0	1	0	0	0

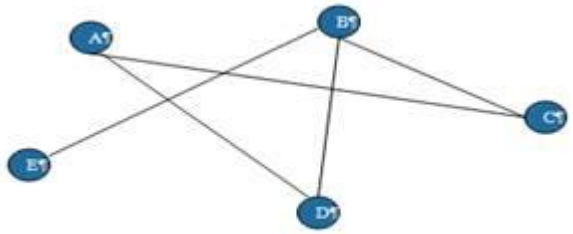

Рис.1.1 Неориентированный граф

Графы широко используются в строительстве, электротехнике, менеджменте, логистике, географии, машиностроении, программировании, автоматизации технологических процессов и производств, психологии, рекламе. Итак, из всего вышесказанного неопровержимо следует практическая ценность теории графов, доказательство которой и являлось целью данного исследования.

В любой области науки и техники можно встретиться с графами. Графы - это математические объекты, с помощью которых можно решать математические, экономические и логические задачи, различные головоломки и упрощать условия задач по физике, химии, электронике, автоматике. Многие математические факты удобно формулировать на языке графов. Теория графов является частью многих наук. Теория графов — одна из самых красивых и наглядных математических теорий. В последнее время теория графов находит всё больше применений и в прикладных науках.

Возникла даже компьютерная химия — сравнительно молодая область химии, основанная на применении теории графов.

1.2 ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ ТЕОРИИ ГРАФОВ

Если две вершины соединены направленным отрезком, то пара называется упорядоченной, а отрезок называется ребром графа. Если вершины соединены ненаправленным отрезком, то вершины называются неупорядоченными, отрезок, их соединяющий, называется дугой. Граф, содержащий только ребра, называется ориентированным. Граф, содержащий только дуги, называется неориентированным. Пара вершин может соединяться двумя или более ребрами одного направления, такие ребра называются кратными. Дуга или ребро может начинаться или заканчиваться в одной вершине, такие дуги называются петлями. Считается, что длина петли равна 1. Вершины, соединенные ребром или дугой называются смежными. Дуги, имеющие общие вершины называются смежными. Ребро и любая из двух ее вершин называется инцидентными.

Длиной пути М называется число К, равное числу дуг, составляющих путь М.

Путь, в котором ни одна дуга не встречается дважды, называется простым. Путь, в котором ни одна вершина не встречается дважды, называется элементарным.

Контур – это конечный путь М, у которого начальная и конечная вершина совпадают. При этом контур называется элементарным, если все его вершины различны (за исключением начальной и конечной вершины).

Для неориентированного графа аналогичными понятиями являются понятия цепи и цикла. С понятием неориентированного графа связано понятие связности графа

Граф связен, если любые две его вершины можно соединить цепью. Если граф не связен, то его можно разбить на такие подграфы, что все вершины в каждом подграфе связны, а вершины из различных подграфов не связны. Такие подграфы называются компонентами связности графа.

Граф сильно связен, если для любых вершин x y существует путь, идущий из x в y.

1.3 ОПЕРАЦИИ НАД ГРАФАМИ

Для получения новых графов из уже имеющихся можно использовать разнообразные операции над графами. Есть два вида операций – локальные, при которых заменяются, удаляются или добавляются отдельные элементы графа, и алгебраические, когда новый граф строится по определенным правилам из нескольких имеющихся.

Локальные операции. Простейшая операция – удаление ребра. Все вершины графа и все его ребра, кроме удаляемого, остаются в неприкосновенности. Заметим, что при удалении ребра получается остовный подграф исходного графа, и любой остовный подграф можно получить удалением некоторых ребер.

Обратная операция – добавление ребра. При удалении вершины вместе с вершиной удаляются и все инцидентные ей ребра. Получается подграф исходного графа, порожденный множеством оставшихся вершин, и любой порожденный подграф можно получить удалением некоторых вершин. При добавлении вершины к графу добавляется новая изолированная вершина. С помощью операций добавления вершин и ребер можно "из ничего", то есть из графа K0, построить любой граф.

Операция стягивания ребра (a, b) преобразует граф следующим образом. Вершины a и b удаляются из графа, к нему добавляется новая вершина c и она соединяется ребром с каждой вершиной, с которой была смежна хотя бы одна из вершин a, b.

Операция подразбиения (подразделения) ребра (a, b) действует следующим образом. Из графа удаляется это ребро, к нему добавляется новая вершина c и два новых ребра (a, c) и (b, c).

Алгебраические операции Поскольку граф состоит из двух множеств (вершины и ребра), то различные операции над множествами естественным образом порождают соответствующие операции над графами. Например, объединение двух графов G1 и G2 определяется как граф G = G1 ∪ G2, у которого VG = VG1 ∪ VG2, EG = EG1 ∪ EG2, а пересечение – как граф G = G1 ∩ G2, у которого VG = VG1 ∩ VG2, EG = EG1 ∩ EG2.

Объединение графов с непересекающимися множествами вершин часто называют их суммой и записывают как G1 + G2. Мы будем применять операцию сложения к любым графам, понимая под этим следующее: сначала вершины графов-слагаемых переименовываются так, чтобы множества вершин стали непересекающимися (очевидно, достаточно переименовать вершины одного слагаемого), затем полученные графы объединяются. По существу, речь идет о сложении абстрактных графов. Операция сложения абстрактных графов ассоциативна, то есть (G1 + G2) + G3 = G1 + (G2 + G3) для любых трех графов. Поэтому можно образовывать сумму любого числа графов, не указывая порядок действий с помощью скобок. Если складываются k экземпляров одного и того же графа G, то полученный граф обозначается через kG. Например, 1 On ≅ nK .

Соединением двух графов G1 и G2 называется граф, получаемый из их суммы добавлением всех ребер, соединяющих вершины первого слагаемого с вершинами второго. Будем записывать эту операцию как G1 ◦ G2. H а рисунке 1.21 представлен граф P3 ◦ O2. Легко видеть, что операции

сложения и соединения графов связаны друг с другом следующими простыми соотношениями: G1 G2 G1 G2 + = , G1 G2 = G1 + G2

Произведение G = G1 × G2 графов G1 и G2 определяется следующим образом. Множеством вершин графа G является декартово произведение множеств VG1 и VG2, то есть вершины этого графа – упорядоченные пары (x, y), где x – вершина первого сомножителя, y – вершина второго. Вершины (x1, y1) и (x2, y2) в G смежны тогда и только тогда, когда x1 = x2 и y1 смежна с y2 в графе G2, или y1 = y2 и x1 смежна с x2 в графе G1. С помощью операции произведения можно выразить некоторые важные графы через простейшие. Например, произведение двух цепей дает прямоугольную решетку. Если один из сомножителей превратить в цикл, добавив одно ребро, то прямоугольная решетка превратится в цилиндрическую, а если и второй сомножитель превратить в цикл, то получится тороидальная решетка. Другой пример – k-мерный куб Qk, описанный в задаче 3 предыдущего раздела.

1.4 ИЗУЧЕНИЕ ВИДОВ КОМПЬЮТЕРНОЙ ГРАФИКИ

Компьютерная графика- область деятельности, в которой компьютеры используются для обработки введенных данных, и выводов этих данных с помощью изображения и визуализации.

Существует множество сфер применения компьютерной графики: интерфейс пользователей, кинематография, интернет, телевидение, компьютерная томография и др. Компьютерная графика делится несколько видов графики. 2D графика которая включает в себя: растровую, фрактальную и векторную. 3D графика в которой для постройки изображения используется 3 плоскости, так же 3D графика называется трехмерной. 2D графика— это такая графика, которая классифицируется по различным типам представления графической информации.

Векторная графика— графика, которая создает изображения в виде геометрических примитивных фигур. В эти геометрические фигуры входят:

точки, круги, прямоугольники и прямые. Объектам этой графики присваивается толщина линий и цвет заполнений. Такие изображения хранятся в виде векторов, координат и других символов, которые являются геометрическими примитивами.

Данные изображения дают возможность редактирования обьектов в любой размер при наилучшем качестве изображения. Картинка может деформироваться, поворачиваться, имитироваться в трехмерности, все это происходит без потери своих размеров. Каждое такое редактирование происходит таки образом: стирается старое изображение или его фрагменты, а вместо него образовывается новое. Математическое описание изображения не изменяется, редактируются лишь значения его коэффициентов или других переменных. При преобразовании рисунка исходными данными является только число пикселей. Поэтому возникает проблема изменения такого числа на больше или на меньше. Решением такой проблемы являются алгоритмы интерполяции. При этих алгоритмах новые пиксели получают коды цветов, которые вычисляются на основе соседних цветов. Но не каждое изображение возможно представить в виде геометрических примитивов. Данный вид графики применяется для создания мультфильмов, масштабирования шрифтов, различных схем, создания абрисов для графического и ЧПУ оборудования.

Растровая графика- графика, которая всегда представляется двухмерной матрицей пикселей. Определенному пикселю сопоставляется определенный элемент цвета, яркости, а также их комбинаций. Образ таких изображений имеет особое число строк и столбцов.

Растровые изображения можно лишь уменьшать при редактировании. Увеличить такие изображения можно, но качество станет намного хуже. Увеличенные рисунки будут состоять из квадратов, которые раньше были пикселями. В растровой графике можно представить любое изображения. Но такая графика имеет множество недостатков в техническом плане.

Фрактальная графика— графика, элементы которой имеют свойства родительских структур. Фракталы дают возможность описывать классы изображений, детальное описание которых требует минимальное количество памяти.

Трехмерная графика — это вид графики, создающий объекты в трехмерном режиме. Результаты таких объектов имеют вид плоской картинки с проекцией. Изображения данного вида графики выглядят, как набор частиц и поверхностей. Трехмерную графику широко используют в кино и компьютерных играх.

