Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционные системы и системное программирование (ОСиСП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

ПРОГРАММНОЕ СРЕДСТВО «МОЛЕКУЛЯРНЫЙ РЕДАКТОР»

БГУИР КР 1-40 01 01 204 ПЗ

Студент: гр. 951002 Будович И.В.

Руководитель: Степанцов Е.В.

Минск 2021

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2021 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Будовичу Ивану Витальевичу

1. Тема работы Программное средство «Молекулярный редактор»

2. Срок сдачи студентом законченной работы––20.12.2021 г.–––

3. Исходные данные к работе язык программирования C++

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Анализ прототипов, литературных источников и моделирование предметной области;

2. Анализ требований к программному средству и разработка функциональных требований;

3. Проектирование программного средства;

4. Создание (конструирование) программного средства;

5. Тестирование, проверка работоспособности и анализ полученных результатов;

6. Руководство по установке и использованию;

Список используемой литературы

Заключение

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Программное средство «Молекулярный редактор», А1, схема программы, чертеж.

6. Консультант по курсовой работе

Степанцов Е.В.

7. Дата выдачи задания 06.09.2021

8. Календарный график работы над курсовой работой на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 15.09.2021 – 15 % готовности работы;

разделы 2, 3 к 15.10.2021 – 30 % готовности работы;

разделы 4, 5 к 15.11.2021 – 60 % готовности работы;

раздел 6 к 15.12.2021 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 17.12.2021 – 100 % готовности работы.

Защита курсовой работы с 13.12.2021 по 20.12.2021 г.––––––––––––––––––––

РУКОВОДИТЕЛЬ–––––– Степанцов Е.В.

(подпись)

Задание принял к исполнению 06.09.2021–––\_\_\_\_\_\_––

(дата и подпись студента)

Содержание

[Содержание 4](#_Toc90161781)

[Введение 6](#_Toc90161782)

[1 Анализ прототипов, литературных источников и моделирование предметной области 7](#_Toc90161783)

[1.1 Молекулярные вещества 7](#_Toc90161784)

[1.2 Способы представления молекулярных веществ 7](#_Toc90161785)

[1.3 Цветовое представление атомов в трехмерных моделях 10](#_Toc90161786)

[1.4 Анализ прототипов 11](#_Toc90161787)

[1.5 Постановка задачи 14](#_Toc90161788)

[2 Анализ требований к программному средству и разработка функциональных требований 15](#_Toc90161789)

[2.1 Описание функциональных требований 15](#_Toc90161790)

[2.2 Описание прочих требований 15](#_Toc90161791)

[3 Проектирование программного средства 17](#_Toc90161792)

[3.1 Разработка алгоритма рисования скелетных формул 17](#_Toc90161793)

[3.2 Разработка алгоритма преобразования двумерного представления в трехмерное 17](#_Toc90161794)

[3.3 Разработка алгоритма рисования полусферической модели 18](#_Toc90161795)

[4 Конструирование программного средства 19](#_Toc90161796)

[4.1 Проектирование интерфейса программы 19](#_Toc90161797)

[4.2 Структура модулей программы 19](#_Toc90161798)

[4.3 Описание класса Win32Application 20](#_Toc90161799)

[4.4 Описание класса SkeletonCanvas 21](#_Toc90161800)

[4.5 Описание класса Toolbar 24](#_Toc90161801)

[4.6 Описание класса CalotteCanvas 24](#_Toc90161802)

[4.7 Описание класса Converter 27](#_Toc90161803)

[4.8 Описание класса FileManager 29](#_Toc90161804)

[4.9 Описание структур 29](#_Toc90161805)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 33](#_Toc90161806)

[5.1 Тестирование алгоритма рисования 33](#_Toc90161807)

[5.2 Тестирование сохранения изображения 34](#_Toc90161808)

[5.3 Тестирование трехмерного моделирования 34](#_Toc90161809)

[6 Руководство по установке и использованию 37](#_Toc90161810)

[Заключение 40](#_Toc90161811)

[Список литературы 41](#_Toc90161812)

[Приложение A 42](#_Toc90161813)

[Приложение Б 43](#_Toc90161814)

[Приложение В 46](#_Toc90161815)

[Листинг файла main.cpp 46](#_Toc90161816)

[Листинг файла Win32Application.h 46](#_Toc90161817)

[Листинг файла Win32Application.cpp 46](#_Toc90161818)

[Листинг файла SkeletonCanvas.h 47](#_Toc90161819)

[Листинг файла SkeletonCanvas.cpp 48](#_Toc90161820)

[Листинг файла Toolbar.h 55](#_Toc90161821)

[Листинг файла Toolbar.cpp 55](#_Toc90161822)

[Листинг файла CalotteCanvas.h 57](#_Toc90161823)

[Листинг файла CalotteCanvas.cpp 58](#_Toc90161824)

[Листинг файла Converter.h 61](#_Toc90161825)

[Листинг файла Converter.cpp 62](#_Toc90161826)

[Листинг файла FileManager.h 74](#_Toc90161827)

[Листинг файла FileManager.cpp 75](#_Toc90161828)

[Листинг файла State.h 77](#_Toc90161829)

[Листинг файла State.cpp 77](#_Toc90161830)

[Листинг файла Atom.h 77](#_Toc90161831)

[Листинг файла AtomColor.h 78](#_Toc90161832)

[Листинг файла Point.h 78](#_Toc90161833)

[Листинг файла Vertex2D.h 78](#_Toc90161834)

[Листинг файла Vertex3D 79](#_Toc90161835)

[Листинг файла Element.h 79](#_Toc90161836)

[Листинг файла ConverterNode.h 79](#_Toc90161837)

Введение

Для исследования естественнонаучных дисциплин используются прикладные пакеты программного обеспечения. Они помогают визуализировать информацию, используя общепринятые соглашения, а также научные данные.

Молекулярные вещества – это обширный класс химических веществ. К веществам молекулярного относится большинство веществ, не имеющих в своем составе атомов металлов.

Широким классом веществ молекулярного строения являются органические вещества. Известно 27 миллионов веществ данного класса, что подтверждает их важность и необходимость использования ПО для их изучения и систематизации.

Цель данной курсовой работы – создать молекулярный редактор, способный работать с органическими веществами и другими низкомолекулярными соединениями.

В ходе выполнения курсовой работы я постараюсь найти решение таким техническим вопросам как:

1. Представление химических веществ в виде структур данных;
2. Использование API, работающих с трехмерной графикой.
3. Работа с парадигмой ООП.

В этой пояснительной записке отображены следующие этапы написания курсовой работы:

1. Анализ прототипов, литературных источников и моделирование предметной области;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.
7. Анализ прототипов, литературных источников и моделирование предметной области
   1. Молекулярные вещества

В классической теории химического строения молекула рассматривается как наименьшая стабильная частица вещества, обладающая всеми его химическими свойствами. К молекулам относятся такие известные вещества, как вода, кислород, аммиак, углекислый газ, а также органические вещества, например, метан или этиловый спирт. Органические вещества — соединения углерода с кислородом, водородом, азотом и друг. элементами, встречающиеся в животном и растительном царстве, а также полученные синтетически.

Химические связи в молекулах могут быть одинарными, двойными и тройными, в зависимости от этого определяется их угол.

Некоторые химические вещества, например, органические, сложно представить в виде формулы, так как существует большое количество изомеров, имеющих одинаковый количественный состав. Например, формула может соответствовать n-пентану, 2-метилбутану и 2,2-диметилпропану. В этих целях в химии широкое распространение получили графические способы представления веществ.

* 1. Способы представления молекулярных веществ
     1. Скелетная структурная формула

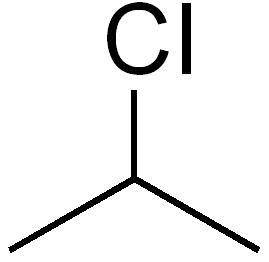


Рисунок 1.1 – представление 2-хлорпропана в виде скелетной структурной формулы

В скелетной структурной формуле длинными чертами представляется связь C-C (основная связь в органической химии), при этом стоящие при атомах углерода атомы водорода опускаются, так как их число легко вычислить исходя из числа связей атома углерода с другими атомами. Связи с атомами других химических элементов обозначаются более короткими чертами. Атомы углерода обозначаются вершинами ломанной, других элементов – соответствующим обозначением в периодической таблице (рис. 1.1).

В соответствии с соглашениями, связи с атомами с и гибридизациями, т.е одинарные и двойные, обозначаются звеньями ломанной, связанными под углом 120 градусов. Связи с гибридизацией (тройные, а также две идущие подряд двойные) обозначаются под углом 180 градусов [1].

Преимущества:

* Удобна для ручного и компьютерного ввода.
* Обладает высокой читабельностью для низкомолекулярных соединений.

Недостатки:

* Не отображает действительную трехмерную структуру.
  + 1. Стержневая модель

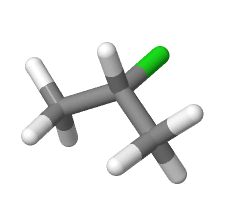


Рисунок 1.2 – представление 2-хлорпропана в виде стержневой модели

Стержневая модель используется редко. Связи в молекуле представлены цилиндрами, со скругленными основаниями, где половина связи обозначена цветом, соответствующим химическому элементу атома (рис. 1.2).

Преимущества:

* Отображает трехмерную структуру.

Недостатки:

* Не реалистична, так как показывает связи, а не атомы.
* Не дает представления о размерах атомов.
  + 1. Шаростержневая модель

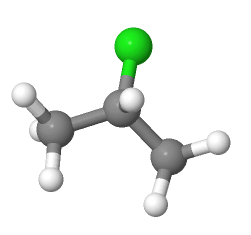


Рисунок 1.3 – представление 2-хлорпропана в виде шаростержневой модели

В шаростержневой модели атомы представлены сферами разных цветов, радиусы которых пропорциональны радиусам Ван-дер-Ваальса для данного химического элемента [2], связи представлены аналогично стержневой модели (рис 1.3).

Преимущества:

* Отображает трехмерную структуру.
* Отображает соотношения радиусов атомов.
* Может быть собрана вручную в реальной жизни.

Недостатки:

* Искажает понятие связи, так как связи являются лишь электронными парами.
  + 1. Полусферическая модель

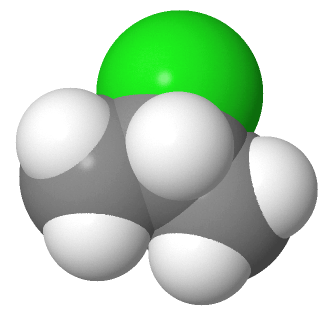


Рисунок 1.4 – представление 2-хлорпропана в виде полусферической модели

Полусферическая модель – трехмерная модель, в которой атомы представлены сферами, радиусы которых пропорциональны радиусам Ван-дер-Ваальса, а расстояния между сферами пропорциональны расстояниям между атомными ядрами (рис. 1.4) [2].

Преимущества:

* Отображает трехмерную структуру.
* Отображает соотношения радиусов атомов и длин связей.
* Наиболее реалистичное представление.

Недостатки:

* Для относительно небольших молекул уже плохо читаема.
* Не дает представления о кратности связи (за исключением меры угла).
  1. Цветовое представление атомов в трехмерных моделях

В химии для обозначения атомов различных химических элементов в молекулярных моделях используется цветовая схема моделей Кори – Полинга – Колтуна (CPK) [3]. Похожая цветовая схема используется Java-приложением Jmol [4].

На таблице 1.1 представлена цветовая схема для наиболее часто встречаемых химических элементов. Исходя из таблицы, видно, что представление Jmol более предпочтительно, так как галогенный ряд представлен более широким спектром цветов.

Таблица 1.1 – цветовая схема отдельных химических элементов

|  |  |  |
| --- | --- | --- |
| Химический элемент | Представление CPK | Представление Jmol |
| Углерод | Черный | Серый |
| Водород | Белый | Белый |
| Кислород | Красный | Красный |
| Азот | Синий | Синий |
| Сера | Желтый | Желтый |
| Фосфор | Фиолетовый | Оранжевый |
| Фтор | Светло-зеленый | Светло-зеленый |
| Хлор | Светло-зеленый | Светло-зеленый |
| Бром | Зеленый | Красно-кирпичный |
| Йод | Темно-зеленый | Фиолетовый |

* 1. Анализ прототипов

Данная курсовая работа ориентирована на создание молекулярного редактора. Для того, чтобы иметь представление о функционале молекулярного редактора, рассмотрим уже существующие в качестве прототипов.

* + 1. Приложение JMol

JMol — программа для просмотра структуры молекул в трёх измерениях. Написана на языке программирования Java [5]. Пример интерфейса представлен на рисунке 1.5.

Преимущества:

* Поддержка макросов и скриптов.
* Возможность сохранения в разные форматы.
* Возможность использования в качестве апплета в веб-странице.

Недостатки:

* Относительно неудобный интерфейс.
* Низкое качество изображения.
* Отсутствие поддержки двухмерных скелетных форм в основной версии.

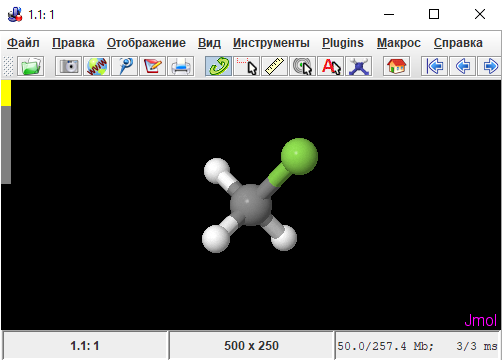


Рисунок 1.5 – интерфейс молекулярного редактора JMol

* + 1. PyMOL

PyMOL – многофункциональная программа для визуализации молекул. Написан на языке Python [6]. Позволяет создавать высококачественные трёхмерные изображения как малых молекул, так и биологических макромолекул, в первую очередь белков. Широко используется в научной литературе. Интерфейс программы представлен на рисунке 1.6.

**Преимущества:**

* Позволяет работать с высокомолекулярными соединениями.
* Позволяет использовать разные модели представления.

**Недостатки:**

* Ориентирован, в основном, на высокомолекулярные органические соединения.
* Отсутствует двумерное представление.