На самом деле, каждое изображение на мониторе становится растровым. Потому, что монитор- это матрица, которая состоит из строк и столбцов. А трехмерная графика это наше воображение. Поэтому компьютерная графика бывает векторная и растровая. А все остальное- набор пикселей, заданных изображениями этих видов графики.

1.5 ОБЩИЕ ПОНЯТИЯ О 3D ГРАФИКЕ

В настоящее время 3D графика уже является вполне сформировавшейся отраслью в науке. Появляется все больше программных продуктов для работы с компьютерной графикой и методов ее обработки, лучшей оптимизации для пользователей и средств ее обработки.

Традиционно рисуют в 2D (по осям X и Y) — на холсте, дереве, бумаге и т.п. При этом отображая только одну из сторон предметов. Картинка сама по себе является плоской. Но если мы захотим получить представление обо всех сторонах предмета, то будет необходимо нарисовать несколько рисунков. Таким образом поступают в традиционной рисованной анимации. Но, вместе с тем, существует, кукольная анимация. Один раз изготовленную куклу снимают в необходимых позах и ракурсах, получая серию «плоских картинок». 3D (к X и Y добавляется координата глубины Z) визуализация —

это те же «куклы», только существующие в цифровом виде. Другими словами, в специальных программах создается объемное изображение.

Преимущество данного метода в том, что в распоряжении, пользователя есть объемная модель, необходимо лишь перенести ее должным образом в кадр, анимировать (задать траекторию передвижения или рассчитать с помощью симулятора) при необходимости, отображение в финальной картинке ложится на специальную программу называемую визуализатором (render). Основное преимущество в том, что модель достаточно отрисовать один раз, а потом использовать в других проектах , изменять, деформировать по своему усмотрению. Для обычного 2D рисунка, такое невозможно. Можно создавать практически бесконечно детализированные модели, например смоделировать мелкую деталь на большом объекте. На общем плане эта деталь может быть и не видна, но стоит нам приблизить камеру, программа-визуализатор сама рассчитает, что видно в кадре, а что — нет,[1].

Выделяют следующие основные виды трехмерной компьютерной графики: полигональная, аналитическая, сплайновая графика. Иногда сюда же относят и фрактальную графику, которая уже была рассмотрена.

Полигональная графика является самой распространенной, так как отличается очень высокой скоростью ее обработки. Любой объект полигональной графики задается набором полигонов. Полигон - это плоский многоугольник. Простейшим вариантом являются треугольные полигоны, потому как известно, через любые три точки в пространстве можно провести плоскость.

Каждый полигон задается несколькими точками. Лучше всего задавать точки в том порядке, в каком они расположены относительно внешней нормали полигона, так как в данном случае полностью решается проблема удаления нелицевых граней без дополнительных затрат времени при последующей обработке. Точка же задается тремя координатами – x, y, z.

Таким образом, можно задать трёхмерный объект как массив или как структуру.

Аналитическая графика заключается в том, что объекты задаются аналитически, т.е. формулами. То есть дается формула относительно которой визуализатор создает изображение объекта , который был задан введенными данными.

Комбинируя различные формулы друг с другом, можно получить оригинальные объекты обтекаемой формы. Но вся сложность заключается в нахождении формулы требуемого объекта. Другой способ создания аналитических объектов – это создание тел вращения. Так, вращая круг вокруг некоторой оси, можно получить тор, а, вращая одновременно сильно вытянутый эллипс вокруг собственной и внешней осей, можно получить достаточно красивый рифленый тор.

Сплайновая графика. Сначала сплайны рассматривались как удобный инструмент в теории и практике приближения функций. Однако довольно скоро область их применения начала быстро расширяться, и обнаружилось, что существует очень много сплайнов самых разных типов. Сплайны стали активно использоваться в численных методах, в системах автоматического проектирования и автоматизации научных исследований, во многих других областях человеческой деятельности и, конечно, в компьютерной графике. Сам термин "сплайн" происходит от английского spline. Именно так называется гибкая стальная полоска, при помощи которой чертежники проводили через заданные точки плавные кривые. Появление компьютеров позволило перейти к эффективному способу задания поверхности обтекаемого тела. В основе этого подхода к описанию поверхностей лежит использование сравнительно несложных формул, позволяющих восстанавливать облик изделия с необходимой точностью. Средства компьютерной графики, визуализация, позволяют конструктору увидеть, что

может получиться в результате, давая ему, возможность сравнить это с тем, что сложилось у него в голове.

1.6 ОБЛАСТИ ПРИМЕНЕНИЯ ТРЕХМЕРНОЙ ГРАФИКИ

Компьютерное проектирование. К области автоматизированного проектирования относятся применения 3D-графики в целях синтеза внешнего вида сложных отливок, деталей, изготовляемых методами штамповки, токарных и фрезерных операций, визуального облика проектируемых автомобилей, катеров, самолетов и т. п.

Компьютерные игры. Это самая большая из наиболее широких областей применения 3D-графики. По мере усовершенствования программных средств моделирования 3D графики, роста производительности и увеличения ресурсов памяти компьютеров виртуальные трехмерные миры, в которых действуют персонажи компьютерных игр, становятся все более сложными и похожими на реальность.

Комбинированная съемка. Трехмерная графика помогает там, где выполнение реальной фотосъемки невозможно, затруднительно или требует значительных материальных затрат, а также позволяет синтезировать изображения событий, которые не встречаются в обыденной жизни.

Компьютерная мультипликация. Областями использования 3D-графики для создания компьютерной мультипликации являются телевизионная реклама, киносъемка с включением анимационных эффектов, подготовка видеороликов на научно-популярные или фантастические сюжеты, создание видеотренажеров для обучения.

1.7 НЕДОСТАТКИ ТРЕХМЕРНОЙ ГРАФИКИ

Повышенные требования к аппаратной части компьютера, в частности к объему оперативной и видео памяти, наличию свободного места на жестком диске и быстродействию процессоров. Так как трехмерная графика требует больших ресурсов аппаратных средств, для работы с ней приходится применять более новые системы преобразования в графических процессорах, которые так же зависят от объема памяти и уровня центрального процессора. Простыми словами для работы с трехмерной графикой требуются мощные компьютеры.

Необходимость большой подготовительной работы по созданию моделей всех объектов сцены, которые могут попасть в поле зрения камеры, и по присвоению им материалов и своиств.

Необходимость контролировать взаимные положения объектов в составе сцены, особенно при выполнении анимации.

Необходимость принятия дополнительных мер, обычно применяемых на этапе вторичной обработки синтезированных изображений, чтобы модернизировать картинку, придав ей более правдоподобный вид. В связи с этим в состав программ трехмерной графики входит целый ряд фильтров, позволяющих имитировать такие эффекты как конечная глубина резкости изображений или смазывание, вызванное движением объектов в момент съемки.

1.8 ОБРАБОТКА КОМПЬЮТЕРНОЙ ГРАФИКИ

Обработка компьютерной графики является самым важным набором действий вывода изображения на экран. Набор таких действий называют 3D-конвейером — каждый этап в конвейере работает с результатами предыдущего.

На самом первом, подготовительном, этапе программа определяет, какие объекты с какими текстурами и эффектами, в каких местах и в какой фазе анимации нужно отобразить на экране. Также выбирается положение и ориентация виртуальной камеры, через которую пользователь смотрит на изображаемый мир. Весь этот материал, подлежащий обработке на конструкторе, называется 3D-сценой.

Далее наступает очередь 3D-конвейера. Первым шагом является тесселяция — процесс деления сложных поверхностей на треугольники. Следующим этапом идут взаимосвязанные процессы трансформации координат точек или вершин, из которых состоят объекты, их освещения, а также отсечения невидимых участков сцены.

Пример трансформации координат. У нас имеется трехмерный мир, в котором расположены разные трехмерные объекты, а в итоге нужно получить двумерное плоское изображение этого мира на мониторе. Поэтому все объекты проходят несколько стадий преобразования в разные системы координат, называемых еще пространствами (spaces). Вначале локальные, или модельные, координаты каждого объекта преобразовываются в глобальные, или мировые, координаты. То есть, используя информацию о расположении, ориентации, масштабе и текущем кадре анимации каждого объекта, программа получает уже набор треугольников в единой системе координат. Затем следует преобразование в систему координат камеры (camera space), с помощью которой мы смотрим на моделируемый мир. После чего отсчет будет начинаться из фокуса этой камеры — по сути как бы "из глаз" наблюдателя.

Параллельно производится освещение (lighting). По полученной информации о расположении, цвете, типе и силе всех размещенных в сцене источников света рассчитывается степень освещенности и цвет каждой вершины преобразованного треугольника. Эти данные будут использованы

позже при растеризации. В самом конце, после коррекции перспективы, координаты трансформируются еще раз, теперь уже в экранное пространство (screen space).

На этом заканчивается трехмерная векторная обработка изображения и наступает очередь двумерной, т. е. текстурирования и растеризации. Сцена теперь представляет собой псевдотрехмерные треугольники, лежащие в плоскости экрана, но еще с информацией о глубине относительно плоскости экрана каждой из вершин. Растеризатор начинает вычислять цвет всех пикселов, составляющих треугольник, и заносит его в кадровый буфер. Для этого на треугольники накладываются текстуры, часто в несколько слоев (основная текстура, текстура освещения, детальная текстура и т. д.) и с различными режимами модуляции. Также производится окончательный расчет освещения с использованием любой модели затенения, теперь уже для каждого пиксела изображения. На этом же этапе выполняется окончательное удаление невидимых участков сцены. Ведь треугольники могут располагаться на разном расстоянии от наблюдателя, перекрывать друг друга полностью или частично, а то и пересекаться. Сейчас повсеместно применяется алгоритм с использованием Z-буфера. Результирующие пикселы заносятся в Z-буфер, и как только все изображение будет готово, его можно отображать на экране и начинать строить следующее.

1.9 ОСНОВНЫЕ ПОНЯТИЯ ПРИ ПРОЕКТИРОВАНИИ ТРЕХМЕРНОЙ ГРАФИКИ

3D-графика предназначена для имитации фотографирования или видеосъемки трехмерных образов объектов, которые должны быть предварительно подготовлены в памяти компьютера. В большинстве подсистем трехмерной графики применяется графический конвейер.

Конвейер – это логическая группа вычислений, выполняемых последовательно одна за другой, дающих на выходе синтезируемую сцену.

Конвейер разделен на множество этапов, на каждом из которых аппаратно или программно выполняется некоторая функция. Наличием переходов между этапами конвейера обеспечивается возможность выбора между программной и аппаратной реализацией очередного этапа. Такой подход к настройке конвейера позволяет приложениям трехмерной графики получать преимущества аппаратной реализации. Таким образом, реализация конвейера может быть программной, полностью аппаратной или смешанной (программно- аппаратной).

Первоначально объект представляется в виде набора точек или координат в трехмерном пространстве. Трехмерная система координат определяется тремя осями: горизонтальной, вертикальной и глубины, т.е. осями x, y и z. Объектом может быть дом, человек, машина, самолет или целый 3D мир. Координаты определяют положение вершин (узловых точек) объекта в пространстве. При помощи соединения вершин объекта линиями можно получить каркасную модель трехмерного тела. Каркасная модель определяет области, составляющие поверхности объекта, которые могут быть заполнены цветом, текстурами и освещаться лучами света.

Основные понятия:

1) Превращения. Операции изменения расположения, размера, или ориентации объекта в пространстве. В общем случае - перемещение, масштабирование и вращение,[10].
2) Деформации. Операции подобные превращениям, но более сложные, так как их исполнение приводят к перемене внешнего вида объекта. К деформациям можно отнести искривление, поворот, рассогласование и т.п. ,[10].
3) Распределение- процесс назначения объекту свойств придающих ему реалистичность. Важнейшая характеристика для оценки трехмерных сцен - реалистичность, однако, простое моделирование трехмерных

объектов не всегда позволяет добиться реалистичности и правильной освещенности. В этом случае, на помощь приходит распределение текстур ,[10].

4) Мультитекстурирование позволяет конвейеризировать наложение текстур с использованием нескольких (обычно, не менее двух) блоков текстурирования,[10].
5) Конвейер рендеринга выполняется по многоступенчатому механизму. Конвейер рендеринга может быть разделен на три стадии: тесселяция, геометрическая обработка и растеризация. Если взять произвольный 3D- ускоритель, то он не будет ускорять все стадии конвейера, и даже более того, стадии могут лишь частично ускоряться им,[10].
6) Тесселяция – процесс разбиения поверхности объектов на треугольники. Эта стадия проводится полностью программно вне зависимости от технического уровня и цены 3D- аппаратуры. Геометрическая обработка делится на несколько фаз и может частично ускоряться 3D-ускорителем,[10].
7) Растеризация наиболее интенсивная операция, обычно реализуемая на аппаратном уровне. Растеризатор выполняет непосредственно рендеринг и является наиболее сложной ступенью конвейера. Если стадия геометрической обработки работает с вершинами, то растеризация включает операции над пикселями. Растеризация включает в себя затенение,[10].
8) Трансформация – преобразование координат (вращение, перенос и масштабирование всех объектов) ,[10].
9) Расчет освещенности – определение цвета каждой вершины с учетом всех световых источников,[10].
10) Проецирование – преобразование координат в систему координат экрана,[10].

11) Коррекция перспективы. Способность корректировать текстуры таким образом, чтобы у наблюдателя создавалось впечатление перспективы объекта,[10].

12) Антиалиасинг. Т.к. цифровые изображения, в основном, представляют собой матрицу из точек, строки этой матрицы, будучи проведенными не строго по горизонтали или вертикали прорисовывают объект неровными, зубчатыми линиями. Наиболее распространенный метод борьбы с этим эффектом состоит в том, что пиксели в этих зубчиках заполняются цветом,[10].

13) Наложение рельефа. Методика моделирования рельефных поверхностей. Для того чтобы подчеркнуть бугорки и впадины рельефа с помощью светотени, надо затемнить либо осветлить стенки этих бугорков и впадин. Другой метод состоит в симуляции рельефности глянцевой или зеркальной поверхности отражением окружающей среды. Это и делает техника наложения рельефа,[10].

14) Затуманивание. Один из наиболее распространенных эффектов. Отдаленные объекты или их детали, как бы скрываются в тумане и проясняются по мере приближения зрителя к ним. Эффект используется не только для моделирования атмосферного явления – уменьшая число деталей разработчики делают сцену менее сложной,[10].

2. РЕАЛИЗАЦИЯ МОДУЛЯ ВИЗУАЛИЗАЦИИ ОРИЕНТИРОВАННОГО 3D ГРАФА

2.1 ВЫБОР СРЕДЫ РАЗРАБОТКИ

В качестве языка программирования и среды разработки принято решение использовать язык Delphi и EMBARCADERO RAD Studio Berlin 10.1 соответственно. В паре со средой разработки оправдан выбор Delphi компилятора для лучшей работы с графическим движком GLScene . На рис.2 представлен интерфейс RAD Studio.

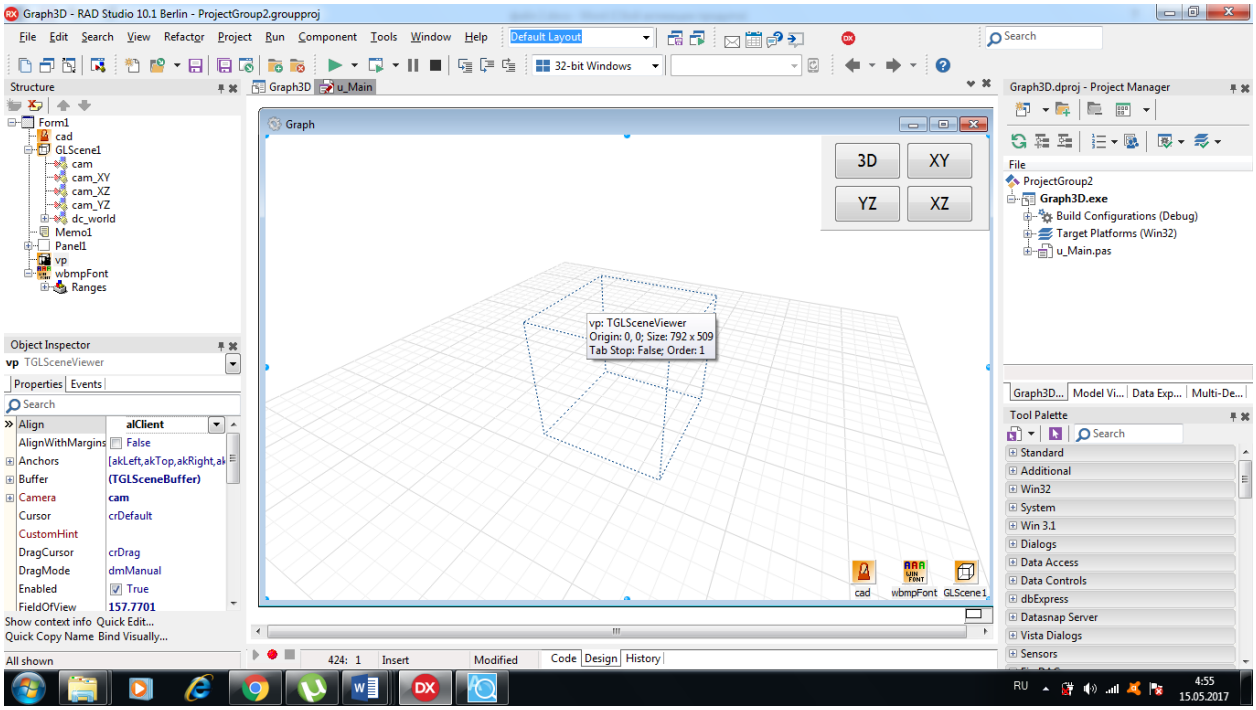

Рис.2.1 Интерфейс RAD Studio

RAD Studio Berlin 10.1 является мощной IDE средой для разработки кросс-платформенных приложений. Эта среда подойдет для широкого спектра разработчиков, а именно для: Delphi,C++, для Android и IOs устройств. Также в ней можно программировать для новой Windows 10.

Существует несколько вариантов программного продукта:

- RAD Studio Starter — для начинающих разработчиков: позволяет создавать приложения, которые взаимодействуют с локальными базами данных. Поставляется без исходных кодов. Разрешено использовать в организациях с не более, чем 5 лицензиями одновремено, а также для разработки программ при сумме дохода до 1000 долларов США.
- RAD Studio Professional — позволяет создавать приложения, которые взаимодействуют с локальными базами данных.
- RAD Studio Enterprise — обеспечивает возможность создания клиент-серверных решений, многоуровневых баз данных и веб-приложений.
- RAD Studio Architect — моделирует и проектирует базы данных Embarcadero ER/Studio Developer Edition.