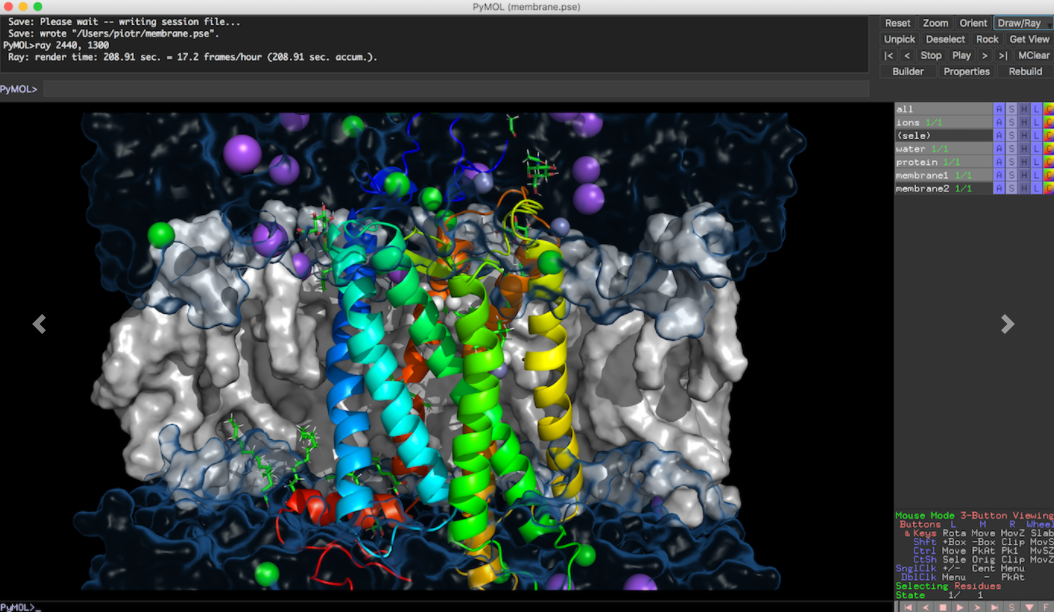


Рисунок 1.6 – интерфейс молекулярного редактора PyMOL

* + 1. Приложение BKChem

BKChem - бесплатный редактор 2D-молекул, написанный на Python 2.5 [7]. Интерфейс программы представлен на рисунке 1.7.

**Преимущества:**

* Удобен в создании скелетных формул.

**Недостатки:**

* Отсутствует трехмерное представление.
* Поддержка приложения прекращена.

Вывод: в ходе работы следует создать программное средство, способное одновременно работать с двух- и трехмерным представлениями молекул.

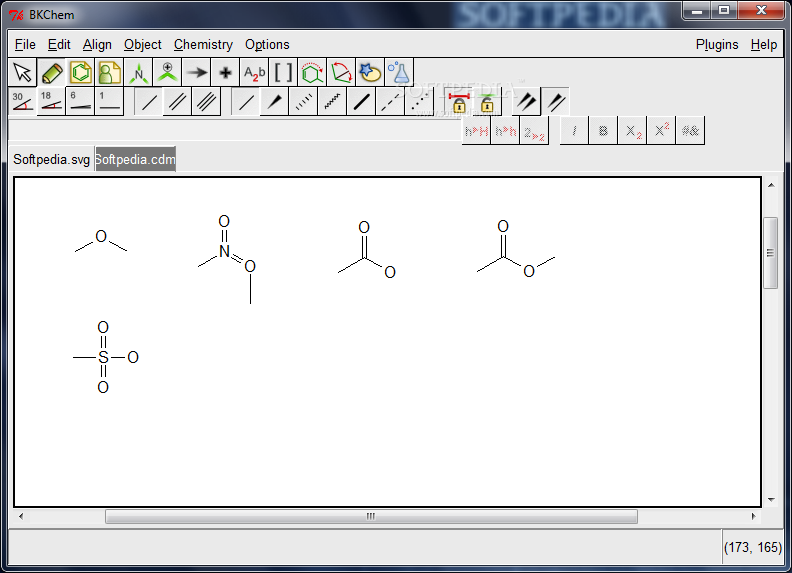


Рисунок 1.7 – интерфейс молекулярного редактора BKChem

* 1. Постановка задачи

Программное средство должно представлять собой редактор низкомолекулярных веществ. Программа должна обеспечивать выполнение перечисленных ниже функций:

* + Работа с двухмерным (скелетная формула) и трехмерным (полусферическая модель) представлениями молекул;
  + возможность выбора основных химических элементов;
  + возможность сохранения полученных изображений.

Пользователь может выбирать химических элемент и необходимую кратность связи и рисовать с помощью компьютерной мыши скелетные формулы. Выходными данными являются полученные изображения.

Пользователь может преобразовать полученную формулу в полусферическую модель, поддерживающую масштабирование.

Для разработки я выбрал среду программирования Microsoft Visual Studio 2019. Данная среда позволяет разрабатывать Win32 приложения с помощью WinAPI, а также обеспечивать взаимодействие с другими библиотеками, такими как OpenGL. В качестве языка программирования выбран C++ в связи с поддержкой парадигмы ООП, наличием стандартных шаблонов STL для упрощенной работы с динамическими массивами и строковым типом данных.

1. Анализ требований к программному средству и разработка функциональных требований
   1. Описание функциональных требований

Программное средство для функциональности должно работать с двухмерной и трехмерной графикой. Должен существовать алгоритм преобразования скелетной формулы в полусферическую модель.

Основными функциями ПС являются создание двухмерного изображения и преобразование в трехмерное, сохранение двухмерного и трехмерного изображений.

Для моего программного средства я определил следующие функциональные требования:

* 1. Наличие в левой части окна программы панели инструментов, на которой можно выбрать химический элемент, кратность связи;
  2. В верхней части окна находится меню, где можно очистить холст, сохранить изображение.
  3. Скелетные формулы рисуются нажатием левой кнопки мыши по одной из вершин формулы или по произвольной части предназначенного для этого экрана при отсутствии изображения;
  4. При нажатии на сохранение изображения отображается диалоговое окно с выбором каталога, имени и расширения файла. К доступным расширениям относится BMP;
  5. В верхнем меню присутствует кнопка, преобразующая скелетную формулу в полусферическую;
  6. Полусферическая модель может быть масштабирована посредством прокрутки колеса мыши.
  7. **Описание прочих требований**

При выполнении работы определяются дополнительные требования:

* К основным химическим элементам относятся: углерод, кислород, азот, фосфор, сера, фтор, хлор, бром, йод. Водород нельзя выбрать на панели инструментов, свободные связи других атомов заполняются атомами водорода.
* Углерод принимает валентность IV, кислород и сера– II, азот и фосфор – III. Остальные принимают валентность – I.
* Вещество может иметь связь, кратность которого не превышает валентность или число 3.
* Интерфейс программного средства должен быть простым и удобным.
* Программное средство должно работать на устройстве под управлением операционной системы Windows.

1. Проектирование программного средства
   1. Разработка алгоритма рисования скелетных формул
      1. Алгоритм добавления двумерных координат

Область рисования скелетных формул представлена холстом. В начальный момент при нажатии на произвольную область добавляется первая координата. Добавление осуществляется в динамический массив.

Последующие координаты рассчитываются относительно первой. В зависимости от кратности связи определяется, куда будет направлено звено ломанной линии: если связь тройная, или две рядом стоящие связи двойные, то угол между тремя координатами будет равен 180 градусам. В противном случае – 120 градусам. Расчет координат осуществляется вращением точки на окружности с радиусом, равным длине связи, на указанный выше угол.

Помимо координаты вершины, добавляется также предыдущая вершина, сведения об элементе (число свободных связей, кратность связи, атом).

Существует так же «безопасная зона» – 30 пикселей с каждой стороны, в пределах которых нельзя рисовать. Это создано для того, чтобы изображение не выходило за пределы холста.

Схема алгоритма представлена на рисунке Б.1.

* + 1. Алгоритм рисования скелетных формул

После того как координаты скелетной формулы получены, поступает команда перерисовать содержимое холста. Если вершина представлена атомом углерода, то рисуется просто ломанная линия, иначе рисуется укороченная с одного или двух концов линия, а в место расположения вершин вставляется строка, с обозначением химического элемента. Если элемент имеет свободные связи, то к этой строке также приписывается строка с обозначением водорода и числом свободных связей, если оно больше единицы. Схема алгоритма представлена на рисунке Б.2.

* 1. Разработка алгоритма преобразования двумерного представления в трехмерное

Для того, чтобы преобразовать скелетную формулу в полусферическую модель, нужно рассчитать координаты. В данной части важное место занимает именно связь соседних атомов. На начальном этапе проверяется, состоит ли соединения из одного элемента в массиве координат. Если это выполняется, то в массив трехмерных координат подаются уже ранее готовые координаты для метана, аммиака, фосфина, сероводорода, воды и галогеноводородов.

В противном случае строится дерево, где каждый элемент имеет четыре потомка и указатель на предка. Элементы добавляются в дерево на основании соответствия текущей двумерной координаты предыдущей двумерной координате в массиве. Указываются кратности их связей. Если у узла дерева есть свободные для данного элемента связи, то они заполняются водородами.

После построения дерева происходит сортировка потомков для каждого узла так, чтобы впереди была связь с наибольшей кратностью.

Координатой корня объявляется начало координат. Далее рассчитываются координаты: если связь узла или связь потомка равны трем или если связи и узла, и потомка равны двум, то они расположены на прямой линии, в противном случае – рассчитываются под углом 120 градусов с помощью угла , определяемого как угол нормали, и угла между осями абсцисс и аппликат.

После определения всех координат, производится обход дерева, добавления необходимых структур (координат и имя химического элемента) в динамический массив и удаление узлов дерева.

Затем выполняется нормировка координат так, чтобы момент всех координат был в начале координат. Для этого находится среднее арифметическое и вычитается из каждой координаты.

После этого массив передается области моделирования и поступает команда перерисовать содержимое. Схема алгоритма представлена на рисунке Б.3.

* 1. Разработка алгоритма рисования полусферической модели

После того, как координаты получены, определяются цвета и радиусы для каждой сферы. Радиусы пропорциональны реальным радиусам Ван-дер-Ваальса, используемым в благородных газах. Цвета определяются также по химическому элементу аналогично тому, как это используется в приложении JMol. Для того, чтобы большая часть изображения была охвачена, вводится коэффициент масштабирования, в начальный момент равный -7. Он представляет собой дополнительное слагаемое в оси аппликат. Он может быть изменен с помощью прокрутки мыши.

1. Конструирование программного средства
   1. Проектирование интерфейса программы

Системный набор инструментов для платформы Windows WinAPI позволяет создавать нативные Win32 приложения [9].

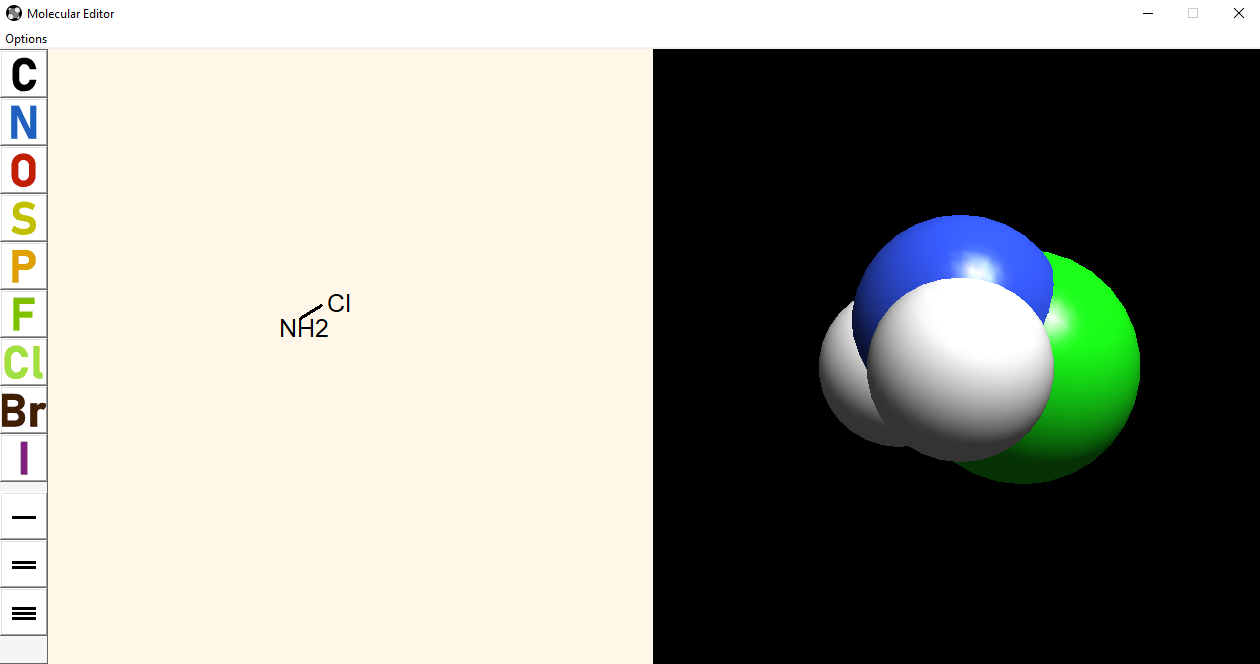
Рассмотрим компоненты интерфейса, которые я использовал.

Рисунок 4.1 – интерфейс приложения

Приложение имеет одно главное окно, на котором находится область рисования двухмерных молекул (рис. 4.1, выделена бежевым цветом), а также окно для панели инструментов с 12 дочерними окнами-кнопкам. Другим дочерним окном является область моделирования. Область моделирования предназначенна для работы с OpenGL [8].

* 1. Структура модулей программы

Программа состоит из следующих файлов:

1. Файл главной функции main.cpp.
2. Файлы модуля приложения Win32Application.h и Win32Application.cpp;
3. Файлы модуля, реализующего логику главного окна SkeletonCanvas.h и SkeletonCanvas.cpp;
4. Файлы модуля Toolbar.h и Toolbar.cpp;
5. Файлы модуля CalotteCanvas.h и CalotteCanvas.cpp;
6. Файлы модуля Converter.h и Converter.cpp;
7. Файлы модуля FileManager.h и FileManager.cpp;
8. Файлы со структурами данных и перечислениями State.h, State.cpp, Atom.h, AtomColor.h, Point.h, Element.h, ConverterNode.h, Vertex2D.h, Vertex3D.h;
9. Файлы ресурсов.
   1. Описание класса Win32Application

Модуль Win32Application является модулем класса Win32Application – приложения, реализующего шаблон «Одиночка». Его задача — это скрыть внутри себя цикл приложения, а также организовать взаимодействие между окнами, посредством сохранения внутри себе оберток над окнами. В таблице 4.1 представлены методы класса Win32Application, в таблице 4.2 – поля класса.