GLScene — графический движок для создания кросс-платформенных приложений на языках программирования Delphi, Free Pascal и C++, использующий библиотеку OpenGL в качестве интерфейса программирования приложений. GLScene является свободным программным обеспечением и распространяется с лицензией Mozilla Public License. С его помощью программирование трёхмерной графики в Windows становится более простым и понятным. Последние версии этого движка также доступны в среде программирования Lazarus для создания приложений для Linux и др.операционных систем.

Разработка данного движка началась в 1999 году Майком Лишке, а с версии 0.5 была выложена в сеть с открытым исходным кодом. Дальнейшее развитие GLScene было продолжено Эриком Гранжем, а после 2006 года поддерживается командой опытных разработчиков. В настоящий момент движок пополнился новыми функциями и стал быстрее благодаря различным оптимизациям проводимыми ими.

GLScene позволяет всем программистам создавать 3D-объекты OpenGL в design-time с использованием интерфейса, показанного на рисунке 2.1. Большое количество объектов и дополнительных визуальных

компонентов VCL помогает программистам создавать мощные 3D-приложения для Delphi, C++Builder и Lazarus.

Так же с помощью GLScene возможно создавать приложения для мобильных платформ Android и IOs, в наше время этот движок широко используется для программирования мобильного софта. Многие знаменитые студии-разработчики используют этот движок для написания игр и графических приложений

Вычисление и визуализация проводимые в работе были выполнены на графическом процессоре AMD Radeon 7470 Mobility 1Gb.

2.2 РЕАЛИЗАЦИЯ МОДУЛЯ ВИЗУАЛИЗАЦИИ

В качестве основы для реализации алгоритма визуализации был выбран тип разработки приложения Win32 Form. При проектировании данного приложения необходимо подключение дополнительных библиотек для лучшей работы GLScene.Так как этот движок имеет открытый код то добавление новых библиотек не составляет труда, и любой программист сможет подключить к нему, удобные для себя плагины и библиотеки.

Для ввода данных в программу была выбрана система загрузки данных из отдельного файла, так как это обеспечивает быструю обработку введенных данных, основным минусом этой системы является перезагрузка программы после обновления.

Загрузка данных для программы производится из отдельного файла DATA в котором хранятся все координаты вершин и ребер графа. Загрузка данных производится путем создания функции для загрузки, как указано в Листинг 2.1.

Листинг 2.1.Алгоритм загрузки данных из файла

```
procedure TForm1.loadData(a_FileName:string);
```

```
      const
          c: array[1..2] of cardinal = ($222222, $ff2222);
      var
          s,t,u: TStringList;
          pi,li,pc,i,j,k: integer;
          p1,p2,p3: TPoint;
          v1,v2,v3: TVector3f;
          f,len: single;
          fs: TFormatSettings;
        function getpid(s:string):TPoint;
        var f:single;
        begin
          f := strtofloat(s, fs);
          result.x := round(f) - 1;
          result.y := round((f - round(f)) * 100);
        end;
begin
  if not fileexists(a_FileName) then exit;
  s := TStringList.Create;
  s.LoadFromFile(a_FileName);
  pi := s.IndexOf('[points]');
  li := s.IndexOf('[links]');
  if (pi < 0) or (li < 0) then begin
    s.Free;
    exit;
  end;
  t := TStringList.Create;
  u := TStringList.Create;
  fs.DecimalSeparator := '.';
```

Добавление новых вершин производится в файле data, указываются порядковые номера вершин, их координаты и координаты ребер. После считывания координат из файла нужно произвести расстановку этих координат, а также парсинга линий(ребер) и точек(вершин). Все эти данные вносятся в общий массив, показанный в Листинг2. 2, который и будет выводиться на экран.

Листинг 2.2. Расстановка координат на сцене проекта

```
pc := 0;
  for i := pi + 1 to li - 1 do
    if length(s[i]) > 7 then inc(pc);
  setlength(points, pc);
  p_cnt := 0;
  for i := 0 to pc - 1 do begin
    for j := 0 to high(points[i].sub) do
      if j = 0 then points[i].sub[j] := 0
        else points[i].sub[j] := 9999;
    t.Text := stringReplace(s[i + pi + 1], ' ', #13#10, [rfReplaceAll]);
    for j := 0 to t.Count - 1 do begin
      u.Text := stringReplace(t[j], ',', #13#10, [rfReplaceAll]);
      if u.Count = 3 then begin
        points[i].id := getpid(u[0]).x;
        points[i].x := strtofloat(u[1], fs);
        points[i].z := strtofloat(u[2], fs);
        inc(p_cnt);
        end
      else begin
        points[i].sub[getpid(u[0]).y] := strtofloat(u[1], fs);
        inc(p_cnt);
        end;
      end;
    end;
```

После создания массива нужно определить плоскость координат в которой будет лежать ориентированный граф. Так как точки будут лежать в разных плоскостях, необходимо создать нулевую плоскость относительно которой будет выстраиваться граф, показанных в Листинг 2.3. Использование нулевой плоскости «по умолчанию» можно объяснить, тем что в основном отрисовка графа лежит в 2D плоскости, но так как мы делаем модуль для 3D графов, координату z мы прописываем отдельно от x и y.

Листинг 2.3. Пересечение с нулевой плоскостью

```
v2 := VectorLerp(v1, v3, - v1.Y / (v3.Y - v1.Y))
          pts.Positions.Add(v2);
          pts.Colors.Add(ConvertWinColor(c[i - li - 1]));
          len := vectorDistance(v2, v3);
          lines2.Nodes.AddNode(v2);
          TGLLinesNode(lines2.Nodes.Last).Color.AsWinColor := c[i - li - 1];
          lines2.Nodes.AddNode(vectorLerp(v2, v3, 1 - f / len));
          TGLLinesNode(lines2.Nodes.Last).Color.AsWinColor := c[i - li - 1];
          len := vectorDistance(v1, v2);
          Lines1.Nodes.AddNode(vectorLerp(v1, v2, f / len));
          TGLLinesNode(lines1.Nodes.Last).Color.AsWinColor := c[i - li - 1];
          Lines1.Nodes.AddNode(v2);
          TGLLinesNode(lines1.Nodes.Last).Color.AsWinColor := c[i - li - 1];
        end
        else
        begin
          with points[p2.x] do
            setvector(v2, x, sub[p2.y], z);
```

Если вершины графа будут уходить ниже нулевой плоскости то линии (ребра), будут выделяться штрих-пунктирной линией. Также обязательно нужно указать область положительных и отрицательных значений для того,

что бы программа понимала где лежит точка и какую линию нужно выстроить , показанных в Листинг 2.4.

Листинг 2.4. Определение нахождения вершин графа

```
      if (points[p2.x].sub[p2.y] < 0) and
                   (points[p1.x].sub[p1.y] < 0) then begin
                  len := vectorDistance(v1, v2);
                  lines2.Nodes.AddNode(vectorLerp(v1, v2, f / len));
           TGLLinesNode(lines2.Nodes.Last).Color.AsWinColor := c[i - li - 1];

lines2.Nodes.AddNode(vectorLerp(v1, v2, 1 - f / len));
      TGLLinesNode(lines2.Nodes.Last).Color.AsWinColor := c[i - li - 1];
                  end
                else begin
                  len := vectorDistance(v1, v2);
                  Lines1.Nodes.AddNode(vectorLerp(v1, v2, f / len));
TGLLinesNode(lines1.Nodes.Last).Color.AsWinColor := c[i - li - 1];
 Lines1.Nodes.AddNode(vectorLerp(v1, v2, 1 - f / len));
      TGLLinesNode(lines1.Nodes.Last).Color.AsWinColor := c[i - li - 1];
                  end;
                end;
              end;
            end;
          s.Free;
          t.Free;
          u.Free;
      end;
```

При вращении сцены в 3D визуализации необходимо использовать компонент TGLCadencer, в котором указываются все параметры вращения плоскостей, а также указание основной камеры направленной на объект , показанных в Листинг 2.5.Для того что бы вращать камеру пользователю

будет необходимо нажать на левую кнопку мыши, тогда курсор зафиксируется и можно будет вращать граф в любой проекции.

Листинг 2.5. Использование TGLCadencer

```
procedure TForm1.cadProgress;
begin
  if m_turn and (vp.Camera = cam) then begin
    with mouse.CursorPos do
      cam.MoveAroundTarget(m_pos.y - y, m_pos.x - x);
    m_pos := mouse.CursorPos;
  end;
  pts.Visible := vp.Camera = cam;
  lines1.Visible := pts.Visible;
  lines2.Visible := pts.Visible;
  grid1.Visible := pts.Visible;
  grid2.Visible := pts.Visible;
end;
procedure TForm1.cam_XYCustomPerspective(const viewport: TRectangle; width,
  height, DPI: Integer; var viewPortRadius: Single);
begin
end;
```

Рендеринг сцены выполняется с использованием библиотеки OpenGL. Для этого в компонент TGLScene добавляется объект Direct OpenGL(DOGL). Объект используется для вывода на экран построенного ранее массива из загруженных данных, а также вывода индексов вершин согласно свойствам элемента WindowsBitMapFont, показанных в Листинг 2.6. Так же DOGL выстраивает линии согласно их цветов и координат точек которые они соединяют. Direct OpenGL является основным элементом GLScene для работы с графическими объектами находящихся на сцене проекта.

Листинг 2.6. Применение Direct OpenGL

```
procedure TForm1.doglRender;
var
   i,j,k: integer;
    v: TVector;
    a1: array of record
      id,sub: integer;
      sx,sy,depth: single;
      end;
 procedure shellSort;
 var
     i,j,k,n: integer;
     f: single;
 begin
   k := high(a1) div 2;
   while k > 0 do
   begin
     for i := 0 to high(a1) - k do begin
       j := i;
       while (j >= 0) and (a1[j].depth < a1[j + k].depth) do
       begin
         n := a1[j].id; a1[j].id := a1[j + k].id; a1[j + k].id := n;
        n := a1[j].sub; a1[j].sub := a1[j + k].sub; a1[j + k].sub := n;
         f := a1[j].sx; a1[j].sx := a1[j + k].sx; a1[j + k].sx := f;
         f := a1[j].sy; a1[j].sy := a1[j + k].sy; a1[j + k].sy := f;
f := a1[j].depth; a1[j].depth := a1[j + k].depth; a1[j + k].depth := f;
         if j > k then Dec(j, k)
```

```
            else j := 0;
            end;
          end;
          k := k div 2
        end;
      end;
    begin
      SetLength(a1, p_cnt);
      k := 0;
      for i := 0 to High(points) do
        for j := 0 to High(points[i].sub) do
        begin
          if points[i].sub[j] = 9999 then
            continue;
          a1[k].id := points[i].id + 1;
          a1[k].sub := j;
```

Для нормального отображения картинки нужно выставить кадрирование камеры относительно проецируемой сцены в которой находится орграф. Указать свойства отображения текста индексов вершин ,показанных в Листинг 2.7.

Листинг 2.7 . Сортировка по расстоянию до камеры

```
shellSort;
  for i := 0 to p_cnt - 1 do
    with a1[i] do begin
      sprt.Position.SetPoint(sx, sy, 0);
```

```
      sprt.Render(rci);
      txt1.Position.SetPoint(sx - 2, sy, 0);
      txt1.Text := inttostr(id);
      txt1.Render(rci);
      if sub > 0 then begin
        txt2.Position.SetPoint(sx + wbmpFont.TextWidth(txt1.Text) / 2 + 1,
          sy - wbmpFont.CharHeight / 2 + 7, 0);
        txt2.Text := inttostr(sub);
        txt2.Render(rci);
        end;
      end;
end;
```

Основным навигационным интерфейсом в программе выступают кнопки, объект Button. Для каждой кнопки прописывается определенное событие. В моем случае это переключение между камерами, на разные плоскости сцены для вывода графа в 2D в проекциях: XY,YZ,XZ. Сами кнопки располагаются на объекте Panel показанных в Листинг 2.8.

Листинг 2.8. Свойства кнопок на панели

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  vp.Camera := cam_XY;
end
procedure TForm1.Button2Click(Sender: TObject);
begin
  vp.Camera := cam_XZ
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  vp.Camera := cam_YZ;
end;
procedure TForm1.Button4Click(Sender: TObject);
begin
```