Таблица 4.1 Методы класса Win32Application

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| Instance | Метод-акцессор для эксземляра класса | static Win32Application\* Instance() | - | - |
| GetCalotte Canvas | Метод-акцессор для области моделирования | CalotteCanvas\* GetCalotteCanvas() | - | - |
| GetSkeleton Canvas | Метод-акцессор для области рисования | SkeletonCanvas\* GetSkeletonCanvas() | - | - |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| Run | Метод для запуска цикла приложения | int Run(HINSTANCE hInstance, int nCmdShow) | hInstance | Дескриптор приложения |
| nCmdShow | Информация о показе окна |
| SetupChild Windows | Метод конфигурации дочерних окон | int SetupChildWindows(HWND hwnd) | hwnd | Дескриптор главного окна |
| Win32 Application | Конструктор | Win32 Application() | - | - |

Таблица 4.2 Поля данных класса Win32Application

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| toolbar | Toolbar\* | Панель инструментов |
| calotteCanvas | CalotteCanvas\* | Холст для трёхмерного моделирования |
| skeletonCanvas | SkeletonCanvas\* | Холст для двумерного рисования |
| hInst | HINSTANCE | Дескриптор приложения |
| szTitle | WCHAR\* | Заголовок приложения |
| szWindowClass | WCHAR\* | Имя класса главного окна |

* 1. Описание класса SkeletonCanvas

Модуль SkeletonCanvas является модулем класса SkeletonCanvas, который является оберткой главного окна приложения. Методы представлены в таблице 4.3, поля – в таблице 4.4.

Таблица 4.3 Методы класса SkeletonCanvas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| SkeletonCanvas | Метод-конструктор | Skeleton Canvas() | - | - |

Продолжение таблицы 4.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| MakeVisible | Метод показа окна | void MakeVisible(int nCmdShow) | nCmdShow | Информация о показе окна |
| LeftClick | Метод обработки нажатия мыши | void LeftClick(int x, int y) | x | Координата-абсцисс нажатия |
| y | Координата-ордината нажатия |
| Paint | Метод обработки рисования | void Paint(HDC dc) | dc | Дескриптор контекста устройства |
| Clear | Метод очистки окна | void Clear() | - | - |
| Save | Метод сохранения изображения | void Save() | - | - |
| Configure | Создание окна | void Configure(HINSTANCE hInstance, WCHAR\* windowClass, WCHAR\* title) | hInstance | Декскриптор приложения |
| windowClass | Имя класса окна |
| title | Заголовок окна |
| GetWindow | Метод-акцесор дескриптора окна | HWND GetWindow() | - | - |
| GetState | Метод-акцесор состояния окна | State\* GetState() | - | - |
| GetVertices | Метод-акцесор массива вершин | std::vector<Vertex2D>& GetVertices() | - | - |
| Convert | Метод обработки создания трехмерной модели | void Convert() | - | - |

Продолжение таблицы 4.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| WindowProc | Метод обработки событий окна | static LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) | hWnd | Дескриптор окна |
| message | Код события |
| wParam | Дополнитель-ные параметры |
| lParam |
| IsWithinSafeZone | Метод проверки на безопасную зону | bool IsWithinSafeZone(int x, int y) | x | Координата |
| y |
| CalculateVertexPosition | Метод определения положения следующей вершины | void CalculateVertexPosition(int x0, int y0, int x, int y, int value, int& x1, int& y1) | x0 | Точка на окружности |
| y0 |
| x | Центр окружности |
| y |
| value | Опция вращения |
| x1 | Результат |
| y1 |

Таблица 4.4 Поля данных класса SkeletonCanvas

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| window | HWND | Дескриптор окна |
| state | State | Состояние (выбранный элемент и связь) |
| vertices | std::vector<Vertex2D> | Массив вершин |
| length | int | Длина расстояния между вершинами |
| pi | double | Константа |

* 1. Описание класса Toolbar

В данном модуле объявлен класс Toolbar, реализующий обертку над панелью инструментов. Методы класса Toolbar представлены в таблице 4.5, поля данных – 4.6.

Таблица 4.5 Методы класса Toolbar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| Toolbar | Метод-конструктор | Toolbar() | - | - |
| GetWindow | Метод-акцесор окна | HWND GetWindow() | - | - |
| Configure | Метод конфигурации создания окна | void Configure(HWND parent, HINSTANCE hInst) | parent | Дескриптор родительского окна |
| hInst | Дескриптор программы |
| ToolsProc | Метод обработки событий окна | static LRESULT CALLBACK ToolsProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) | hWnd | Дескриптор окна |
| message | Код события |
| wParam | Дополнитель-ные параметры |
| lParam |

Таблица 4.6 Поля данных класса Toolbar

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| window | HWND | Дескриптор окна |
| szToolsDialogClass | LPCWSTR | Имя класса окна |

* 1. Описание класса CalotteCanvas

Класс CalotteCanvas поддерживает работу с OpenGL, в связи с чем поддерживает рисование продвинутой двумерной и трехмерной графики. В своей работе я использовал функции OpenGL и функции библиотеки утилит GLU [8]. Библиотека GLU позволяет создавать готовые сферы по радиусам и координатам. Методы класса CalotteCanvas представлены в таблице 4.7, поля данных – 4.8.

Таблица 4.7 Методы класса CalotteCanvas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| CalotteCanvas | Метод-конструктор | CalotteCanvas() | - | - |
| GetWindow | Метод-акцесор дескриптора окна | HWND GetWindow() | - | - |
| Configure | Метод конфигурации и создания окна | void Configure(HWND parent, HINSTANCE hInst) | parent | Дескриптор родительского окна |
| hInst | Дескриптор программы |
| Zoom | Метод обработки масштабирования | void Zoom(int delta) | delta | Изменение колеса прокрутки |
| ReleaseResources | Метод освобождения ресурсов OpenGL | void ReleaseResources() | - | - |
| Render | Метод рендеринга | void Render() | - | - |
| Save | Метод вызова сохранения | void Save() | - | - |
| SetVertices | Метод-мутатор для массива вершин | void SetVertices(std::vector<Vertex3D> vertices) | vertices | Массив вершин |
| Sphere | Рисование сферы | void Sphere(Vertex3D& vertex, int pos) | vertex | Координата |
| pos | Индекс вершины |
| InitGL | Подготовка рендеринга | void InitGL() | - | - |

Продолжение таблицы 4.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| SetupContext | Настройка контекста устройства для OpenGL | void SetupContext() | - | - |
| CanvasProc | Метод обработки событий окна | static LRESULT CALLBACK CanvasProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) | hWnd | Дескриптор окна |
| message | Код события |
| wParam | Дополнитель-ные параметры |
| lParam |

Таблица 4.8 Поля данных класса CalotteCanvas

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| vertices3D | std::vector <Vertex3D> | Массив трехмерных вершин |
| reusableQuadrics | std::vector <GLUquadricObj\*> | Массив тел библиотеки GLU |
| hRC | HGLRC | Графический контекст OpenGL |
| hDC | HDC | Контекст устройства окна |
| light\_ambient | const GLfloat[4] | Цветовая матрица рассеянного света |
| light\_diffuse | const GLfloat[4] | Цветовая матрица рассеянного света для одной плоскости |
| light\_specular | const GLfloat[4] | Цветовая матрица точечного света |
| light\_position | const GLfloat[4] | Цветовая матрица света |
| mat\_ambient | const GLfloat[4] | Матрица рассеянного света |
| mat\_diffuse | const GLfloat[4] | Матрица рассеянного света для одной плоскости |
| mat\_specular | const GLfloat[4] | Матрица точечного света |
| high\_shininess | const GLfloat[1] | Матрица яркости |
| dist | float | Изменение оси аппликат |
| szCanvasClass | LPCWSTR | Имя класса окна |

* 1. Описание класса Converter

Класс Converter организовывает преобразование двумерных вершин в трехмерные. Детальное описание алгоритма приведено в пункте 3.2. Методы класса представлены в таблице 4.9, поля данных – в таблице 4.10.

Таблица 4.9 Методы класса Converter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| Converter | Метод-конструктор | Converter() | - | - |
| ~Converter | Метод-деструктор | ~Converter() | - | - |
| ConvertTo Callote | Рисование сферы | std::vector <Vertex3D> ConvertToCallote (std::vector <Vertex2D> &vertices2D) | vertices2D | Ссылка на массив двумерных координат |
| Tethrahedron | Тетраэдр для простого объекта | void Tethrahedron(Element vertex) | vertex | Элемент вершины |
| Triangle | Рисование треугольной молекулы для простого объекта | void Triangle(Element vertex) | vertex | Элемент вершины |
| Linear | Рисование линейной молекулы для простого объекта | void Linear(Element vertex) | vertex | Элемент вершины |
| IsSimple | Проверка на «простоту», соответствие базовым молекулам | bool IsSimple(std::vector<Vertex2D>& vertices2D) | vertices2D | Ссылка на массив двумерных координат |

Продолжение таблицы 4.9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| Spin | Расчет координат простого тетраэдра | Point spin(Point &root, float phi, float theta) | root | Координата центра сферы |
| phi | Угол нормали |
| theta | Угол между осями абсцисс и аппликат. |
| MakeTree | Построение дерева с четырьмя потомками | void MakeTree(ConverterNode\* root, std::vector<Vertex2D>& vertices2D) | root | Узел дерева |
| vertices2D | Массив вершин |
| SortTree | Сортировка потомков по возрастанию | void SortTree(ConverterNode\* root) | root | Узел дерева |
| Init Coordinates | Начало расчета координат | void InitCoordinates(ConverterNode\* root) | root | Узел дерева |
| Implement Coordinates | Рекурсивный расчет координат | void ImplementCoordinates(ConverterNode\* root) | root | Узел дерева |
| ClearTree | Очистка дерева | void ClearTree(ConverterNode\* root) | root | Узел дерева |
| Centrify Coordinates | Центрирование координат относительно начала координат | void Centrify Coordinates() | - | - |
| Advanced Spin | Непосредственный координат для нелинейных связей | void AdvancedSpin(ConverterNode\* root) | root | Узел дерева |

Таблица 4.10 Поля данных класса Converter

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| vertices3D | std::vector <vertex3D> | Массив тел библиотеки GLU |
| pi | double | Константа |

* 1. Описание класса FileManager

Класс FileManager объявляет ряд статических функций по преобразованию растрового изображения в памяти в файл в формате BMP. Методы перечислены в таблице 4.11.

Таблица 4.11 Методы класса FileManager

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| Save | Метод сохранения и вызова диалогового окна | static void Save(HBITMAP bitmap) | bitmap | Дескриптор растрового изображения |
| CreateBMPFile | Создание файла растрового изображения | static void CreateBMPFile(LPTSTR pszFile, HBITMAP hBMP) | pszFile | Имя файла |
| hBMP | Дескриптор растрового изображения |
| CreateBitmapInfoStruct | Создание структуры с информацией о растровом изображении | static PBITMAPINFO CreateBitmapInfoStruct(HBITMAP hBmp) | hBMP | Дескриптор растрового изображения |

* 1. Описание структур

Структура Atom хранит информацию о химическом элементе, свободных и занятых связях. Информацией о полях указана в таблице 4.12.

Структура State хранит информацию для рисования молекул, настраивуемую с помощью панели инструментов. Информация о методах структуры указана в таблице 4.13, о полях – в таблице 4.14.

Структура Vertex2D хранит информацию о вершине в скелетной формуле, ее координатах, координатах связанной вершины. Информация о методах структуры указана в таблице 4.15, о полях – в таблице 4.16.

Структура Point хранит информацию о трехмерной координате. Информация о полях структуры указана в таблице 4.17.

Структура Vertex3D хранит информацию о вершине в трехмерной модели, ее координатах и элементе. Информация о полях структуры указана в таблице 4.18.

Структура AtomColor кодирует RGB-цвета в формате от 0 до 1 для удобного представления в OpenGL. Информация о полях структуры указана в таблице 4.19.

Структура ConverterNode используется для построения дерева с четырьмя потомками для дальнейшего преобразования в трехмерные координаты. Информация о полях структуры указана в таблице 4.20.

Таблица 4.12 Поля данных структуры Atom

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| element | Element | Перечисление с данными об элементе |
| freeBonds | int | Количество свободных связей |
| bond | int | Количество связей |

Таблица 4.11 Методы структуры State

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| GetBond | Метод-акцесор кратности связи | int GetBond() | - | - |
| SetBond | Метод-мутатор кратности связи | void SetBond(int bond) | bond | Кратность связи |
| GetElement | Метод-акцесор элемента | Element GetElement() | - | - |
| SetElement | Метод-мутатор элемента | void SetElement( Element element) | element | Химический элемент |
| GetMaxBonds | Определение максимального возможного числа связей | static int GetMaxBonds( Element element) | element | Химический элемент |

Таблица 4.14 Поля данных структуры State

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| element | Element | Перечисление с данными об элементе |
| freeBonds | int | Количество свободных связей |
| bond | int | Количество связей |

Таблица 4.15 Методы структуры Vertex2D

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| Description | Описание текста вершины | std::wstring Description() | - | - |

Таблица 4.16 Поля данных структуры Vertex2D

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| atom | Atom | Перечисление с данными об элементе |
| x | int | Координата |
| y | int |
| prevx | int | Координата предыдущей вершины |
| prevy | int |
| shortRear | bool | Флаг того, что нужно сократить длину черты с конца при рисовании |
| shortFront | bool | Флаг того, что нужно сократить длину черты с головы при рисовании |

Таблица 4.17 Поля данных структуры Point

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| x | float | Координата x |
| y | float | Координата y |
| z | float | Координата z |

Таблица 4.18 Поля данных структуры Vertex3D

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| element | Element | Перечисление с данными об элементе |
| point | Point | Координата |

Таблица 4.19 Поля данных структуры AtomColor

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| red | float | Красный цвет от 0 до 1 |
| green | float | Зеленый цвет от 0 до 1 |
| blue | float | Синий цвет от 0 до 1 |

Таблица 4.20 Поля данных структуры ConverterNode

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение поля** |
| first | ConverterNode\* | Первый дочерний узел |
| second | ConverterNode\* | Второй дочерний узел |
| third | ConverterNode\* | Третий дочерний узел |
| fourth | ConverterNode\* | Четвертый дочерний узел |
| parent | ConverterNode\* | Родительский узел |
| vertex | Vertex3D | Значение вершины |
| index | int | Индекс вершины |
| lastBond | int | Количество связей |
| bond | int | Индекс связи родительской |

В результате этапа конструирования создано программное средство. Схема программы представлена в приложении А. Листинг программы представлен в приложении В.