```
  vp.camera := cam;
end;
```

В результате получаем реализованный модуль визуализации ориентированного 3D графа с использованием GLScene.

Полный исходный код основных программных модулей библиотеки находится в Приложении 1.

3 ТЕСТИРОВАНИЕ МОДУЛЯ ВИЗУАЛИЗАЦИИ

Проверка работоспособности является важной и неотъемлемой частью создания программного продукта. От того, насколько досконально проведены соответствующие проверки, зависит то, как скоро программное средство может считаться готовым к использованию, будет ли необходимость впоследствии устранять ошибки.

В данной главе рассматривается методика тестирования для модуля визуализация ориентированного 3D графа. Тестирование будет проводится по методу белого ящика, то есть полностью тестировать приложение с нажатием всех кнопок и использованием всего интерфейса.

На Рис. 3.1 мы можем увидеть интерфейс программы, на нем для пользователя доступна панель с кнопками для отображения графа в разных плоскостях и основная рабочая область для вывода изображения. При запуске приложения рабочая область по умолчанию находится в состоянии 3D визуализации. Если дуга выходит на плоскости координат в отрицательное значение, то она выделяется штрих-пунктирной линией.

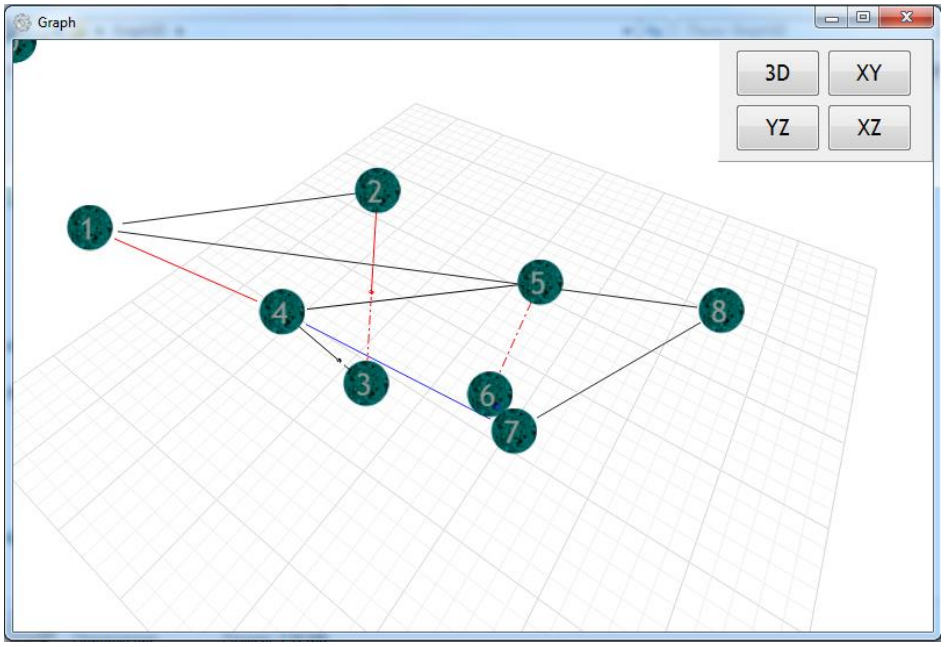

Рис. 3.1 Интерфейс программы

Ввод данных в программу осуществляется через файл данных data,открытый файл изображен на рисунке 3.2.

Вводятся координаты точек в столбец с перечислением координат, изначально они находятся в нулевой плоскости после чего нужно указать движение точки по вертикали. В файле указываются координаты линий(ребер) соединяющих вершины графа. Цвет дуги зависит от ее направления: синяя в одну сторону к следующей вершине, красная в обе стороны.

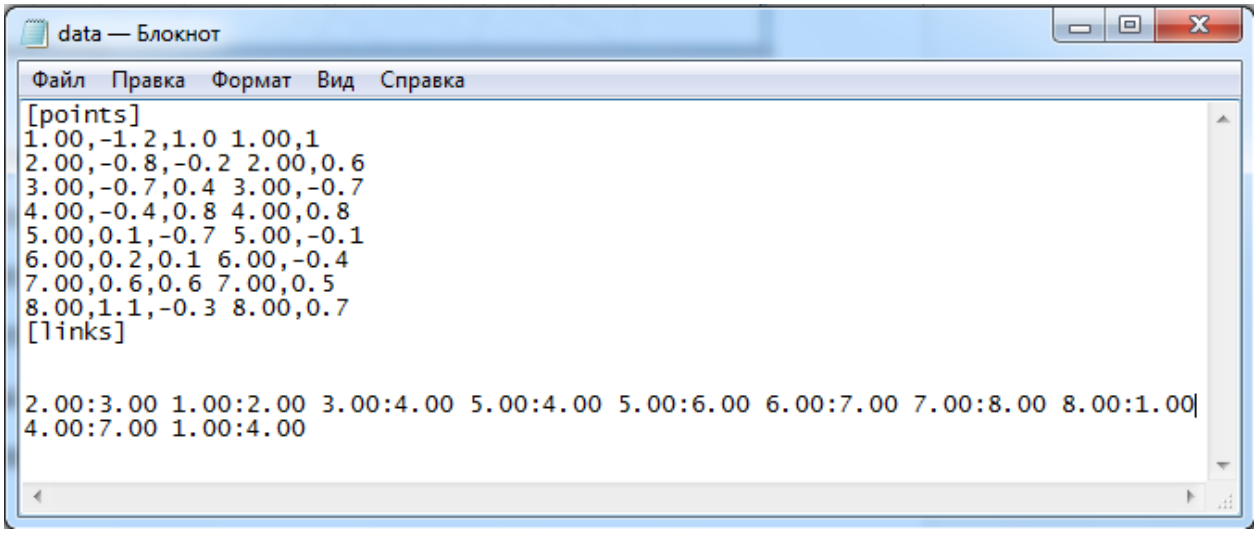

Рис. 3.2 Ввод данных для визуализации

На рисунке 3.3 изображен граф в проекции XY, для того что бы изменить проекцию изображения пользователю нужно воспользоваться панелью с кнопками для переключения проекций.

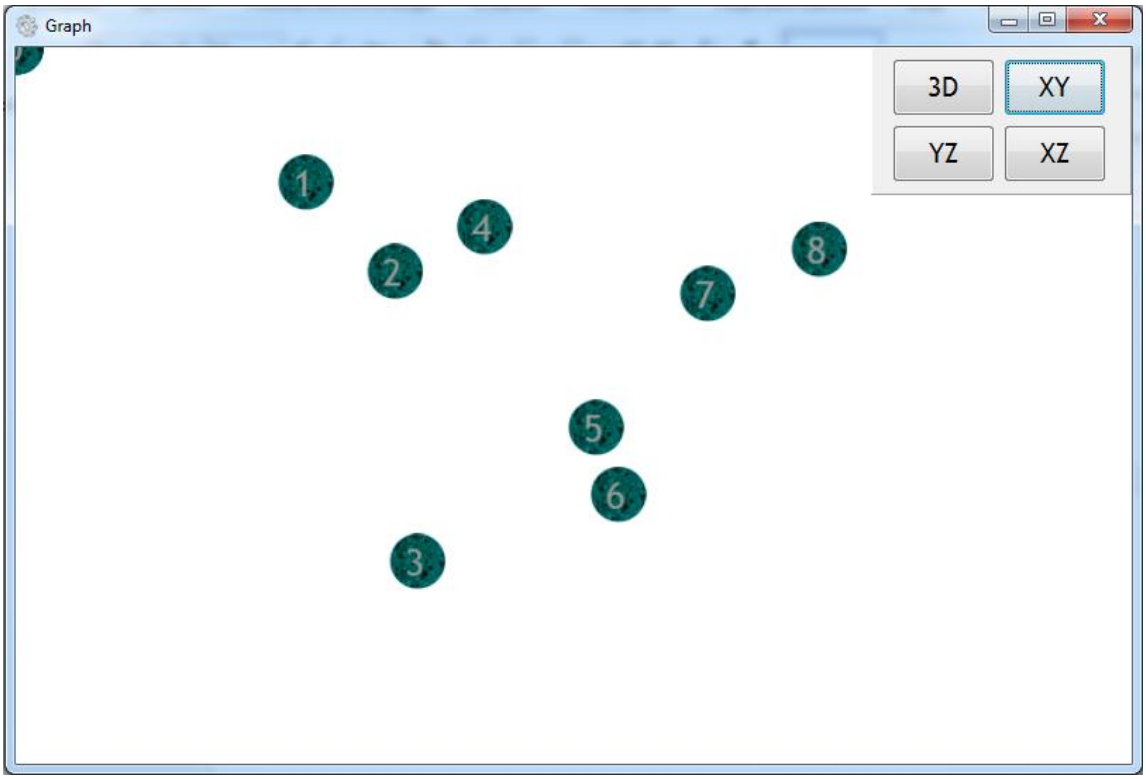

Рис. 3.3 Визуализация плоскости XY

На рисунке 3.4 изображена проекция YZ. Переключения между проекциями было создано для быстрого просмотра нарисованного графа с определенной стороны системы координат.

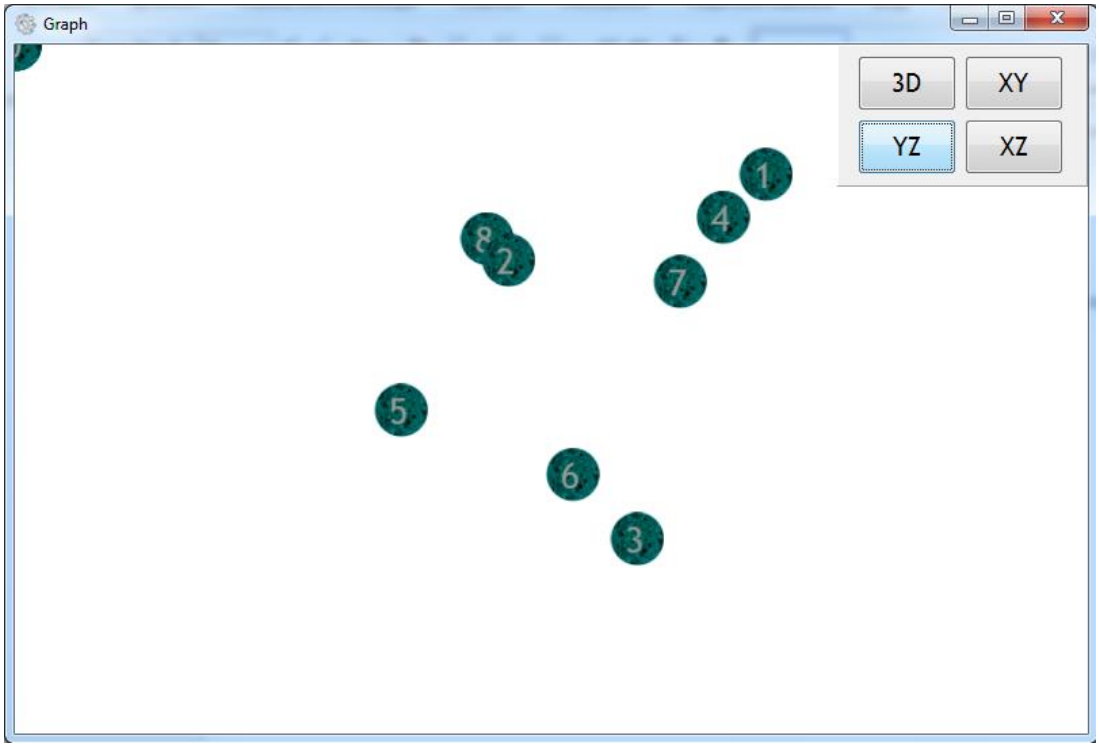

Рис. 3.4 Визуализация плоскости YZ

Основной задачей функции переключения является быстрый просмотр проекции, для оперативного редактирования графа, так как выставить проекцию с помощью мыши будет затруднительно.

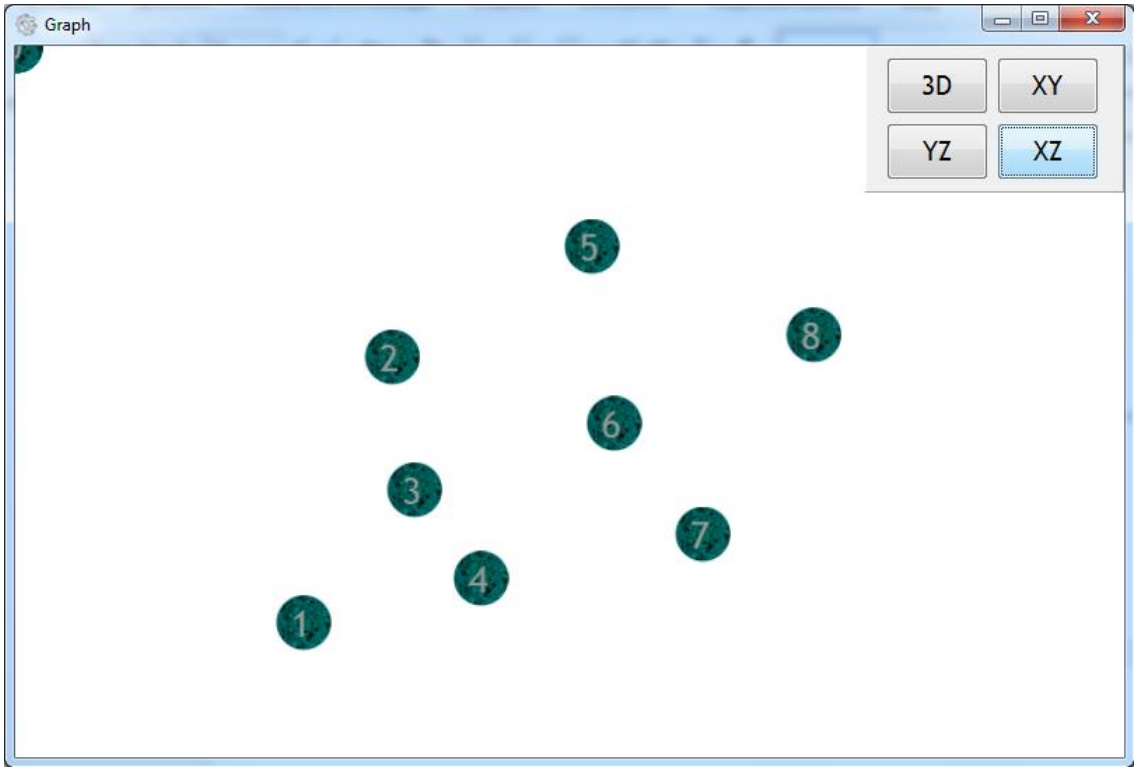

Рис. 3.5 Визуализация плоскости XZ

Добавление данных осуществляется через файл data, вводим индекс вершины и ее координаты, так как указана нулевая плоскость и по умолчанию координата Z равна нулю, в файле нужно второй раз ее проиндексировать и после этого выставить линию(ребро) от одной вершины к другой. Пример добавления данных изображен на рис. 8.

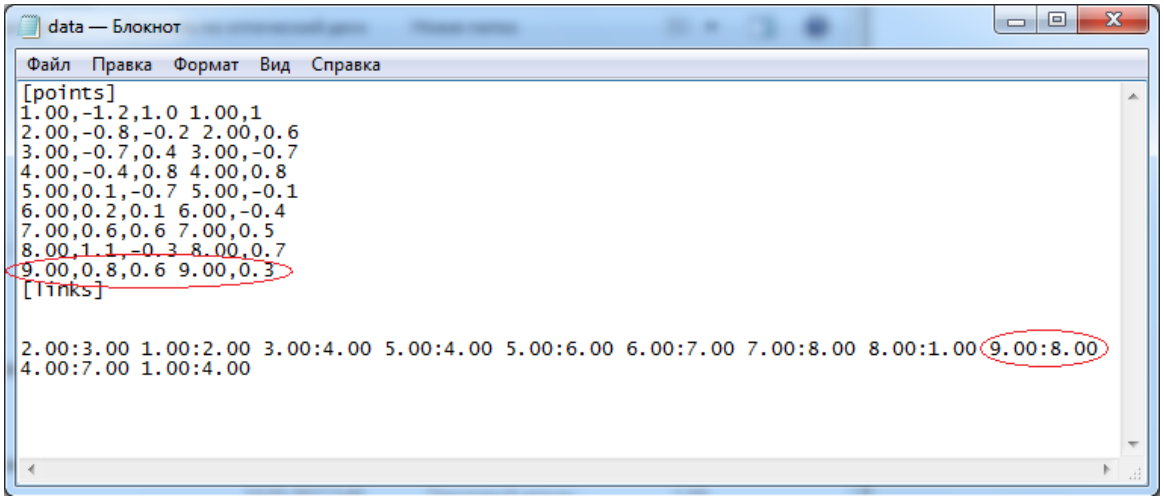

Рис. 3.6 Ввод новой координаты в программу

После того как мы ввели новую координату нам нужно сохранить текстовый файл и открыть файл Graph3D.exe. Пример выполненной программы можно увидеть на рис. 9. Добавлена вершина с индексом 9 и ребро от 8 к 9 вершинам.

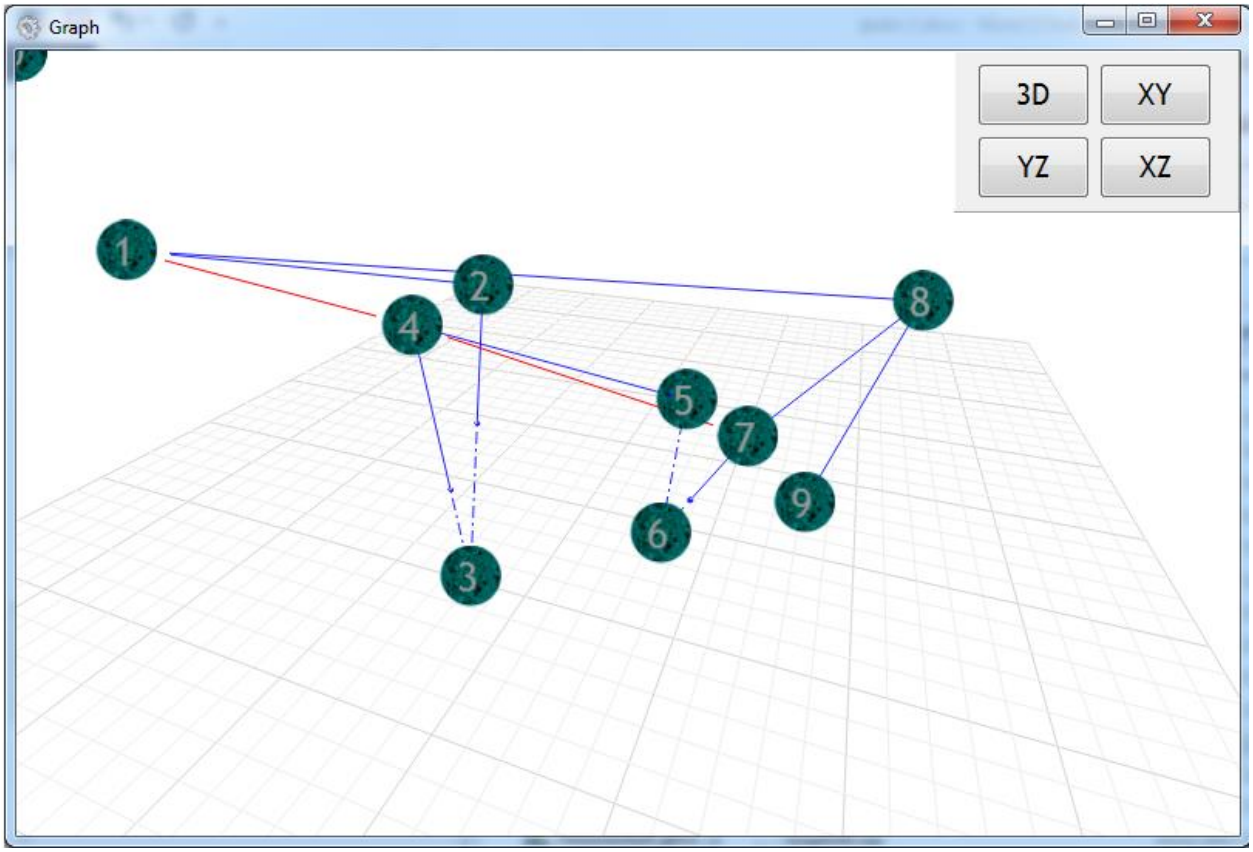

Рис. 3.7 Выполнение программы после добавления данных

Программу можно использовать для визуализации любого графа, так как связи между вершинами расставляются самим пользователем, так же как и вершины. Количество точек и связей между ними не ограничены. Пользователь может изменить текстуру вершин, отредактировав файл bubble.tga, в любом графическом редакторе. Разработанное приложение можно считать гибким и универсальным для большого количества пользователей.

ЗАКЛЮЧЕНИЕ

В настоящей дипломной работе была рассмотрена возможность визуализации ориентированного 3D графа, в результате чего было разработано соответствующее программное средство, представленное в форме приложения Win32. Особенностью реализации является использование графического движка GLScene. Поэтому, основываясь на полученных результатах, можно утверждать, что поставленная цель исследования достигнута.

Для разработки модуля визуализации применялась технология OpenGL. Были изучены методические и технические материалы о среде программирования.

В целом разработанный модуль может считаться надежным инструментом для визуализации графов и быть встроенным в более модифицированное программное средство.

Основная цель по изучению теории графов и применению практических навыков программирования для написания модуля, достигнута в полном объеме

Задача по разработке модуля визуализации ориентированного 3D графа с использованием GLScene, выполнена.

СПИСОК ИСЛЬЗОВАЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дж. Ли, Б. Уэр. Трёхмерная графика и анимация. — 2-е изд. — М.: Вильямс, 2002. — 640 с.
2. Кристофидес Н. Теория графов. Алгоритмический подход. Пер. с анг. - М.:Мир, 1978, 432 с.
3. Логиновский, А.Н. Инженерная 3D-компьютерная графика: Учебное пособие для бакалавров / А.Н. Логиновский. - М.: Юрайт, 2013. - 464 с.
4. МакКоннелл Дж. Основы современных алгоритмов. — М.: Техносфера, Москва «Мир» 1989— 368 с
5. Миронов, Д.Ф. Компьютерная графика в дизайне: Учебник / Д.Ф. Миронов. - СПб.: БХВ-Петербург, 2008. - 560 с.
6. Немцова, Т.И. Практикум по информатике. Компьютерная графика и Web-дизайн. Практикум: Учебное пособие / Т.И. Немцова. - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2013. - 288 с.
7. Никулин Е. А. Компьютерная геометрия и алгоритмы машинной графики. — СПб: БХВ-Петербург, 2003. — 560 с.
8. Ойстин, Оре. Теория графов. — М.: УРСС, 2008. — 352 с
9. Пантюхин, П.Я. Компьютерная графика. В 2-х т.Т. 1. Компьютерная графика: Учебное пособие / П.Я. Пантюхин. - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2012. - 88 с.
10. Свами М., Тхуласираман К. Графы, сети и алгоритмы: Пер. с англ. - М.:Мир, 1984, 456 с.
11. Татт У. Теория графов. Пер. с англ. - М.:Мир, 1988, 424 с.
12. Тозик, В.Т. Компьютерная графика и дизайн: Учебник для нач. проф. образования / В.Т. Тозик, Л.М. Корпан. - М.: ИЦ Академия, 2013. - 208 с.

13. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
14. Харари Ф. Теория графов. — М.: УРСС, 2003. — 300 с.
15. Херн, Д., Бейкер М.П. Компьютерная графика и стандарт OpenGL. — 3-е изд. — М., 2005. — 1168 с.
16. Шикин Е. В., А. В. Боресков, А. А. Зайцев «Начала компьютерной графики» — Москва «Диалог - МИФИ» 1993, —217 с.
17. Э. Энджел. Интерактивная компьютерная графика. Вводный курс на базе OpenGL. — 2-е изд. — М.: Вильямс, 2001. — 592 с.

ПРИЛОЖЕНИЕ 1

ЛИСТИНГ ПРОГРАММНОГО КОДА

```
unit u_Main;

interface

uses
  Winapi.Windows,
  Winapi.Messages,
  System.SysUtils,
  System.Variants,
  System.Classes,
  System.Math,
  Vcl.Graphics,
  Vcl.Controls,
  Vcl.Forms,
  Vcl.Dialogs,
  Vcl.StdCtrls,
  Vcl.ComCtrls,
  Vcl.ToolWin,
  Vcl.ExtCtrls,
  //GLS
  GLScene,
  GLObjects,
  GLCoordinates, GLWin32Viewer, GLCrossPlatform,
  GLNodes,
  GLBaseClasses, GLHUDObjects,
  GLBitmapFont, GLWindowsFont, GLRenderContextInfo,
  GLVectorGeometry, GLVectorTypes,
  GLCadencer, GLColor,
```

```
  GLFileTGA,
  GLGraph;


type
  TForm1 = class(TForm)
    Memo1: TMemo;
    GLScene1: TGLScene;
    vp: TGLSceneViewer;
    cam: TGLCamera;
    dc_world: TGLDummyCube;
    dogl: TGLDirectOpenGL;
    sprt: TGLHUDSprite;
    txt1: TGLHUDText;
    wbmpFont: TGLWindowsBitmapFont;
    cad: TGLCadencer;
    Lines1: TGLLines;
    Lines2: TGLLines;
    Panel1: TPanel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    grid1: TGLXYZGrid;
    grid2: TGLXYZGrid;
    txt2: TGLHUDText;
    pts: TGLPoints;
    cam_XY: TGLCamera;
    cam_XZ: TGLCamera;
    cam_YZ: TGLCamera;
```