1. Тестирование, проверка работоспособности и анализ полученных результатов

В данном разделе я остановлюсь на тестировании программного средства. В таблицах 5.1, 5.2 и 5.3 представлены результаты тестирования программного средства.

* 1. Тестирование алгоритма рисования

Таблица 5.1 – тестирование алгоритма рисования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Рисование разветвленных углеводород-ных молекул | Рисование одинарными связями. Из одной вершины рисуется несколько других. | Связи рисуются под углом 120 градусов к другим, кроме четвертой (60 градусов) |  |
| 2 | Рисование многоэлемент-ных молекул | Рисование молекулы со сменой активного элемента | Связи у неуглеродных атомов укорочены, имеются обозначения буквенные атомов |  |
| 3 | Рисование тройных связей | Рисование молекулы, смена активной связи на тройную | Две соседние связи стройной расположены на одной прямой с тройной |  |
| 4 | Рисование нескольких двойных связей | Выбор двойной связи для углерода и рисование молекулы | Двойные связи расположены на одной прямой |  |
| 5 | Тестирование молекулы с одной вершиной | Выбор элемента, нажатие ЛКМ по области рисования | Водородное соединение |  |
| 6 | Очистка экрана | Рисование молекулы, нажатие на кнопку «Очистить» | Очистка экрана | Тест пройден |

* 1. Тестирование сохранения изображения

Таблица 5.2 – тестирование сохранения изображения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Сохранение двумерного изображения в формат BMP | Рисование молекулы, нажатие на кнопку «сохранить» для двумерного изображения. Формат файла –  BMP. | Получение изображения  “Molecule.bmp”в выбранной папке | Тест пройден |
| 2 | Сохранение трехмерного изображения в формат BMP | Рисование молекулы, моделирование, нажатие на кнопку «сохранить» для трехмерного изображения. Формат файла –  BMP. | Получение изображения  “Molecule3D.bmp”в выбранной папке | Тест пройден |

* 1. Тестирование трехмерного моделирования

Таблица 5.3 – тестирование трехмерного моделирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последователь-ность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Моделирование молекулы воды | Выбор элемента «кислород», нажатие ЛКМ на область рисования, нажатие на кнопку «Преобразовать». | Треугольная бело-красно-белая молекула |  |
| 2 | Моделирование молекулы фтора | Выбор элемента «фтор», нажатие ЛКМ на область рисования, нажатие на нарисованную вершину, нажатие на кнопку «Преобразовать». | Двойная светло-зеленая молекула |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | Моделирование линейной молекулы на примере сероуглерода | Выбор элемента «сера», нажатие ЛКМ на область рисования, выбор элемента «углерод», кратности связи «двойная», нажатие ЛКМ на область рисования, выбор элемента «сера», нажатие ЛКМ на область рисования, нажатие на кнопку «Преобразовать». | Желтая сфера, серая сфера, желтая сфера расположенные на одной прямой |  |
| 4 | Моделирование линейной молекулы на примере ацетилена | Выбор элемента «углерод», нажатие ЛКМ на область рисования, выбор кратности связи «тройная», нажатие ЛКМ на область рисования, нажатие на кнопку «Преобразовать». | Белая сфера, две серые сферы, белая сфера, расположенные на одной прямой |  |
| 5 | Моделирование молекулы с дисульфидным мостиком | Выбор элемента «углерод», нажатие ЛКМ на область рисования, выбор элемента «сера», два нажатия ЛКМ на область рисования, выбор элемента «углерод», нажатие ЛКМ на область рисования, нажатие на кнопку «Преобразовать». | Две серые сферы с тремя белыми маленькими сферами рядом и две желтые сферы между серыми |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональ-ность** | **Последователь-ность действий** | **Ожидаемый результат** | **Полученный результат** |
| 6 | Моделирование молекулы перекиси водорода | Выбор элемента «фосфор», два клика ЛКМ, выбор элемента, нажатие на кнопку «Преобразовать». | Две красные сферы и две белые по краям.  Молекула изогнута |  |
| 7 | Моделирование молекулы хлорэтана | Выбор элемента «углерод», два нажатия ЛКМ на область рисования, выбор элемента «хлор», нажатие ЛКМ по второй вершине, нажатие на кнопку «Преобразовать». | Две серые сферы: у первой три белых сферы рядом, у другой две белых и одна зеленая |  |
| 8 | Моделирование молекулы тетрафторида дифосфора | Выбор элемента «фосфор», два клика ЛКМ, выбор элемента «фтор», по два клика ЛКМ по вершинам, нажатие на кнопку «Преобразовать». | Две оранжевые сферы и четыре зеленые |  |
| 9 | Масштабирование изображения | Получение модели, прокрутка колесиком в одном из направлений | Удаление или приближение молекулы | Тест пройден |

Продолжение таблицы 5.3

Подводя итог, отмечу, что программа отвечает заданным функциональным требованиям, наблюдается стабильность в работе. Вопросов к эстетической части не имеется.

1. Руководство по установке и использованию

Молекулярный редактор обеспечивает рисование скелетных формул молекулярных веществ, а также преобразовывает создание пользователем формулы в трехмерные полусферические модели.

Данное программное средство разработано для использования на операционной системе Windows (рис. 6.1).



Рисунок 6.1 – Программное средство на устройстве под управлением Windows 10

Для нормальной работы программы папка программного средства должна содержать исполняемый файл Project.exe.

При запуске программы отображается основное окно (рис 6.1). В левой части окна приложения расположена панель инструментов. В верхней части расположено выпадающее меню.

Панель инструментов можно условно разделить на следующие области:

1. Область выбора химического элемента;
2. Область выбора химической связи;

При запуске программы по умолчанию текущим элементом устанавливается «углерод», а текущей кратностью связи – «одинарная связь».

Рисование осуществляется нажатием левой кнопки мыши по области рисования светло-желтого цвета (рис 6.2).

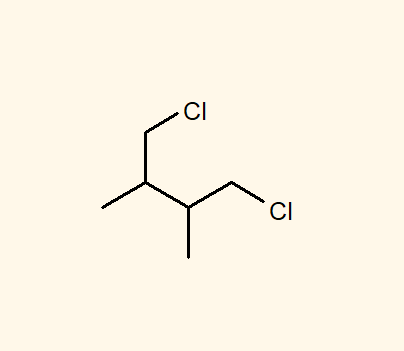


Рисунок 6.2 – область рисования

В начале пользователь нажимает на произвольную область, после чего все остальные вершины строятся относительно первой. Добавление связи к вершине осуществляется нажатием на нее или ее строковое обозначение, если свободные связи имеются.

К предлагаемым пользователю элементам относятся: углерод, азот, кислород, фосфор, сера, фтор, хлор, бром, йод.

Очистка области рисования осуществляется нажатием на кнопку «очистить» в выпадающем меню.

Нарисованную структурную формулу можно преобразовать в трехмерную полусферическую модель, которая отобразится в правой (черной) части окна (рис 6.4). Преобразование осуществляется посредством нажатия на кнопку «Преобразовать» в выпадающем меню.

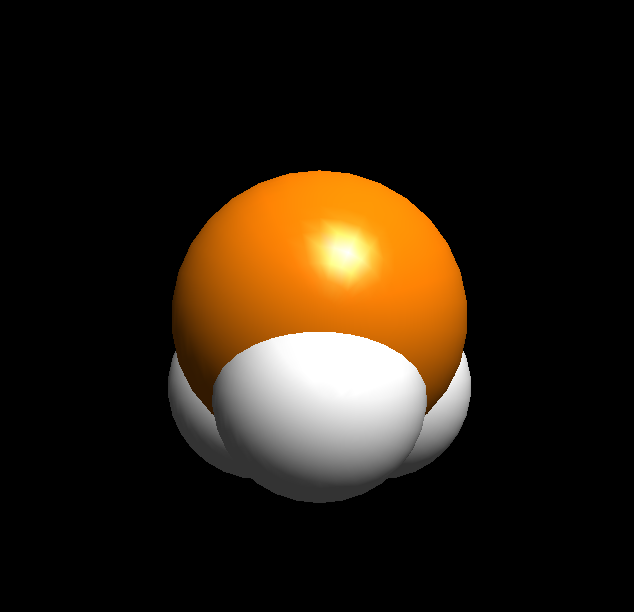


Рисунок 6.4 – область моделирования

Трехмерные модели можно масштабировать с помощью прокрутки колесиком мыши.

Полученные структурные формулы и трехмерные модели можно сохранить в формат BMP. Для этого необходимо нажать на кнопку «Сохранить структурную формулу» или «Сохранить трехмерную формулу».

Заключение

В ходе данной работы создано программное средство «Молекулярный редактор», которое предоставляет пользователю возможность рисовать и моделировать молекулы.

При этом в ходе работы мною получен опыт работы с парадигмой ООП, с созданием собственных классов, структур. Я использовал разделение методов и полей данных на открытые и закрытые, ознакомился с работой конструкторов и деструкторов.

При работе с трехмерными изображениями я ознакомился с работой библиотеки OpenGL и библиотекой утилит GLU.

В ходе работы я изучил жизненный цикл Win32 приложения, принципы создания подобных приложений. При создании приложения все компоненты интерфейса прописывались вручную, для чего потребовалось хорошо разобраться в англоязычной документации.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Список литературы

[1] Wikipedia [Электронный ресурс] – Skeletal Formula – Режим доступа: <https://en.wikipedia.org/wiki/Skeletal_formula>

[2] University of California – Van der Waals radii. – Режим доступа: <https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/midas/vdwrad.html>

[3] Wikipedia [Электронный ресурс] – CPK Coloring – Режим доступа: <https://en.wikipedia.org/wiki/CPK_coloring>

[4] Цветовая палитра JMOL [Электронный ресурс] – Coloring. – Режим доступа: <http://jmol.sourceforge.net/jscolors/>

[5] Редактор JMOL [Электронный ресурс] – JMOL – Режим доступа: <https://sourceforge.net/projects/jmol/>

[6] Редактор PyMol [Электронный ресурс] – PyMol – Режим доступа: <https://pymol.org/2/>

[7] Редактор BKChem [Электронный ресурс] – BKChem – Режим [доступа: http://bkchem.zirael.org/](доступа:%20%20http:/bkchem.zirael.org/)

[8] Официальная документация OpenGL [Электронный ресурс] – OpenGL® 2.1, GLX, and GLU Reference Pages. – Режим доступа: <https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/>

[9] Официальная документация Microsoft [Электронный ресурс] – Windows API index. – Режим доступа: https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list/