```
    procedure FormCreate(Sender: TObject);
    procedure doglRender(Sender: TObject; var rci: TGLRenderContextInfo);
    procedure vpMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure vpMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure cadProgress(Sender: TObject; const deltaTime,
      newTime: Double);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure cam_XYCustomPerspective(const viewport: TRectangle; width,
height,
      DPI: Integer; var viewPortRadius: Single);

  public
    procedure loadData(a_FileName:string);
  end;



type
  t_point = record
    x,z: single;
    id: integer;
    sub: array[0..9] of single;
  end;



var
```

```
  Form1: TForm1;


  points: array of t_point;
  p_cnt: integer;
  m_turn: boolean = false;
  m_pos: TPoint;


implementation

{$R *.dfm}


procedure TForm1.loadData(a_FileName:string);
const
    c: array[1..4] of cardinal = ($222222, $ff2222, $0000ff, $00ff00);

var
    s,t,u: TStringList;
    pi,li,pc,i,j,k: integer;
    p1,p2,p3: TPoint;
    v1,v2,v3: TVector3f;
    f,len: single;
    fs: TFormatSettings;

  function getpid(s:string):TPoint;
  var f:single;
  begin
    f := strtofloat(s, fs);
```

```
  result.x := round(f) - 1;
  result.y := round((f - round(f)) * 100);
 end;

begin
 if not fileexists(a_FileName) then exit;

 s := TStringList.Create;
 s.LoadFromFile(a_FileName);

 pi := s.IndexOf('[points]');
 li := s.IndexOf('[links]');

 if (pi < 0) or (li < 0) then begin
  s.Free;
  exit;
 end;

 t := TStringList.Create;
 u := TStringList.Create;
 fs.DecimalSeparator := '.';



 pc := 0;
 for i := pi + 1 to li - 1 do
  if length(s[i]) > 7 then inc(pc);

 setlength(points, pc);
 p_cnt := 0;
```

```
for i := 0 to pc - 1 do begin

  for j := 0 to high(points[i].sub) do
    if j = 0 then points[i].sub[j] := 0
      else points[i].sub[j] := 9999;

  t.Text := stringReplace(s[i + pi + 1], ' ', #13#10, [rfReplaceAll]);



  for j := 0 to t.Count - 1 do begin

    u.Text := stringReplace(t[j], ',', #13#10, [rfReplaceAll]);

    if u.Count = 3 then begin
      points[i].id := getpid(u[0]).x;
      points[i].x := strtofloat(u[1], fs);
      points[i].z := strtofloat(u[2], fs);
      inc(p_cnt);
      end
    else begin
      points[i].sub[getpid(u[0]).y] := strtofloat(u[1], fs);
      inc(p_cnt);
      end;
    end;
  end;



Lines1.Nodes.Clear;
```

```
Lines2.Nodes.Clear;
pts.Positions.Clear;
pts.Colors.Clear;

f := 0.11;

for i := li + 1 to min(s.Count - 1, li + 5) do
begin
  t.Text := stringReplace(s[i], ' ', #13#10, [rfReplaceAll]);
  for j := 0 to t.Count - 1 do
    if length(t[j]) = 9 then begin

      u.Text := stringReplace(t[j], ':', #13#10, [rfReplaceAll]);

      p1 := getpid(u[0]);
      p2 := getpid(u[1]);

      if (points[p1.x].sub[p1.y] = 9999) or
         (points[p2.x].sub[p2.y] = 9999) then continue;

      if points[p2.x].sub[p2.y] > points[p1.x].sub[p1.y] then
      begin
        p3 := p2;
        p2 := p1;
        p1 := p3;
      end;

      with points[p1.x] do
        setvector(v1, x, sub[p1.y], z);
```

```
if (points[p2.x].sub[p2.y] < 0) and
   (points[p1.x].sub[p1.y] >= 0) then
begin
  with points[p2.x] do
    setvector(v3, x, sub[p2.y], z);


  v2 := VectorLerp(v1, v3, - v1.Y / (v3.Y - v1.Y));

  pts.Positions.Add(v2);
  pts.Colors.Add(ConvertWinColor(c[i - li - 1]));

  len := vectorDistance(v2, v3);

  lines2.Nodes.AddNode(v2);
  TGLLinesNode(lines2.Nodes.Last).Color.AsWinColor := c[i - li - 1];

  lines2.Nodes.AddNode(vectorLerp(v2, v3, 1 - f / len));
  TGLLinesNode(lines2.Nodes.Last).Color.AsWinColor := c[i - li - 1];

  len := vectorDistance(v1, v2);

  Lines1.Nodes.AddNode(vectorLerp(v1, v2, f / len));
  TGLLinesNode(lines1.Nodes.Last).Color.AsWinColor := c[i - li - 1];

  Lines1.Nodes.AddNode(v2);
  TGLLinesNode(lines1.Nodes.Last).Color.AsWinColor := c[i - li - 1];
end
else
```

```
    begin
      with points[p2.x] do
        setvector(v2, x, sub[p2.y], z);


      if (points[p2.x].sub[p2.y] < 0) and
        (points[p1.x].sub[p1.y] < 0) then begin

        len := vectorDistance(v1, v2);

        lines2.Nodes.AddNode(vectorLerp(v1, v2, f / len));
        TGLLinesNode(lines2.Nodes.Last).Color.AsWinColor := c[i - li - 1];

        lines2.Nodes.AddNode(vectorLerp(v1, v2, 1 - f / len));
        TGLLinesNode(lines2.Nodes.Last).Color.AsWinColor := c[i - li - 1];

        end

      else begin
        len := vectorDistance(v1, v2);
        Lines1.Nodes.AddNode(vectorLerp(v1, v2, f / len));

        TGLLinesNode(lines1.Nodes.Last).Color.AsWinColor := c[i - li - 1];

        Lines1.Nodes.AddNode(vectorLerp(v1, v2, 1 - f / len));
        TGLLinesNode(lines1.Nodes.Last).Color.AsWinColor := c[i - li - 1];
        end;
      end;
    end;
end;
```

```
  s.Free;
  t.Free;
  u.Free;
end;



procedure TForm1.FormCreate;
begin
  loadData('data');
end;


procedure TForm1.vpMouseDown;
begin
  if shift = [ssleft] then begin
    m_turn := true;
    m_pos := mouse.CursorPos;
  end;
end;



procedure TForm1.vpMouseUp;
begin
  m_turn := false;
end;
```

```
procedure TForm1.cadProgress;
begin
  if m_turn and (vp.Camera = cam) then begin
    with mouse.CursorPos do
      cam.MoveAroundTarget(m_pos.y - y, m_pos.x - x);

    m_pos := mouse.CursorPos;
  end;
  pts.Visible := vp.Camera = cam;

  lines1.Visible := pts.Visible;
  lines2.Visible := pts.Visible;

  grid1.Visible := pts.Visible;
  grid2.Visible := pts.Visible;
end;


procedure TForm1.cam_XYCustomPerspective(const viewport: TRectangle;
width,
  height, DPI: Integer; var viewPortRadius: Single);
begin

end;


procedure TForm1.doglRender;
var
    i,j,k: integer;
    v: TVector;
```

```
  a1: array of record
    id,sub: integer;
    sx,sy,depth: single;
    end;

procedure shellSort;
var
   i,j,k,n: integer;
   f: single;
begin
 k := high(a1) div 2;
 while k > 0 do
 begin
   for i := 0 to high(a1) - k do begin
     j := i;
     while (j >= 0) and (a1[j].depth < a1[j + k].depth) do
     begin
       n := a1[j].id; a1[j].id := a1[j + k].id; a1[j + k].id := n;
       n := a1[j].sub; a1[j].sub := a1[j + k].sub; a1[j + k].sub := n;
       f := a1[j].sx; a1[j].sx := a1[j + k].sx; a1[j + k].sx := f;
       f := a1[j].sy; a1[j].sy := a1[j + k].sy; a1[j + k].sy := f;
       f := a1[j].depth; a1[j].depth := a1[j + k].depth; a1[j + k].depth := f;
       if j > k then Dec(j, k)
       else j := 0;
       end;
     end;
     k := k div 2
   end;
 end;
```

```
begin
  SetLength(a1, p_cnt);
  k := 0;
  for i := 0 to High(points) do
    for j := 0 to High(points[i].sub) do
    begin
      if points[i].sub[j] = 9999 then
        continue;

      a1[k].id := points[i].id + 1;
      a1[k].sub := j;


      with points[i] do
        setVector(v, x, sub[j], z);
      a1[k].depth := vectorNorm(vectorSubtract(v, cam.AbsolutePosition));


      v := vp.Buffer.WorldToScreen(v);
      a1[k].sx := v.X;
      a1[k].sy := vp.Height - v.Y;
      inc(k);
    end;


  shellSort;
  for i := 0 to p_cnt - 1 do
    with a1[i] do begin
      sprt.Position.SetPoint(sx, sy, 0);
      sprt.Render(rci);
```

```
      txt1.Position.SetPoint(sx - 2, sy, 0);
      txt1.Text := inttostr(id);
      txt1.Render(rci);

      if sub > 0 then begin
        txt2.Position.SetPoint(sx + wbmpFont.TextWidth(txt1.Text) / 2 + 1,
          sy - wbmpFont.CharHeight / 2 + 7, 0);
        txt2.Text := inttostr(sub);
        txt2.Render(rci);
        end;
      end;
end;


procedure TForm1.Button1Click(Sender: TObject);
begin
  vp.Camera := cam_XY;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  vp.Camera := cam_XZ
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  vp.Camera := cam_YZ;
end;

procedure TForm1.Button4Click(Sender: TObject);
```

```
begin
  vp.camera := cam;
end;

end.
```

Выпускная квалификационная работа выполнена мной самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«___» _____ _____ г.

_____ _____

(подпись) *(Ф.И.О.)*