Приложение A

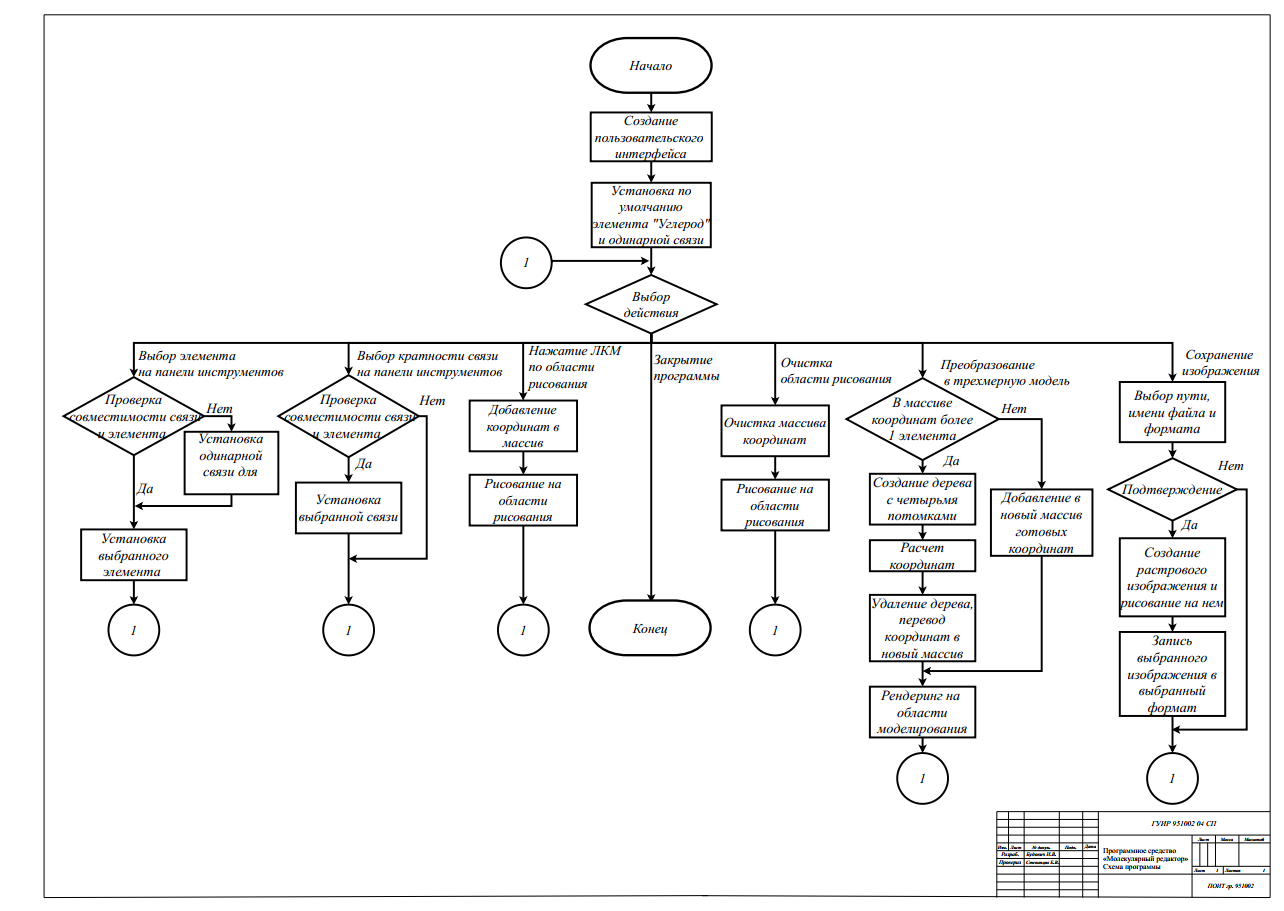


Рисунок А.1 – Схема алгоритма программы

Приложение Б

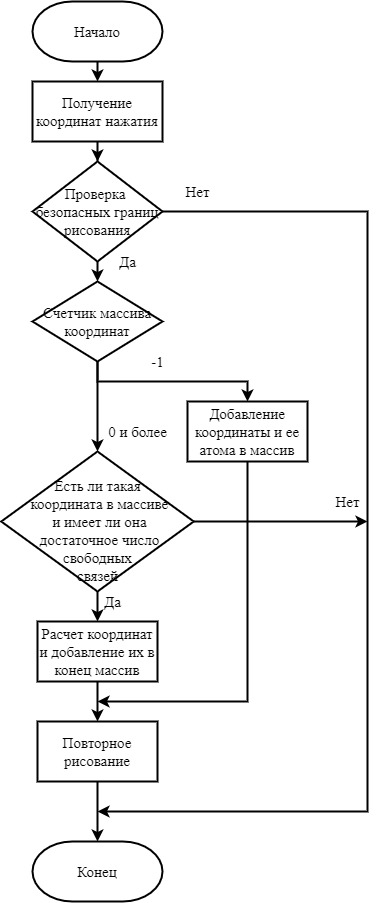


Рисунок Б.1 – Схема алгоритма добавления координат

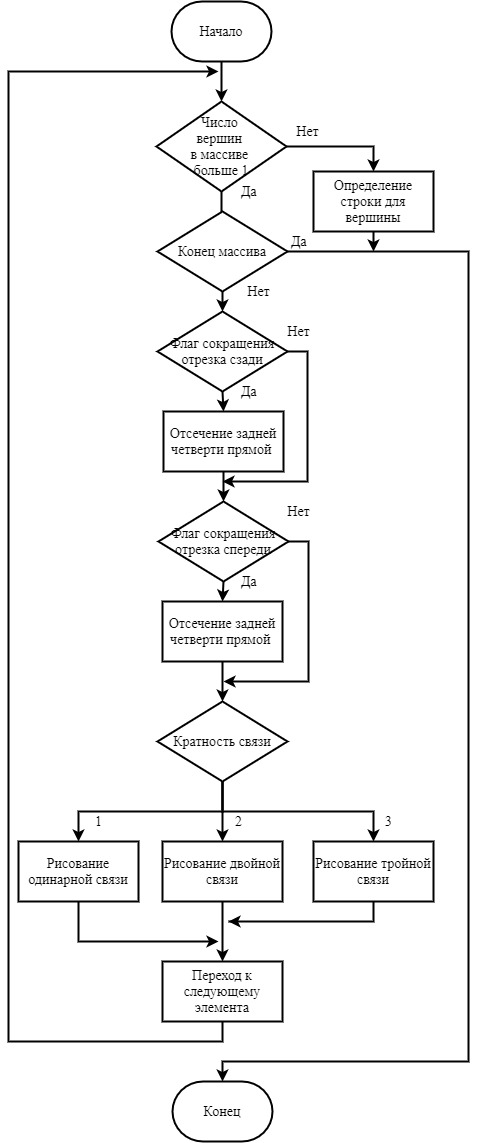


Рисунок Б.2 – Схема алгоритма рисования

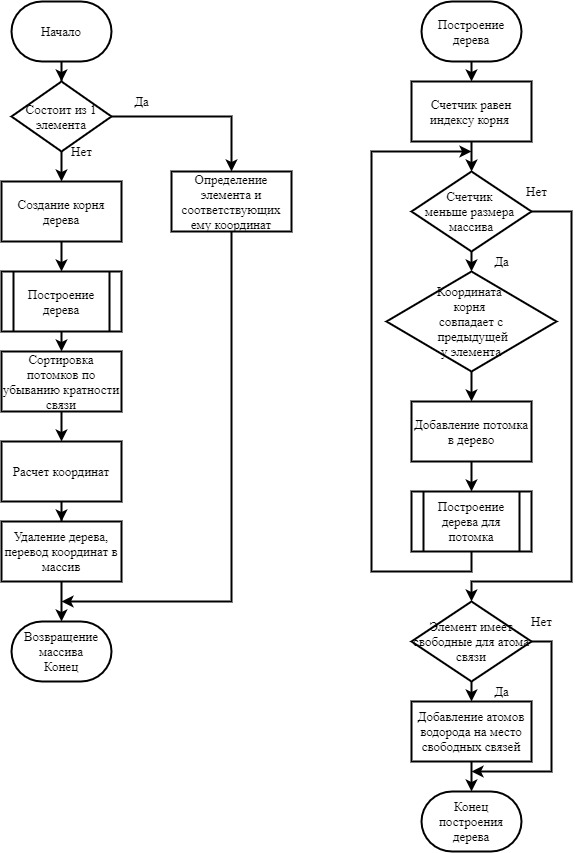


Рисунок Б.3 – Схема алгоритма преобразования в трехмерное представление

Приложение В

Листинг файла main.cpp

#include "framework.h"

#include "main.h"

#include "Win32Application.h"

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

return Win32Application::Instance()->Run(hInstance, nCmdShow);}

Листинг файла Win32Application.h

#include "CalotteCanvas.h"

#include "resource.h"

class SkeletonCanvas;

class Toolbar;

class CalotteCanvas;

class Win32Application

{

public:

static Win32Application\* Instance();

CalotteCanvas\* GetCalotteCanvas() { return calotteCanvas; }

SkeletonCanvas\* GetSkeletonCanvas() { return skeletonCanvas; }

int Run(HINSTANCE hInstance, int nCmdShow);

int SetupChildWindows(HWND hwnd);

private:

Win32Application();

Toolbar\* toolbar;

CalotteCanvas\* calotteCanvas;

SkeletonCanvas\* skeletonCanvas;

HINSTANCE hInst;

WCHAR szTitle[MAX\_LOADSTRING];

WCHAR szWindowClass[MAX\_LOADSTRING];

};

Листинг файла Win32Application.cpp

#include "Win32Application.h"

Win32Application\* Win32Application::Instance() {

static Win32Application instance;

return &instance;

}

Win32Application::Win32Application() {

this->toolbar = nullptr;

this->skeletonCanvas = nullptr;

this->hInst = nullptr;

this->calotteCanvas = nullptr;

}

int Win32Application::Run(HINSTANCE hInstance, int nCmdShow) {

hInst = hInstance;

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_COURSEPROJECTIV, szWindowClass, MAX\_LOADSTRING);

this->skeletonCanvas = new SkeletonCanvas();

skeletonCanvas->Configure(hInstance, szWindowClass, szTitle);

skeletonCanvas->MakeVisible(nCmdShow);

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_COURSEPROJECTIV));

MSG msg;

while (GetMessage(&msg, NULL, 0, 0))

{

if (!TranslateAcceleratorW(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

this->calotteCanvas->ReleaseResources();

delete calotteCanvas;

delete toolbar;

delete skeletonCanvas;

return (int)msg.wParam;

}

int Win32Application::SetupChildWindows(HWND hwnd) {

this->toolbar = new Toolbar();

this->toolbar->Configure(hwnd, this->hInst);

this->calotteCanvas = new CalotteCanvas();

this->calotteCanvas->Configure(hwnd, this->hInst);

return 0;

}

Листинг файла SkeletonCanvas.h

#include <Windows.h>

#include "resource.h"

#include "Win32Application.h"

#include "State.h"

#include <vector>

#include <string>

#include "Vertex2D.h"

#include <windowsx.h>

#include "Converter.h"

#include "AtomColor.h"

#include "FileManager.h"

class FileManager;

class SkeletonCanvas

{

public:

SkeletonCanvas();

void MakeVisible(int nCmdShow);

void LeftClick(int x, int y);

void Paint(HDC dc);

void Clear();

void Save();

void Configure(HINSTANCE hInstance, WCHAR\* windowClass, WCHAR\* title);

HWND GetWindow() { return window; }

State\* GetState() { return &state; }

std::vector<Vertex2D>& GetVertices() { return vertices; }

private:

static LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void Convert();

bool IsWithinSafeZone(int x, int y);

void CalculateVertexPosition(int x0, int y0, int x, int y, int value, int& x1, int& y1);

HWND window;

State state;

std::vector<Vertex2D> vertices;

const int length = 50;

const double pi = 3.14159265358979323846;

};

Листинг файла SkeletonCanvas.cpp

#include "SkeletonCanvas.h"

SkeletonCanvas::SkeletonCanvas() {

this->window = nullptr;

}

void SkeletonCanvas::Configure(HINSTANCE hInstance, WCHAR\* windowClass, WCHAR\* title) {

WNDCLASSEXW wcex = { 0 };

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WindowProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_COURSEPROJECTIV));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = CreateSolidBrush(RGB(255, 248, 233));

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_COURSEPROJECTIV);

wcex.lpszClassName = windowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

RegisterClassExW(&wcex);

this->window = CreateWindowW(windowClass, title, WS\_OVERLAPPEDWINDOW ^ WS\_THICKFRAME ^ WS\_MAXIMIZEBOX,

CW\_USEDEFAULT, 0, 1280, 674, nullptr, nullptr, hInstance, nullptr);

}

void SkeletonCanvas::MakeVisible(int nCmdShow) {

ShowWindow(window, nCmdShow);

UpdateWindow(window);

}

void SkeletonCanvas::LeftClick(int x, int y) {

Vertex2D currentVertex{};

if (IsWithinSafeZone(x, y))

{

int counter = vertices.size() - 1;

switch (counter)

{

case -1:

currentVertex.x = x;

currentVertex.y = y;

currentVertex.prevx = x + round(length \* cos(pi / 6));

currentVertex.prevy = y - round(length \* sin(pi / 6));

currentVertex.shortFront = (state.GetElement() != Element::Carbon);

currentVertex.shortRear = false;

currentVertex.atom.element = state.GetElement();

currentVertex.atom.freeBonds = State::GetMaxBonds(state.GetElement());

currentVertex.atom.bond = 0;

vertices.push\_back(currentVertex);

break;

case 0:

if ((vertices[0].x - 10 < x) && (vertices[0].x + 10 > x) && (vertices[0].y - 10 < y) && (vertices[0].y + 10 > y) && (vertices[0].atom.freeBonds - state.GetBond() >= 0))

{

currentVertex.x = vertices[0].prevx;

currentVertex.y = vertices[0].prevy;

currentVertex.prevx = vertices[0].x;

currentVertex.prevy = vertices[0].y;

currentVertex.atom.element = state.GetElement();

vertices[0].atom.freeBonds -= state.GetBond();

vertices[0].atom.bond = state.GetBond();

currentVertex.atom.freeBonds = State::GetMaxBonds(state.GetElement()) - state.GetBond();

currentVertex.atom.bond = state.GetBond();

currentVertex.shortFront = (state.GetElement() != Element::Carbon);

currentVertex.shortRear = vertices[0].shortFront;

vertices.push\_back(currentVertex);

}

break;

default:

bool flag = false;

int i;

for (i = 0; i <= counter; i++)

{

if ((vertices[i].x - 10 < x) && (vertices[i].x + 10 > x) && (vertices[i].y - 10 < y) && (vertices[i].y + 10 > y) && (vertices[i].atom.freeBonds - state.GetBond() >= 0))

{

flag = true;

break;

}

}

if (flag)

{

int x1;

int y1;

int exodus;

if (state.GetBond() == 3 || vertices[i].atom.bond == 3 || (state.GetBond() == 2 && vertices[i].atom.bond == 2))

exodus = 3;

else

{

if (vertices[i].atom.bond == 2)

exodus = 2 - vertices[i].atom.freeBonds;

else

exodus = (State::GetMaxBonds(vertices[i].atom.element) - vertices[i].atom.freeBonds - 1);

}

CalculateVertexPosition(vertices[i].prevx, vertices[i].prevy, vertices[i].x, vertices[i].y, exodus, x1, y1);

currentVertex.x = x1;

currentVertex.y = y1;

currentVertex.prevx = vertices[i].x;

currentVertex.prevy = vertices[i].y;

currentVertex.atom.element = state.GetElement();

vertices[i].atom.freeBonds -= state.GetBond();

currentVertex.atom.bond = state.GetBond();

currentVertex.atom.freeBonds = State::GetMaxBonds(state.GetElement()) - state.GetBond();

currentVertex.shortFront = (state.GetElement() != Element::Carbon);

currentVertex.shortRear = vertices[i].shortFront;

vertices.push\_back(currentVertex);

}

}

RedrawWindow(window, NULL, NULL, RDW\_INVALIDATE | RDW\_UPDATENOW | RDW\_ERASE);

}

}

void SkeletonCanvas::CalculateVertexPosition(int x0, int y0, int x, int y, int value, int &x1, int &y1)

{

double alpha, c, s;

int rx, ry;

switch (value)

{

case 0:

alpha = 2\*pi/3;

if (y0>=y)

alpha = -alpha;

break;

case 1:

alpha = 2\*pi/3;

if (y0<y)

alpha = -alpha;

break;

case 2:

alpha = pi/3;

if (y0>=y)

alpha = -alpha;

break;

case 3:

alpha = pi;

break;

default:

alpha = 2\*pi/3;

}

rx = x0 - x;

ry = y0 - y;

c = cos(alpha);

s = sin(alpha);

x1 = round( x + rx \* c - ry \* s);

y1 = round(y + rx \* s + ry \* c);

}

bool SkeletonCanvas::IsWithinSafeZone(int x, int y) {

int x0 = 640;

int y0 = 650;

return ((x > 100) && (x < x0 - 50) && (y > 50) && (y < y0 - 50));

}

void SkeletonCanvas::Paint(HDC dc) {

if (vertices.size() == 0) {

return;

}

HGDIOBJ original = NULL;

original = SelectObject(dc, GetStockObject(DC\_PEN));

DeletePen(original);

HPEN pen = CreatePen(PS\_SOLID, 3, RGB(0, 0, 0));

SelectObject(dc, pen);

HFONT hFont = CreateFontW(30, 0, 0, 0, FW\_MEDIUM, FALSE, FALSE, FALSE, DEFAULT\_CHARSET, OUT\_OUTLINE\_PRECIS,

CLIP\_DEFAULT\_PRECIS, CLEARTYPE\_QUALITY, VARIABLE\_PITCH, TEXT("Arial"));

SelectObject(dc, hFont);

SetBkMode(dc, TRANSPARENT);

std::wstring str;

int counter = vertices.size() - 1;

if (counter == 0)

{

str = vertices[0].Description();

LPCWSTR result =str.c\_str();

RECT rect{};

rect.left = vertices[0].x;

rect.top = vertices[0].y - 10;

rect.bottom = rect.top + 40;

rect.right = rect.left + 80;

DrawText(dc, result, -1, &rect, DT\_LEFT | DT\_TOP);

}

else

{

if (vertices[0].atom.element != Element::Carbon)

{

str = vertices[0].Description();

LPCWSTR result = str.c\_str();

RECT rect{};

rect.left = vertices[0].x - 10;

rect.top = vertices[0].y - 10;

rect.bottom = rect.top + 40;

rect.right = rect.left + 80;

DrawText(dc, result, -1, &rect, DT\_LEFT | DT\_TOP);

}

int x0, x1, y0, y1;

for (int i = 1; i <= counter; i++)

{

if (vertices[i].atom.element != Element::Carbon)

{

str = vertices[i].Description();

LPCWSTR result = str.c\_str();

RECT rect{};

rect.left = vertices[i].x - 5;

rect.top = vertices[i].y - 10;

rect.bottom = rect.top + 40;

rect.right = rect.left + 80;

DrawText(dc, result, -1, &rect, DT\_LEFT | DT\_TOP);

}

if (vertices[i].shortRear)

{

x0 = round(vertices[i].x / 4.0 + vertices[i].prevx / 4.0 \* 3);

y0 = round(vertices[i].y / 4.0 + vertices[i].prevy / 4.0 \* 3);

}

else

{

x0 = vertices[i].prevx;

y0 = vertices[i].prevy;

}

if (vertices[i].shortFront)

{

x1 = round(vertices[i].x / 4.0 \* 3 + vertices[i].prevx / 4.0);

y1 = round(vertices[i].y / 4.0 \* 3 + vertices[i].prevy / 4.0);

}

else

{

x1 = vertices[i].x;

y1 = vertices[i].y;

}

switch (vertices[i].atom.bond)

{

case 2:

if (vertices[i].prevx == vertices[i].x)

{

MoveToEx(dc, x0 - 2, y0, NULL);

LineTo(dc, x1 - 2, y1);

MoveToEx(dc, x0 + 2, y0, NULL);

LineTo(dc, x1 + 2, y1);

}

else

{

MoveToEx(dc, x0, y0 - 3, NULL);

LineTo(dc, x1, y1 - 3);

MoveToEx(dc, x0, y0 + 3, NULL);

LineTo(dc, x1, y1 + 3);

}

break;

case 3:

if (vertices[i].prevx == vertices[i].x)

{

MoveToEx(dc, x0 - 5, y0, NULL);

LineTo(dc, x1 - 5, y1);

MoveToEx(dc, x0 + 5, y0, NULL);

LineTo(dc, x1 + 5, y1);

}

else

{

MoveToEx(dc, x0, y0 - 5, NULL);

LineTo(dc, x1, y1 - 5);

MoveToEx(dc, x0, y0 + 5, NULL);

LineTo(dc, x1, y1 + 5);

}

case 1:

MoveToEx(dc, x0, y0, NULL);

LineTo(dc, x1, y1);

break;

}

}

}

DeleteObject(hFont);

}

void SkeletonCanvas::Clear() {

vertices.clear();

RedrawWindow(window, NULL, NULL, RDW\_INVALIDATE | RDW\_UPDATENOW | RDW\_ERASE);

}

void SkeletonCanvas::Save() {

HDC hdc = GetWindowDC(window);

auto memdc = CreateCompatibleDC(hdc);

HBITMAP hBmp = CreateCompatibleBitmap(hdc, 650, 600);

SelectObject(memdc, hBmp);

FloodFill(memdc, 5, 5, RGB(255, 255, 255));

Paint(memdc);

FileManager::Save(hBmp);

DeleteObject(hBmp);

DeleteDC(memdc);

}

LRESULT CALLBACK SkeletonCanvas::WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {

Win32Application\* delegate = Win32Application::Instance();

switch (message)

{

case WM\_CREATE:

{

delegate->SetupChildWindows(hWnd);

break;

}

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case ID\_OPTIONS\_CLEAR:

delegate->GetSkeletonCanvas()->Clear();

break;

case ID\_SAVE2D:

delegate->GetSkeletonCanvas()->Save();

break;

case ID\_OPTIONS\_SAVECALOTTEIMAGE:

delegate->GetCalotteCanvas()->Save();

break;

case ID\_OPTIONS\_CONVERTTOCALOTTE:

delegate->GetSkeletonCanvas()->Convert();

delegate->GetCalotteCanvas()->Render();

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_LBUTTONUP:

{

int x = GET\_X\_LPARAM(lParam);

int y = GET\_Y\_LPARAM(lParam);

delegate->GetSkeletonCanvas()->LeftClick(x, y);

break;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

delegate->GetSkeletonCanvas()->Paint(hdc);

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return DefWindowProc(hWnd, message, wParam, lParam);

}

void SkeletonCanvas::Convert() {

auto delegate = Win32Application::Instance();

auto converter = new Converter();

auto calotteCanvas = delegate->GetCalotteCanvas();

calotteCanvas->SetVertices(converter->ConvertToCalotte(delegate->GetSkeletonCanvas()->GetVertices()));

delete converter;

}

}

Листинг файла Toolbar.h

class Toolbar

{

public:

Toolbar();

HWND GetWindow() { return window; }

void Configure(HWND parent, HINSTANCE hInst);

private:

HWND window;

LPCWSTR szToolsDialogClass = L"Tools Bar";

static LRESULT CALLBACK ToolsProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

};

Листинг файла Toolbar.cpp

#include "Toolbar.h"

#define IDT\_CARBON 30017

#define IDT\_NITROGEN 30018

#define IDT\_OXYGEN 30019

#define IDT\_SULFUR 30020

#define IDT\_PHOSPHORUS 30021

#define IDT\_FLUORINE 30022

#define IDT\_CHLORINE 30023

#define IDT\_BROMINE 30024

#define IDT\_IODINE 30025

#define IDT\_SINGLE 30026

#define IDT\_DOUBLE 30027

#define IDT\_TRIPLE 30028

Toolbar::Toolbar() {

this->window = nullptr;

}

void Toolbar::Configure(HWND parent, HINSTANCE hInst) {

WNDCLASSEX wct;

wct.cbSize = sizeof(WNDCLASSEX);

wct.style = 0;

wct.lpfnWndProc = (WNDPROC)ToolsProc;

wct.cbClsExtra = 0;

wct.cbWndExtra = 0;

wct.hInstance = (HINSTANCE)GetWindowLong(parent, GWL\_HINSTANCE);

wct.hIcon = NULL;

wct.hIconSm = NULL;

wct.hCursor = LoadCursor(NULL, IDC\_ARROW);

wct.hbrBackground = CreateSolidBrush(RGB(245, 245, 245));;

wct.lpszMenuName = NULL;

wct.lpszClassName = szToolsDialogClass;

RegisterClassEx(&wct);

this->window = CreateWindowExW(WS\_EX\_TOOLWINDOW, szToolsDialogClass, L"Tools", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 0, 0, 50, 615, parent, NULL, hInst, NULL);

}

LRESULT CALLBACK Toolbar::ToolsProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {

static HWND hTools[12];

static HBITMAP hBitmap[12];

State\* state = Win32Application::Instance()->GetSkeletonCanvas()->GetState();

switch (message)

{

case WM\_CREATE:

for (int i = 0, y = 0, id = IDT\_CARBON, idb = IDB\_CARBON; i < 12; y += 48, i++, id++, idb++)

{

if (i == 9) { y += 10; }

hTools[i] = CreateWindowW(L"BUTTON", NULL, WS\_CHILD | WS\_VISIBLE | BS\_BITMAP, 1, y, 48, 48,

hWnd, (HMENU)id, (HINSTANCE)GetWindowLong(hWnd, GWL\_HINSTANCE), NULL);

hBitmap[i] = (HBITMAP)LoadImageW((HINSTANCE)GetWindowLong(hWnd, GWL\_HINSTANCE), MAKEINTRESOURCEW(idb), IMAGE\_BITMAP, 48, 48, NULL);

SendDlgItemMessage(hWnd, id, BM\_SETIMAGE, (WPARAM)IMAGE\_BITMAP, (LPARAM)hBitmap[i]);

}

break;

case WM\_COMMAND:

{

switch (LOWORD(wParam))

{

case IDT\_CARBON:

state->SetElement(Element::Carbon);

break;

case IDT\_NITROGEN:

state->SetElement(Element::Nitrogen);

break;

case IDT\_OXYGEN:

state->SetElement(Element::Oxygen);

break;

case IDT\_SULFUR:

state->SetElement(Element::Sulfur);

break;

case IDT\_PHOSPHORUS:

state->SetElement(Element::Phosphorus);

break;

case IDT\_FLUORINE:

state->SetElement(Element::Fluorine);

break;

case IDT\_CHLORINE:

state->SetElement(Element::Chlorine);

break;

case IDT\_BROMINE:

state->SetElement(Element::Bromine);

break;

case IDT\_IODINE:

state->SetElement(Element::Iodine);

break;

case IDT\_SINGLE:

state->SetBond(1);

break;

case IDT\_DOUBLE:

state->SetBond(2);

break;

case IDT\_TRIPLE:

state->SetBond(3);

break;

}

break;

}

case WM\_DESTROY:

for (int i = 0; i < 12; i++)

{

DestroyWindow(hTools[i]);

DeleteObject(hBitmap[i]);

}

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

Листинг файла CalotteCanvas.h

#include <Windows.h>

#include <vector>

#include <string>

#include <gl/gl.h>

#include <gl/glu.h>

#pragma comment( lib, "./OpenGL32.Lib" )

#pragma comment( lib, "./GLU32.Lib" )

#include "Vertex3D.h"

#include "Win32Application.h"

class CalotteCanvas

{

public:

CalotteCanvas();

HWND GetWindow() { return window; }

void Configure(HWND parent, HINSTANCE hInst);

void Zoom(int delta);

void ReleaseResources();

void Render();

void Save();

void SetVertices(std::vector<Vertex3D> vertices) { vertices3D.clear(); vertices3D = vertices; }

private:

void Sphere(Vertex3D& vertex, int pos);

void InitGL();

void SetupContext();

HWND window;

std::vector<GLUquadricObj\*> reusableQuadrics;

LPCWSTR szCanvasClass = L"Calotte Canvas";

const GLfloat light\_ambient[4] = { 0.0f, 0.0f, 0.0f, 1.0f };

const GLfloat light\_diffuse[4] = { 1.0f, 1.0f, 1.0f, 1.0f };

const GLfloat light\_specular[4] = { 1.0f, 1.0f, 1.0f, 1.0f };

const GLfloat light\_position[4] = { 2.0f, 5.0f, 5.0f, 0.0f };

const GLfloat mat\_ambient[4] = { 0.7f, 0.7f, 0.7f, 1.0f };

const GLfloat mat\_diffuse[4] = { 0.8f, 0.8f, 0.8f, 1.0f };

const GLfloat mat\_specular[4] = { 1.0f, 1.0f, 1.0f, 1.0f };

const GLfloat high\_shininess[1] = { 100.0f };

std::vector<Vertex3D> vertices3D;

HGLRC hRC;

HDC hDC;

static LRESULT CALLBACK CanvasProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

float dist = 7;

};

Листинг файла CalotteCanvas.cpp

#include "CalotteCanvas.h"

CalotteCanvas::CalotteCanvas() {

this->window = nullptr;

}

void CalotteCanvas::Configure(HWND parent, HINSTANCE hInst) {

WNDCLASS wc{} ;

wc.style = CS\_HREDRAW | CS\_VREDRAW | CS\_OWNDC;

wc.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wc.lpfnWndProc = CanvasProc;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = NULL;

wc.lpszMenuName = NULL;

wc.hInstance = hInst;

wc.lpszClassName = szCanvasClass;

RegisterClass(&wc);

this->window = CreateWindowW(szCanvasClass, L"CalotteCanvas", WS\_VISIBLE | WS\_CHILD , 655, 0, 615, 615, parent, nullptr, hInst, nullptr);

this->SetupContext();

this->InitGL();

}

void CalotteCanvas::InitGL() {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glClearDepth(1.0f);

glEnable(GL\_DEPTH\_TEST);

glDepthFunc(GL\_LEQUAL);

glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_CULL\_FACE);

glCullFace(GL\_BACK);

glEnable(GL\_DEPTH\_TEST);

glDepthFunc(GL\_LESS);

glEnable(GL\_LIGHT0);

glEnable(GL\_NORMALIZE);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT, GL\_SHININESS, high\_shininess);

glViewport(0, 0, 615, 615);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

float ratio\_w\_h = 1;

gluPerspective(45 , ratio\_w\_h, 0.1 , 200 );

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void CalotteCanvas::SetupContext() {

GLuint PixelFormat;

static PIXELFORMATDESCRIPTOR pfd = // pfd Tells Windows How We Want Things To Be

{

sizeof(PIXELFORMATDESCRIPTOR), // Size Of This Pixel Format Descriptor

1, // Version Number

PFD\_DRAW\_TO\_WINDOW | // Format Must Support Window

PFD\_SUPPORT\_OPENGL | // Format Must Support OpenGL

PFD\_DOUBLEBUFFER, // Must Support Double Buffering

PFD\_TYPE\_RGBA, // Request An RGBA Format

16, // Select Our Color Depth

0, 0, 0, 0, 0, 0, // Color Bits Ignored

0, // No Alpha Buffer

0, // Shift Bit Ignored

0, // No Accumulation Buffer

0, 0, 0, 0, // Accumulation Bits Ignored

16, // 16Bit Z-Buffer (Depth Buffer)

0, // No Stencil Buffer

0, // No Auxiliary Buffer

PFD\_MAIN\_PLANE, // Main Drawing Layer

0, // Reserved

0, 0, 0 // Layer Masks Ignored

};

if (!(hDC = GetDC(window)))

{

ReleaseResources();

return;

}

if (!(PixelFormat = ChoosePixelFormat(hDC, &pfd)))

{

ReleaseResources();

return;

}

if (!SetPixelFormat(hDC, PixelFormat, &pfd))

{

ReleaseResources();

return;

}

if (!(hRC = wglCreateContext(hDC)))

{

ReleaseResources();

return;

}

if (!wglMakeCurrent(hDC, hRC))

{

ReleaseResources();

return;

}

return;

}

void CalotteCanvas::Render() {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear Screen And Depth Buffer

glLoadIdentity();

while (reusableQuadrics.size() < vertices3D.size())

{

GLUquadricObj\* quad = gluNewQuadric();

reusableQuadrics.push\_back(quad);

}

for (unsigned int i = 0; i < vertices3D.size(); i++)

Sphere(vertices3D.at(i), i);

glFlush();

SwapBuffers(hDC);

}

void CalotteCanvas::Sphere(Vertex3D& vertex, int pos)

{

AtomColor color;

float radius;

switch (vertex.element)

{

case Element::Carbon:

color = { 0.57, 0.57, 0.57 };

radius = 1.12;

break;

case Element::Oxygen:

color = { 0.96, 0.07, 0.01 };

radius = 1;

break;

case Element::Nitrogen:

color = { 0.2, 0.33, 0.9255 };

radius = 1.02;

break;

case Element::Phosphorus:

color = { 0.98, 0.5, 0.02 };

radius = 1.18;

break;

case Element::Sulfur:

color = { 1, 0.98, 0.267 };

radius = 1.18;

break;

case Element::Chlorine:

color = { 0.094, 0.964, 0.09 };

radius = 1.15;

break;

case Element::Fluorine:

color = { 0.54, 0.89, 0.325 };

radius = 0.97;

break;

case Element::Bromine:

color = { 0.65, 0.16, 0.15 };

radius = 1.22;

break;

case Element::Iodine:

color = { 0.58, 0, 0.58 };

radius = 1.3;

break;

default:

color = { 1,1,1 };

radius = 0.79;

break;

}

glPushMatrix();

glColor3f(color.red, color.green, color.blue);

glTranslatef(vertex.point.x, vertex.point.y, vertex.point.z - dist);

gluSphere(reusableQuadrics[pos], radius, 30, 30);

glPopMatrix();

}

void CalotteCanvas::Zoom(int delta) {

dist += round(delta / 120.0);

RedrawWindow(window, NULL, NULL, RDW\_INVALIDATE | RDW\_UPDATENOW | RDW\_ERASE);

}

LRESULT CALLBACK CalotteCanvas::CanvasProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {

auto canvas = Win32Application::Instance()->GetCalotteCanvas();

int zDelta;

switch (message) {

case WM\_MOUSEWHEEL:

{

zDelta = GET\_WHEEL\_DELTA\_WPARAM(wParam);

canvas->Zoom(zDelta);

break;

}

case WM\_PAINT:

{

canvas->Render();

break;

}

}

return DefWindowProc(hWnd, message, wParam, lParam);

}

void CalotteCanvas::Save() {

HDC hdc = GetWindowDC(window);

auto memdc = CreateCompatibleDC(hdc);

HBITMAP hBmp = CreateCompatibleBitmap(hdc, 600, 600);

SelectObject(memdc, hBmp);

BitBlt(memdc,0,0,600,600,hdc,0, 0,SRCCOPY);

FileManager::Save(hBmp);

DeleteObject(hBmp);

DeleteDC(memdc);

}

void CalotteCanvas::ReleaseResources() {

if (hRC)

{

wglMakeCurrent(nullptr, nullptr);

wglDeleteContext(hRC);

hRC = nullptr;

}

if (hDC && !ReleaseDC(window, hDC))

{

hDC = nullptr;

} }

Листинг файла Converter.h

#include <vector>

#include <math.h>

#include <stdlib.h>

#include "Vertex2D.h"

#include "Vertex3D.h"

#include "ConverterNode.h"

#include "State.h"

class Converter

{

public:

Converter();

~Converter();

std::vector<Vertex3D> ConvertToCalotte(std::vector<Vertex2D>& vertices2D);

private:

void Tethrahedron(Element vertex);

void Triangle(Element vertex);

void Linear(Element vertex);

bool IsSimple(std::vector<Vertex2D>& vertices2D);

Point Spin(Point& root, float phi, float theta);

void MakeTree(ConverterNode\* root, std::vector<Vertex2D>& vertices2D);

void SortTree(ConverterNode\* root);

void InitCoordinates(ConverterNode\* root);

void ImplementCoordinates(ConverterNode\* root);

void ClearTree(ConverterNode\* root);

void CentrifyCoordinates();

void AdvancedSpin(ConverterNode\* root);

std::vector<Vertex3D> vertices3D;

const double pi = 3.14159265358979323846;

};

Листинг файла Converter.cpp

#include "Converter.h"

#include "Converter.h"

Converter::Converter()

{

}

Converter::~Converter()

{

vertices3D.clear();

}

std::vector<Vertex3D> Converter::ConvertToCalotte(std::vector<Vertex2D>& vertices2D)

{

vertices3D.clear();

if (IsSimple(vertices2D))

return vertices3D;

ConverterNode\* root = new ConverterNode();

root->vertex.element = vertices2D[0].atom.element;

MakeTree(root, vertices2D);

SortTree(root);

InitCoordinates(root);

ClearTree(root);

CentrifyCoordinates();

return vertices3D;

}

void Converter::MakeTree(ConverterNode\* root, std::vector<Vertex2D>& vertices2D)

{

for (int k = root->index + 1; k < vertices2D.size(); k++)

{

if ((vertices2D.at(root->index).x == vertices2D.at(k).prevx) && (vertices2D.at(root->index).y == vertices2D.at(k).prevy))

{

ConverterNode\* daughter = new ConverterNode();

daughter->parent = root;

daughter->index = k;

daughter->vertex.element = vertices2D.at(k).atom.element;

daughter->bond = vertices2D.at(k).atom.bond;

daughter->lastBond = 0;

switch (root->lastBond)

{

case 0:

root->first = daughter;

break;

case 1:

root->second = daughter;

break;

case 2:

root->third = daughter;

break;

default:

root->fourth = daughter;

break;

}

root->lastBond += 1;

if (State::GetMaxBonds(daughter->vertex.element) - daughter->bond > 0)

MakeTree(daughter, vertices2D);

}

}

if (vertices2D.at(root->index).atom.freeBonds)

for (int j = 0; j < vertices2D.at(root->index).atom.freeBonds; j++)

{

ConverterNode\* daughter = new ConverterNode();

daughter->parent = root;

daughter->index = -1;

daughter->vertex.element = Element::Hydrogen;

daughter->bond = 1;

daughter->lastBond = 0;

switch (root->lastBond)

{

case 0:

root->first = daughter;

break;

case 1:

root->second = daughter;

break;

case 2:

root->third = daughter;

break;

default:

root->fourth = daughter;

break;

}

root->lastBond++;

}

}

void Converter::InitCoordinates(ConverterNode\* root)

{

root->vertex.point = { 0, 0, 0 };

if (root->first->bond == 3)

{

root->first->vertex.point = { 0, 0.8, 0 };

if (root->second != nullptr)

root->second->vertex.point = { 0, -0.8, 0 };

}

else

AdvancedSpin(root);

if (root->first != nullptr)

ImplementCoordinates(root->first);

if (root->second != nullptr)

ImplementCoordinates(root->second);

if (root->third != nullptr)

ImplementCoordinates(root->third);

if (root->fourth != nullptr)

ImplementCoordinates(root->fourth);

}

void Converter::ImplementCoordinates(ConverterNode\* root)

{

if (root->lastBond == 0)

return;

if ((root->first->bond == 3) || ((root->first->bond == 2) && (root->bond == 2)) || (root->bond == 3))

{

root->first->vertex.point.x = 2 \* root->vertex.point.x - root->parent->vertex.point.x;

root->first->vertex.point.y = 2 \* root->vertex.point.y - root->parent->vertex.point.y;

root->first->vertex.point.z = 2 \* root->vertex.point.z - root->parent->vertex.point.z;

}

else

AdvancedSpin(root);

if (root->first != nullptr)

ImplementCoordinates(root->first);

if (root->second != nullptr)

ImplementCoordinates(root->second);

if (root->third != nullptr)

ImplementCoordinates(root->third);

}

void Converter::SortTree(ConverterNode\* root)

{

switch (root->lastBond)

{

case 2:

if (root->first->bond < root->second->bond)

{

ConverterNode\* temp = root->second;

root->second = root->first;

root->first = temp;

}

break;

case 3:

if (root->second->bond > root->first->bond)

{

ConverterNode\* temp = root->second;

root->second = root->first;

root->first = temp;

}

else if (root->third->bond > root->second->bond)

{

ConverterNode\* temp = root->third;

root->third = root->second;

root->second = temp;

}

break;

}

if (root->first != nullptr)

SortTree(root->first);

if (root->second != nullptr)

SortTree(root->second);

if (root->third != nullptr)

SortTree(root->third);

if (root->fourth != nullptr)

SortTree(root->fourth);

}

void Converter::ClearTree(ConverterNode\* root)

{

if (root->first != nullptr)

ClearTree(root->first);

if (root->second != nullptr)

ClearTree(root->second);

if (root->third != nullptr)

ClearTree(root->third);

if (root->fourth != nullptr)

ClearTree(root->fourth);

vertices3D.push\_back(root->vertex);

delete root;

}

Point Converter::Spin(Point& root, float phi, float theta)

{

Point result{};

float r = 0.8;

result.z = r \* cos(theta) \* sin(phi);

result.x = r \* sin(theta) \* sin(phi);

result.y = r \* cos(phi);

return result;

}

void Converter::AdvancedSpin(ConverterNode\* root)

{

const float r = 0.8;

if (root->parent == nullptr)

{

switch (root->lastBond)

{

case 4:

root->fourth->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(0);

root->fourth->vertex.point.y = root->vertex.point.y + r \* cos(0);

root->fourth->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(0);

case 3:

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(2.0944);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(2.0944);

case 2:

root->second->vertex.point.x = root->vertex.point.x + r \* sin(4.18879) \* sin(2.0944);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(4.18879) \* sin(2.0944);

case 1:

root->first->vertex.point.x = root->vertex.point.x + r \* sin(2.0944) \* sin(2.0944);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(2.0944) \* sin(2.0944);

break;

}

}

else

{

if ((round(root->vertex.point.x \* 100) == round(root->parent->vertex.point.x \* 100)) && (round(root->vertex.point.z \* 100) == round(root->parent->vertex.point.z \* 100)))

{

if (root->vertex.point.y > root->parent->vertex.point.y)

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(3.14159) \* sin(1.0472);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(3.14159) \* sin(1.0472);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(-1.0472) \* sin(1.0472);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(-1.0472) \* sin(1.0472);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(1.0472) \* sin(1.0472);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(1.0472) \* sin(1.0472);

}

return; } else

{ if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(2.0944);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(2.0944);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(4.18879) \* sin(2.0944);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(4.18879) \* sin(2.0944);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(2.0944) \* sin(2.0944);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(2.0944) \* sin(2.0944);

}

return; } }

else

{

if ((root->vertex.point.x < root->parent->vertex.point.x) && (root->vertex.point.y < root->parent->vertex.point.y) && (root->vertex.point.z < root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(3.14159);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(3.14159);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(3.14159);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(3.14159) \* sin(1.0472);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(3.14159) \* sin(1.0472);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(-1.0472) \* sin(1.0472);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(-1.0472) \* sin(1.0472);

}

return;

}

if ((root->vertex.point.x > root->parent->vertex.point.x) && (root->vertex.point.y < root->parent->vertex.point.y) && (root->vertex.point.z < root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(3.14159);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(3.14159);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(3.14159);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(3.14159) \* sin(1.0472);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(3.14159) \* sin(1.0472);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(1.0472) \* sin(1.0472);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(1.0472) \* sin(1.0472); } return; }

if ((root->vertex.point.x < root->parent->vertex.point.x) && (root->vertex.point.y > root->parent->vertex.point.y) && (root->vertex.point.z > root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(0);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(0);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(0);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(2.0944);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(2.0944);

}

if (root->first != nullptr)

{root->first->vertex.point.x = root->vertex.point.x + r \* sin(4.18879) \* sin(2.0944);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(4.18879) \* sin(2.0944);

} return;

}

if ((root->vertex.point.x > root->parent->vertex.point.x) && (root->vertex.point.y > root->parent->vertex.point.y) && (root->vertex.point.z > root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(0);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(0);root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(0);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(2.0944);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);root->second->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(2.0944);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(2.0944) \* sin(2.0944);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(2.0944) \* sin(2.0944);

}

return;

}

if ((round(root->vertex.point.x \* 100) == round(root->parent->vertex.point.x \* 100)) && (root->vertex.point.y < root->parent->vertex.point.y) && (root->vertex.point.z > root->parent->vertex.point.z))

{

if (root->third != nullptr) {

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(3.14159);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(3.14159);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(3.14159);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(1.0472) \* sin(1.0472);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(1.0472) \* sin(1.0472);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(-1.0472) \* sin(1.0472);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(-1.0472) \* sin(1.0472);

}

return;

}

if ((round(root->vertex.point.x \* 100) == round(root->parent->vertex.point.x \* 100)) && (root->vertex.point.y > root->parent->vertex.point.y) && (root->vertex.point.z < root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(0);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(0);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(0);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(4.18879) \* sin(2.0944);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(4.18879) \* sin(2.0944);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(2.0944) \* sin(2.0944);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(2.0944) \* sin(2.0944);

}

return;

}

if ((round(root->vertex.point.x \* 100) == round(root->parent->vertex.point.x \* 100)) && (root->vertex.point.y < root->parent->vertex.point.y) && (root->vertex.point.z < root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(3.14159);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(3.14159);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(3.14159);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(2.0944) \* sin(1.0472);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(2.0944) \* sin(1.0472);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(4.18879) \* sin(1.0472);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(4.18879) \* sin(1.0472);

}

return;

}

if ((round(root->vertex.point.x \* 100) == round(root->parent->vertex.point.x \* 100)) && (root->vertex.point.y > root->parent->vertex.point.y) && (root->vertex.point.z > root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(0);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(0);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(0);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(1.0472) \* sin(1.0472);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(1.0472) \* sin(1.0472);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(-1.0472) \* sin(1.0472);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(-1.0472) \* sin(1.0472);

}

return;

}

if ((root->vertex.point.x < root->parent->vertex.point.x) && (root->vertex.point.y > root->parent->vertex.point.y) && (root->vertex.point.z < root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(0);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(0);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(0);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(3.14159) \* sin(2.0944);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(3.14159) \* sin(2.0944);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(-1.0472) \* sin(2.0944);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(-1.0472) \* sin(2.0944);

}

return;

}

if ((root->vertex.point.x > root->parent->vertex.point.x) && (root->vertex.point.y > root->parent->vertex.point.y) && (root->vertex.point.z < root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(0);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(0);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(0);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(3.14159) \* sin(2.0944);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(3.14159) \* sin(2.0944);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(1.0472) \* sin(2.0944);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(2.0944);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(1.0472) \* sin(2.0944);

}

return;

}

if ((root->vertex.point.x < root->parent->vertex.point.x) && (root->vertex.point.y < root->parent->vertex.point.y) && (root->vertex.point.z > root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(3.14159);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(3.14159);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(3.14159);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(4.18879) \* sin(1.0472);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(4.18879) \* sin(1.0472);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(1.0472);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(1.0472);

}

return;

}

if ((root->vertex.point.x > root->parent->vertex.point.x) && (root->vertex.point.y < root->parent->vertex.point.y) && (root->vertex.point.z > root->parent->vertex.point.z))

{

if (root->third != nullptr)

{

root->third->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(3.14159);

root->third->vertex.point.y = root->vertex.point.y + r \* cos(3.14159);

root->third->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(3.14159);

}

if (root->second != nullptr)

{

root->second->vertex.point.x = root->vertex.point.x + r \* sin(2.0944) \* sin(1.0472);

root->second->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->second->vertex.point.z = root->vertex.point.z + r \* cos(2.0944) \* sin(1.0472);

}

if (root->first != nullptr)

{

root->first->vertex.point.x = root->vertex.point.x + r \* sin(0) \* sin(1.0472);

root->first->vertex.point.y = root->vertex.point.y + r \* cos(1.0472);

root->first->vertex.point.z = root->vertex.point.z + r \* cos(0) \* sin(1.0472);

}

return;

}

}

}

}

void Converter::Tethrahedron(Element vertex)

{

bool top = false;

float degree;

switch (vertex)

{

case Element::Carbon:

top = true;

degree = 1.9111355; //109.5

break;

case Element::Phosphorus:

degree = 1.623156; //93.5

break;

case Element::Nitrogen:

default:

degree = 1.8675023; //107.8

break;

}

Point root, \_1, \_2, \_3, \_4;

if (top)

{

root = { 0, 0, 0 };

\_1 = Spin(root, 0, 0);

\_2 = Spin(root, degree, 2.0944);

\_3 = Spin(root, degree, 4.18879);

\_4 = Spin(root, degree, 0);

}

else

{

root = { 0, 0, 0 };

degree = cos(degree / 2);

degree = pi - acos(sqrt(1 - 4 / 3 \* degree \* degree));

\_1 = Spin(root, degree, 2.0944);

\_2 = Spin(root, degree, 4.18879);

\_3 = Spin(root, degree, 0);

}

Vertex3D \_point = { root, vertex };

vertices3D.push\_back(\_point);

\_point.point = \_1;

\_point.element = Element::Hydrogen;

vertices3D.push\_back(\_point);

\_point.point = \_2;

vertices3D.push\_back(\_point);

\_point.point = \_3;

vertices3D.push\_back(\_point);

if (top)

{

\_point.point = \_4;

vertices3D.push\_back(\_point);

}

}

void Converter::CentrifyCoordinates()

{

float x, y, z;

x = y = z = 0;

for (unsigned int i = 0; i < vertices3D.size(); i++)

{

x += vertices3D[i].point.x;

y += vertices3D[i].point.y;

z += vertices3D[i].point.z;

}

x /= vertices3D.size();

y /= vertices3D.size();

z /= vertices3D.size();

for (unsigned int i = 0; i < vertices3D.size(); i++)

{

vertices3D[i].point.x -= x;

vertices3D[i].point.y -= y;

vertices3D[i].point.z -= z;

}

}

void Converter::Triangle(Element vertex)

{

float degree;

switch (vertex)

{

case Element::Sulfur:

degree = 1.605703;

break;

default:

degree = 1.81514242;

}

Vertex3D root = { {0, 0, 0}, vertex };

vertices3D.push\_back(root);

float r = 0.8;

Point \_1{};

\_1.y = -r \* cos(degree / 2);

\_1.z = 0;

\_1.x = r \* sin(degree / 2);

root.point = \_1;

root.element = Element::Hydrogen;

vertices3D.push\_back(root);

Point \_2;

\_2 = \_1;

\_2.x = -\_2.x;

root.point = \_2;

vertices3D.push\_back(root);

}

void Converter::Linear(Element vertex)

{

Vertex3D \_1 = { {-0.4, 0, 0}, vertex };

vertices3D.push\_back(\_1);

Vertex3D \_2 = { {0.4, 0, 0}, Element::Hydrogen };

vertices3D.push\_back(\_2);

}

bool Converter::IsSimple(std::vector<Vertex2D>& vertices2D)

{

if (vertices2D.size() == 1)

{

switch (vertices2D[0].atom.element)

{

case Element::Carbon:

case Element::Phosphorus:

case Element::Nitrogen:

Tethrahedron(vertices2D[0].atom.element);

break;

case Element::Sulfur:

case Element::Oxygen:

Triangle(vertices2D[0].atom.element);

break;

default:

Linear(vertices2D[0].atom.element);

}

return true;

}

return false;

}

Листинг файла FileManager.h

#pragma once

#include <Windows.h>

#include "Win32Application.h"

#include <assert.h>

class FileManager

{

public:

static void Save(HBITMAP bitmap);

private:

static void CreateBMPFile(LPTSTR pszFile, HBITMAP hBMP);

static PBITMAPINFO CreateBitmapInfoStruct(HBITMAP hBmp);

};

Листинг файла FileManager.cpp

#include "FileManager.h"

void FileManager::Save(HBITMAP bitmap) {

OPENFILENAME ofn;

TCHAR szFileName[MAX\_PATH] = L"";

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = Win32Application::Instance()->GetSkeletonCanvas()->GetWindow();

ofn.lpstrFile = szFileName;

ofn.nMaxFile = sizeof(szFileName);

ofn.lpstrFilter = TEXT("Bitmap(\*.bmp)\0\*.bmp\0Все файлы(\*.\*)\0\*.\*\0");

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

ofn.Flags = OFN\_EXPLORER | OFN\_PATHMUSTEXIST | OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT;

ofn.lpstrTitle = TEXT("Select destination path");

ofn.lpstrDefExt = TEXT("bmp");

if (GetSaveFileName(&ofn) == TRUE) {

CreateBMPFile(ofn.lpstrFile, bitmap);

}

}

void FileManager::CreateBMPFile(LPTSTR pszFile, HBITMAP hBMP)

{

HANDLE hf; // file handle

BITMAPFILEHEADER hdr; // bitmap file-header

PBITMAPINFOHEADER pbih; // bitmap info-header

LPBYTE lpBits; // memory pointer

DWORD dwTotal; // total count of bytes

DWORD cb; // incremental count of bytes

BYTE\* hp; // byte pointer

DWORD dwTmp;

PBITMAPINFO pbi;

HDC hDC;

hDC = CreateCompatibleDC(GetWindowDC(GetDesktopWindow()));

SelectObject(hDC, hBMP);

pbi = CreateBitmapInfoStruct(hBMP);

pbih = (PBITMAPINFOHEADER)pbi;

lpBits = (LPBYTE)GlobalAlloc(GMEM\_FIXED, pbih->biSizeImage);

assert(lpBits);

assert(GetDIBits(hDC, hBMP, 0, (WORD)pbih->biHeight, lpBits, pbi,

DIB\_RGB\_COLORS));

hf = CreateFile(pszFile,

GENERIC\_READ | GENERIC\_WRITE,

(DWORD)0,

NULL,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

(HANDLE)NULL);

assert(hf != INVALID\_HANDLE\_VALUE);

hdr.bfType = 0x4d42;

hdr.bfSize = (DWORD)(sizeof(BITMAPFILEHEADER) +

pbih->biSize + pbih->biClrUsed

\* sizeof(RGBQUAD) + pbih->biSizeImage);

hdr.bfReserved1 = 0;

hdr.bfReserved2 = 0;

hdr.bfOffBits = (DWORD)sizeof(BITMAPFILEHEADER) +

pbih->biSize + pbih->biClrUsed

\* sizeof(RGBQUAD);

assert(WriteFile(hf, (LPVOID)&hdr, sizeof(BITMAPFILEHEADER),

(LPDWORD)&dwTmp, NULL));

assert(WriteFile(hf, (LPVOID)pbih, sizeof(BITMAPINFOHEADER)

+ pbih->biClrUsed \* sizeof(RGBQUAD),

(LPDWORD)&dwTmp, (NULL)));

dwTotal = cb = pbih->biSizeImage;

hp = lpBits;

assert(WriteFile(hf, (LPSTR)hp, (int)cb, (LPDWORD)&dwTmp, NULL));

assert(CloseHandle(hf));

// Free memory.

GlobalFree((HGLOBAL)lpBits);

}

PBITMAPINFO FileManager::CreateBitmapInfoStruct(HBITMAP hBmp)

{

BITMAP bmp;

PBITMAPINFO pbmi;

WORD cClrBits;

assert(GetObject(hBmp, sizeof(BITMAP), (LPSTR)&bmp));

cClrBits = (WORD)(bmp.bmPlanes \* bmp.bmBitsPixel);

if (cClrBits == 1)

cClrBits = 1;

else if (cClrBits <= 4)

cClrBits = 4;

else if (cClrBits <= 8)

cClrBits = 8;

else if (cClrBits <= 16)

cClrBits = 16;

else if (cClrBits <= 24)

cClrBits = 24;

else cClrBits = 32;

if (cClrBits < 24)

pbmi = (PBITMAPINFO)LocalAlloc(LPTR,

sizeof(BITMAPINFOHEADER) +

sizeof(RGBQUAD) \* (1 << cClrBits));

else

pbmi = (PBITMAPINFO)LocalAlloc(LPTR,

sizeof(BITMAPINFOHEADER));

pbmi->bmiHeader.biSize = sizeof(BITMAPINFOHEADER);

pbmi->bmiHeader.biWidth = bmp.bmWidth;

pbmi->bmiHeader.biHeight = bmp.bmHeight;

pbmi->bmiHeader.biPlanes = bmp.bmPlanes;

pbmi->bmiHeader.biBitCount = bmp.bmBitsPixel;

if (cClrBits < 24)

pbmi->bmiHeader.biClrUsed = (1 << cClrBits);

pbmi->bmiHeader.biCompression = BI\_RGB;

pbmi->bmiHeader.biSizeImage = ((pbmi->bmiHeader.biWidth \* cClrBits + 31) & ~31) / 8

\* pbmi->bmiHeader.biHeight;

pbmi->bmiHeader.biClrImportant = 0;

return pbmi;

}

Листинг файла State.h

#include "Element.h"

struct State

{

public:

int GetBond() { return this->bond; }

void SetBond(int bond);

Element GetElement() { return this->element; }

void SetElement(Element element);

static int GetMaxBonds(Element element);

private:

int bond = 1;

Element element = Element::Carbon;

};

Листинг файла State.cpp

#include "State.h"

void State::SetBond(int bond) {

if (GetMaxBonds(element) >= bond)

this->bond = bond;

}

void State::SetElement(Element element) {

this->element = element;

if (bond > GetMaxBonds(element))

bond = 1;

}

int State::GetMaxBonds(Element element) {

switch (element)

{

case Element::Carbon:

return 4;

case Element::Nitrogen:

case Element::Phosphorus:

return 3;

case Element::Oxygen:

case Element::Sulfur:

return 2;

default:

return 1;

}

}

Листинг файла Atom.h

#include "Element.h"

struct Atom

{

Element element;

int freeBonds;

int bond;

};

Листинг файла AtomColor.h

struct AtomColor

{

float red;

float green;

float blue;

};

Листинг файла Point.h

struct Point

{

float x;

float y;

float z;

};

Листинг файла Vertex2D.h

#pragma once

#include "Atom.h"

#include <string>

struct Vertex2D

{

int x;

int y;

int prevx;

int prevy;

bool shortRear;

bool shortFront;

Atom atom;

std::wstring Description() {

std::wstring vertex = L"";

if (atom.freeBonds > 0)

{

vertex = L"H";

if (atom.freeBonds > 1)

vertex += std::to\_wstring(atom.freeBonds);

}

switch (atom.element)

{

case Element::Chlorine:

vertex += L"Cl";

break;

case Element::Bromine:

vertex += L"Br";

break;

case Element::Fluorine:

vertex += L" F";

break;

case Element::Iodine:

vertex += L" I";

break;

case Element::Sulfur:

vertex += L"S";

break;

case Element::Oxygen:

if (atom.freeBonds != 1)

vertex += L"O";

else

vertex = L"O" + vertex;

break;

case Element::Phosphorus:

vertex = L"P" + vertex;

break;

case Element::Nitrogen:

vertex = L"N" + vertex;

break;

case Element::Carbon:

vertex = L"C" + vertex;

break;

}

return vertex;

}

};

Листинг файла Vertex3D

#include "Point.h"

#include "Element.h"

struct Vertex3D

{

Point point;

Element element;

};

Листинг файла Element.h

#pragma once

enum class Element {

Carbon,

Nitrogen,

Oxygen,

Phosphorus,

Sulfur,

Fluorine,

Chlorine,

Bromine,

Iodine,

Hydrogen

};

Листинг файла ConverterNode.h

#pragma once

#include "Vertex3D.h"

struct ConverterNode

{

ConverterNode\* first = nullptr;

ConverterNode\* second = nullptr;

ConverterNode\* third = nullptr;

ConverterNode\* fourth = nullptr;

ConverterNode\* parent = nullptr;

Vertex3D vertex{};

int index = 0;

int lastBond = 0;

int bond = 0;

};

ВЕДОМОСТЬ

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1–40 01 01 204 ПЗ | | | | Пояснительная записка | | | | 80 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951002 204 СП | | | | Схема программы Программное средство «Молекулярный редактор» | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КР 1-40 01 01 204 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Программное средство «Молекулярный редактор»  Ведомость курсовой  работы |  | | | | Лист | Листов |
| Разраб. | | Будович И.В. |  | 20.12.21 | Т |  | |  | 80 | 80 |
| Пров. | | Степанцов Е.В. |  | 20.12.21 | Кафедра ПОИТ  гр. 951002 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |