МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Воронежский государственный технический университет»

Кафедра компьютерных интеллектуальных технологий проектирования

6-2024

ГЕОМЕТРИЧЕСКИЕ ПОСТРОЕНИЯ В 3D

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ
по дисциплине «Геометрическое моделирование»
для студентов направления
09.03.01 «Информатика и вычислительная техника»
очной и заочной форм обучения

Воронеж 2024

УДК 004.42/.65
ББК 32.973

Составитель

канд. техн. наук А. Н. Юров

Геометрические построения в 3D: методические указания к выполнению лабораторных работ по дисциплине «Геометрическое моделирование» для студентов направления 09.03.01 «Информатика и вычислительная техника» очной и заочной формы обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост. А. Н. Юров. – Воронеж, 2024. – 28 с.

Методические указания будут полезны для изучения курса «Геометрическое моделирование» и выполнения лабораторных работ по указанной дисциплине.

Предназначены для студентов направления 09.03.01 «Информатика и вычислительная техника» очной и заочной формы обучения.

Методические указания подготовлены в электронном виде и содержатся в файле MU_GeomMod3D.pdf.

Ил. 22. Библиогр.: 5 назв.

УДК 004.42/.65
ББК 32.973

Рецензент – *М. Н. Подоприхин, канд. техн. наук, доцент, проф. кафедры инженерной и компьютерной графики ВГТУ*

Издаётся по решению редакционно-издательского совета Воронежского государственного технического университета

ВВЕДЕНИЕ

Геометрическое моделирование - процесс создания и манипулирования геометрическими объектами с помощью компьютерных программ. Широко применяется в различных областях, таких как компьютерная графика, компьютерное зрение, робототехника и многих других.

Для реализации геометрического моделирования в C++ можно использовать различные библиотеки и инструменты. В методических указаниях используется библиотека с геометрическим ядром Open Cascade. Все работы связаны с моделированием 3D моделей, операций над ними и функциональными возможностями, которые могут быть полезны при разработке программных решений в области САПР.

Для освоения лабораторного практикума по курсу «Геометрическое моделирование» в работах требуется выполнить следующие задачи:

-ознакомится с целью и задачами к лабораторной работе;

-изучить предлагаемый теоретический материал, а также ссылки на дополнительные источники;

-разработать программное средство согласно заданию к лабораторной работе;

-провести тестирование разработанного программного средства;

-продемонстрировать функциональные возможности разработанного приложения (в интегрированной среде разработки и / или на «на чистой» операционной системе без установленных пакетных зависимостей).

Разработка приложений должна быть реализована на языке программирования C++. Выбор интегрированных сред разработки (IDE) и операционной системы должен быть сделан студентом самостоятельно.

Требования к приложению:

- программное средство должно работать под заданную ОС и не требовать при запуске установки дополнительных зависимостей (выполнять все требуемые функции);

-включать графический интерфейс;

-включать типовые элементы управления на обработку событий в приложении;

-применение функциональных возможностей последних стандартов языка C++ 17, 20, 23.

ЛАБОРАТОРНАЯ РАБОТА № 1
ГЕОМЕТРИЧЕСКИЕ 3D ПРИМИТИВЫ В OPEN CASCADE

Цель работы: построение простых твердотельных моделей Open Cascade.

Задачи и требования к выполнению:

1. Изучить классы простых 3D объектов Open Cascade.

2. Ознакомиться с примерами из документации по построению простых моделей на Open Cascade.

3. Собрать и подготовить к выполнению программное решение.

Теоретические сведения

Моделирование трёхмерных примитивов и выполнение над ними булевых операций является одним из основных методов моделирования деталей. Так же на основе трёхмерных примитивов можно создавать конструктивные элементы, такие как отверстие, паз, бобышка. С помощью ядра Open Cascade можно создавать 5 типов трёхмерных примитивов: блок (куб), цилиндр, конус, сфера и тор.

Для создания примитивов необходимо настроить свойства проекта: для в проект добавляются 3 статические библиотеки Open Cascade в файл CmakeLists.txt: TKTopAlgo.lib, TKPrim.lib, TKBRep.lib.

Также для заданного типа примитива в состав программы необходимо включить определённый заголовочный файл:

BRepPrimAPI_MakeBox- для создания блока (куб);

BRepPrimAPI_MakeCylinder - для создания цилиндра;

BRepPrimAPI_MakeCone - для создания конуса;

BRepPrimAPI_MakeSphere - для создания сферы;

BRepPrimAPI_MakeTorus - для создания тора.

Один и тот же примитив может быть создан разными способами, в качестве параметров примитивов могут быть заданы начальные и конечные точки, ось вращения, система координат, направления оси. Для задания точек необходимо включить заголовочный файл gp_Pnt, вектор направления оси задаётся с помощью заголовочного файла gp_Dir, с помощью заголовочного файла gp_Ax2 можно задать вектор направления оси в определённой точки.

Для отображения результата необходимо воспользоваться сервисом AIS (Application Interactive Services), который связывает структуру данных с

отображаемым объектом. Сначала требуется получить объект представления детали:

```
Handle (AIS_Shape) aAISShape = new AIS_Shape(cyl_1);
```

Следом задаётся материал «Медь» и режим отображения объекта «Закраска», а также полупрозрачность:

```
aAISShape->SetMaterial(Graphic3d_NOM_COPPER);
aAISShape->SetDisplayMode(AIS_Shaded);
aAISShape->SetTransparency(0.7);
myIC->Display(aAISShape);
```

Листинг приложения по построению цилиндра представлен следующим листингом:

```
#include <AIS_Shape.hxx>
#include <TopoDS_Shape.hxx>
#include <BRepPrimAPI_MakeCylinder.hxx>
#include <gp_Pnt.hxx>
#include <gp_Dir.hxx>
#include <gp_Ax2.hxx>
void occQt::buildPrim3D()
{
   // создание цилиндра
   TopoDS_Shape cyl_1 = BrepPrimAPI_MakeCylinder
(gp_Ax2(gp_Pnt(0, 0, 0),gp_Dir(0, 0, 1)), 50, 250,
(270 * M_PI / 180)).Shape();
   Handle (AIS_Shape) aAISShape = new AIS_Shape(cyl_1);
   aAISShape->SetMaterial(Graphic3d_NOM_COPPER);
   aAISShape->SetDisplayMode(AIS_Shaded);
   aAISShape->SetTransparency(0.7);
   myOccView->getContext()->Display(aAISShape, true );
```

Результаты работы по построению цилиндра показаны на рис. 1.

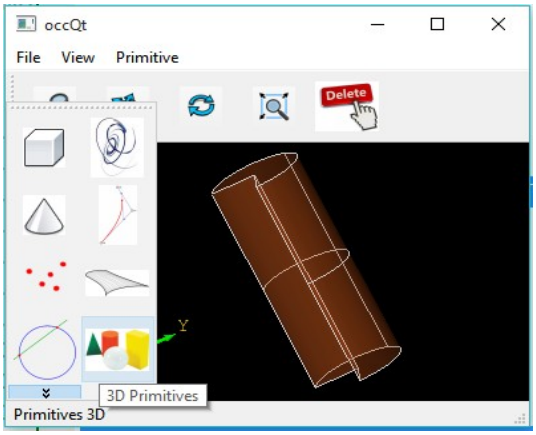

Рис. 1. Построение цилиндра с вырезом фрагмента

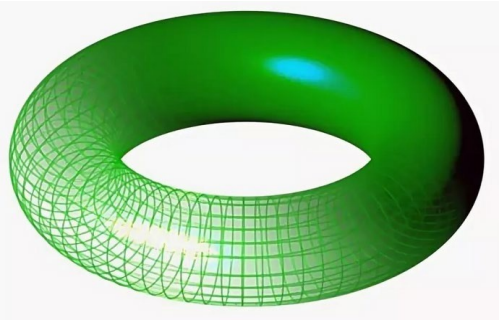

Рис. 2. Модель тора

Задания для самостоятельной работы:

1. Создать блок с помощью 2-х диагональных точек. Координаты 1-ой точки (0, 0, 0), 2-ой точки (150, 100, 150).

2. Создать тор (рис. 2) по 2-м радиусам с углом. Радиус = 150, радиус сечения = 5, угол = 180.

ЛАБОРАТОРНАЯ РАБОТА № 2
МОДЕЛИРОВАНИЕ ТЕЛ С ПОМОЩЬЮ БУЛЕВЫХ ОПЕРАЦИЙ

Цель работы: построение твердотельных моделей посредством булевых операций.

Задачи и требования к выполнению:

1. Изучить классы булевых операций.

2. Ознакомиться с примерами из документации на Open Cascade.

3. Собрать и подготовить к выполнению программное решение.

Теоретические сведения

С помощью набора операций ядра Open Cascade можно быстро создавать сложные трёхмерные тела. Также можно вносить изменения в полученное тело, редактируя его параметры или напрямую изменяя размеры и положения граней, при необходимости редактирования импортированной геометрии можно использовать технику прямого моделирования. Твёрдое тело может быть создано путём построения примитивных (параллелепипед, конус, цилиндр, и т.д.) элементов формы и их объединением, вычитанием или пересечением и последующим добавлением к детали. При работе с примитивами каждая

отдельная операция порождает простую геометрию, в принципе, можно построить тело, что и в первом случае, однако его редактирование может оказаться более трудоёмким, но и более гибким и предсказуемым. Добавляют возможности по построению моделей булевы операции.

Булевы операции включают в себя: объединение двух и более тел, вычитание одного тела из другого, пересечение двух тел.

Подход к моделированию тел в Open Cascade. Необходимо добавить заголовочные файлы проектное решение.

```
//булевая операция вычитания
 include<BRepAlgoAPI_Cut.hxx>
//булевая операция объединения
include< BRepAlgoAPI_Fuse.hxx>
```

После чего для уже созданных моделей будет доступен функционал для использования операций объединения и вычитания. В следующем фрагменте программного кода показано, как построить модель основания, изображённой на рис. 3.

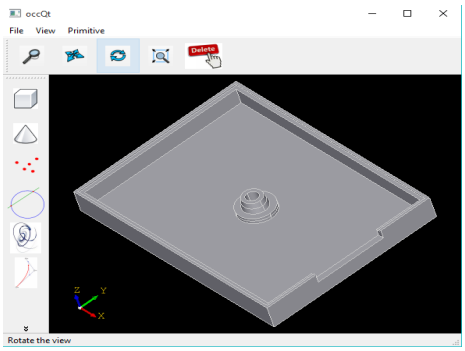

Рис. 3. Модель основания

```
   TopoDS_Shape box_1 = BrepPrimAPI_MakeBox
(gp_Pnt(0, 0, 0), 100, 75, 12).Shape();
     TopoDS_Shape box_2 = BrepPrimAPI_MakeBox
(gp_Pnt(2.5, 2.5, 2.5), 95, 70, 9.5).Shape();
     TopoDS_Shape ShapeCut_1 = BRepAlgoAPI_Cut(box_1, box_2);
     TopoDS_Shape cyl_1 = BrepPrimAPI_MakeCylinder
(gp_Ax2(gp_Pnt(50, 37.5, 2.5), gp_Dir(0, 0, 1)), 7.5, 3).Shape();
     TopoDS_Shape ShapeFuse_1 = BRepAlgoAPI_Fuse(ShapeCut_1, cyl_1);
     TopoDS_Shape conus = BrepPrimAPI_MakeCone
(gp_Ax2(gp_Pnt(50, 37.5, 5.5), gp_Dir(0, 0, 1)), 5, 4, 5.5).Shape();
     TopoDS_Shape ShapeFuse_2 = BRepAlgoAPI_Fuse(ShapeFuse_1, conus);
     TopoDS_Shape cyl_2 = BrepPrimAPI_MakeCylinder
(gp_Ax2(gp_Pnt(50, 37.5, 11), gp_Dir(0, 0, -1)), 2.5, 11).Shape();
     TopoDS_Shape ShapeCut_2 = BRepAlgoAPI_Cut(ShapeFuse_2, cyl_2);
```

```
    TopoDS_Shape box_3 = BrepPrimAPI_MakeBox
(gp_Pnt(97.5, 22.5, 7.5), 2.5, 30, 4.5).Shape();
    TopoDS_Shape ShapeCut_3 = BRepAlgoAPI_Cut(ShapeCut_2, box_3);
    TopoDS_Shape box_4 = BrepPrimAPI_MakeBox
(gp_Ax2(gp_Pnt(1, 1, 11), gp_Dir(0, 0, 1)), 98, 73, 1).Shape();
    TopoDS_Shape ShapeCut_4 = BRepAlgoAPI_Cut(ShapeCut_3, box_4);
    Handle (AIS_Shape) aAISShape = new AIS_Shape(ShapeCut_4);
    aAISShape->SetMaterial(Graphic3d_NOM_CHROME);
    aAISShape->SetDisplayMode(AIS_Shaded);
    myOccView->getContext()->Display(aAISShape, true );
```

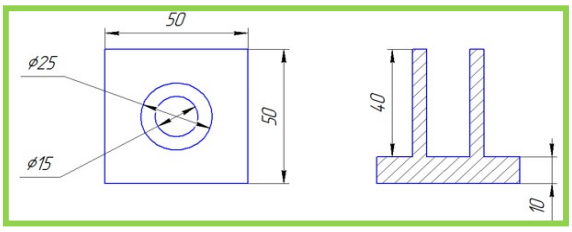

Рис. 4. Эскиз платформы

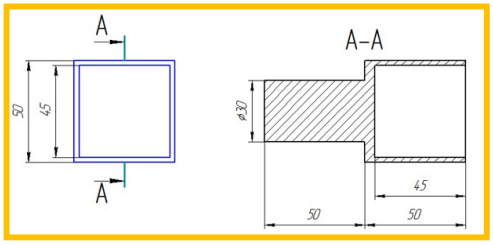

Рис. 5. Эскиз стойки

Задания для самостоятельной работы:

1. Выполнить построение моделей, показанных на рис. 4 и 5 средствами Open Cascade.

2. Добавить параметризацию в созданные модели посредством интерфейсного диалога на ImGui.

ЛАБОРАТОРНАЯ РАБОТА № 3
ПОСТРОЕНИЕ КОНСТРУКТИВНЫХ ЭЛЕМЕНТОВ В МОДЕЛИ

Цель работы: построение фасок и скруглений.

Задачи и требования к выполнению:

1. Изучить классы конструктивных операций.

2. Ознакомиться с примерами из документации на Open Cascade.

3. Собрать и подготовить к выполнению программное решение.

Теоретические сведения

Конструктивные элементы обработки влияют на существующие ребра или грани модели. С помощью ядра Open Cascade можно строить конструктивные элементы:

-скругление с постоянным радиусом для одного или нескольких рёбер модели;

-скругление с переменным радиусом для одного или нескольких рёбер модели;

-фаска для одного или нескольких рёбер модели;

-уклон существующей грани модели на заданный угол относительно базовой плоскости;

-«тонкостенное тело», часто применяется для конструирования корпусов и деталей из пластика.

По возможности не следует создавать скругления и фаски в эскизах, даже если это необходимо для построения сечения элемента модели. Лучше всего добавлять конструктивные элементы непосредственно на 3D модель в самую последнюю очередь. Это даёт следующие преимущества:

- требуется меньшее количество элементов эскиза;

- отдельно созданные скругления и фаски могут быть «подавлены», изменены или удалены без особых усилий для перестроения модели;

- геометрия сечений для кинематических элементов проще, требуется меньшее время для пересчёта модели;

- проще решить проблемы топологии, если скругления и фаски выполнены в отдельных элементах.

С помощью операции «Скругление» можно сгладить острые края модели. Скругление можно применить только на ребра. Построить фаску можно между двумя гранями вдоль общего ребра. В большинстве случаев фаска добавляется в деталь уже на конечном этапе моделирования. В деталях не следует строить фаски при создании профиля модели. Если добавить фаски на завершающем этапе, как операции обработки, то внесение изменений будет эффективным и простым действием.

Построение скругления (рис. 6):

-Создаётся массив и заполняется рёбрами модели:

```
TopTools_IndexedMapOfShape edge; // массив ребер
TopExp::MapShapes(box, TopAbs_EDGE, edge);
```

-Выполняется операция по скруглению ребра, номер ребра 1, радиус задан 20 мм.

```
BRepFilletAPI_MakeFillet aFillet(box);
aFillet.Add(20, TopoDS::Edge(edge.FindKey(1)));
```

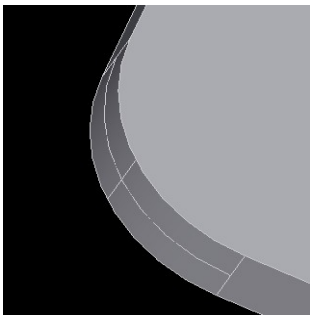

Рис. 6. Построение скругления в модели

Построение фаски (рис. 7):

-Создаётся массив и заполняется гранями модели:

```
TopTools_IndexedMapOfShape face; // массив граней
TopExp::MapShapes(aFillet , TopAbs_FACE, face);
```

-Выполняется операция по построению фаски, указывается номер ребра и номер грани, рядом стоящей, а также размер фаски. В случае несимметричной – надо определить два размера.

```
BRepFilletAPI_MakeChamfer aChamfer(aFillet);
//aChamfer.Add(15, 20, TopoDS::Edge(edge.FindKey(5)),
TopoDS::Face(face.FindKey(6)));
aChamfer.Add(30,TopoDS::Edge(edge.FindKey(5)),
TopoDS::Face(face.FindKey(6)));
```

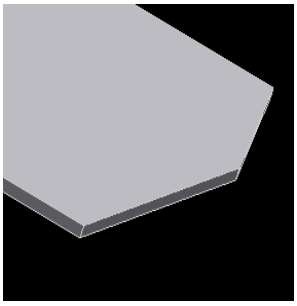

Рис. 7. Построение скругления в модели

Фрагмент программного кода по построению скруглений и фасок в твердотельной модели блока:

```
 TopoDS_Shape box = BRepPrimAPI_MakeBox(gp_Pnt(0, 0, 0), 100, 50,
10).Shape();
    TopTools_IndexedMapOfShape edge; // массив ребер
    TopExp::MapShapes(box, TopAbs_EDGE, edge);
    qDebug()<<"Edge's is:"<<edge.Size();
    BRepFilletAPI_MakeFillet aFillet(box);
    aFillet.Add(20, TopoDS::Edge(edge.FindKey(1)));
    TopTools_IndexedMapOfShape face; // массив граней
    TopExp::MapShapes(aFillet , TopAbs_FACE, face);
    qDebug()<<"Face's is:"<<face.Size();
    BRepFilletAPI_MakeChamfer aChamfer(aFillet);
    //aChamfer.Add(15, 20, TopoDS::Edge(edge.FindKey(5)),
   TopoDS::Face(face.FindKey(6)));
 aChamfer.Add(30,TopoDS::Edge(edge.FindKey(5)),
 TopoDS::Face(face.FindKey(6)));
    Handle (AIS_Shape) aAISShape = new AIS_Shape(aChamfer);
    aAISShape->SetMaterial(Graphic3d_NOM_CHROME);
    aAISShape->SetDisplayMode(AIS_Shaded);
    myOccView->getContext()->Display(aAISShape, true );
```

Результаты работы фрагмента программного кода показаны на рис. 8.

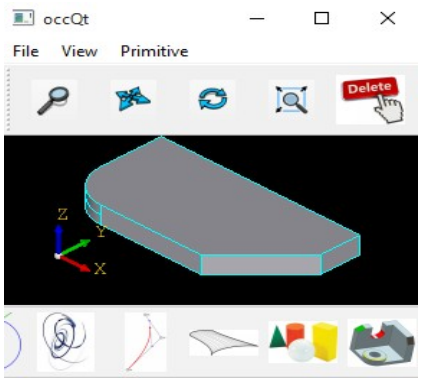

Рис. 8. Построение скруглений и фасок в твердотельной модели блока

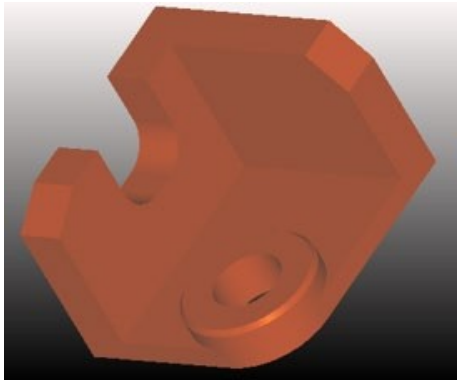

Рис. 9. Твердотельная модель уголка

Задания для самостоятельной работы:

1. Выполнить построение модели, представленной рис. 9, средствами Open Cascade. Размеры задать произвольным образом, однако требуется сохранить пропорции модели, как показано на изображении.

2. Добавить параметризацию в модель на ImGui. Необходимо изменить диаметр отверстия в заданном диапазоне от 10 до 25 мм.

ЛАБОРАТОРНАЯ РАБОТА № 4
ПОСТРОЕНИЕ МОДЕЛЕЙ
ПОСРЕДСТВОМ КИНЕМАТИЧЕСКИХ ОПЕРАЦИЙ

Цель работы: построение твердотельных моделей с использованием классов с реализацией кинематических операций.

Задачи и требования к выполнению:

1. Изучить классы кинематических операций.

2. Ознакомиться с примерами из документации на Open Cascade.

3. Собрать и подготовить к выполнению программное решение.

Теоретические сведения

Создаваемые геометрические тела в Open Cascade делятся на поверхности и твёрдые тела. Твердотельное моделирование представляет собой создание замкнутого геометрического объёма, описывающего геометрию детали. Ранее было сказано, что твёрдое тело может быть получено двумя основными способами: с помощью булевых операций и с помощью кинематических операций. Для создания исходного тела с помощью кинематических операций на основе Open Cascade можно использовать следующие команды:

Выдавливание. Тело выдавливания образуется путём перемещения плоского сечения в направлении, перпендикулярном его плоскости.

Вращение. С помощью операции вращения можно создать детали, представляющее собой тела вращения. Тела вращения образуется путем вращения плоского сечения вокруг оси.

В настройки проекта необходимо добавит статическую библиотеку TKPrim.lib, которая требуется для работы с такими операциями по моделированию тел, как построение моделей вращением и выдавливанием. Для

использования указанных операций требуется в проект включить два заголовочных файла, чтобы реализовать программное построение тел:

«BRepPrimAPI_MakePrism.hxx и BRepPrimAPI_MakeRevol.hxx

Построение тела операцией выдавливания (ход моделирования):

Производится построение замкнутого контура по набору точек:

gp_Pnt pnt1(x1, y1, z1);

……

gp_Pnt pntN(xn, yn, zn);

На основе точек создаются ребра, которые затем объединим в один профиль:

```
TopoDS_Edge aEdge1 = BRepBuilderAPI_MakeEdge(pnt1, pnt2);
TopoDS_Edge aEdgeN = BRepBuilderAPI_MakeEdge(pntN-1, pntN);
BRepBuilderAPI_MakeWire makeW;
makeW.Add(aEdge1);
makeW.Add(aEdgeN);
TopoDS_Wire Wc = makeW.Wire();
TopoDS_Face F = BRepBuilderAPI_MakeFace(Wc);
```

Задаётся направление выдавливания и расстояние, на которое необходимо выдавить профиль:

```
gp_Vec move(gp_Dir(0, 0, 1));
move *= 20;
```

Выполняется операция выдавливания:

```
TopoDS_Shape shape = BRepPrimAPI_MakePrism(F, move);
```

Отображается результат операции на экран (рис. 10):

```
Handle(AIS_Shape) anAISExtrude1 = new AIS_Shape(shape);
```

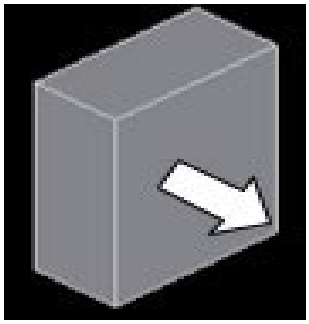

Рис. 10. Кинематическая операция выдавливания замкнутого контура

Построение тела операцией вращения:

Производится построение окружности с радиусом 10 мм. на плоскости XZ, центр окружности помещается в заданную точку (X, 0, 0). Построенную окружность определим как грань:

```
gp_Circ c1 = gp_Circ(gp_Ax2(gp_Pnt(X, 0, 0), gp_Dir(0.,1.,0.)), 10);
TopoDS_Edge Ec1 = BRepBuilderAPI_MakeEdge(c1);
TopoDS_Wire Wc1 = BRepBuilderAPI_MakeWire(Ec1);
TopoDS_Face F1 = BRepBuilderAPI_MakeFace(Wc1);
```

Определяется ось OZ, как ось вращения:

```
gp_Ax1 ax1(gp_Pnt(0, 0, 0), gp_Dir(0, 0, 1));
```

Выполняется операция по вращению замкнутого контура с углом поворота 270° с отображением результата на экране:

```
TopoDS_Shape shape = BRepPrimAPI_MakeRevol(F1, ax1, (270 * M_PI / 180));
Handle(AIS_Shape) aAISShape = new AIS_Shape(shape);
```

Результаты показаны на рисунке 11.

Рис. 11. Кинематическая операция вращения замкнутого контура

Листинг по построению блока выдавливанием:

```
#include <BRepPrimAPI_MakePrism.hxx>
void occQt::buildExtude()
{
   gp_Pnt pnt1(0, 0, 0);
   gp_Pnt pnt2(20, 0, 0);
   gp_Pnt pnt3(20, 40, 0);
   gp_Pnt pnt4(0, 40, 0);
   TopoDS_Edge aEdge1 = BRepBuilderAPI_MakeEdge(pnt1, pnt2);
   TopoDS_Edge aEdge2 = BRepBuilderAPI_MakeEdge(pnt2, pnt3);
   TopoDS_Edge aEdge3 = BRepBuilderAPI_MakeEdge(pnt3, pnt4);
   TopoDS_Edge aEdge4 = BRepBuilderAPI_MakeEdge(pnt4, pnt1);
   BRepBuilderAPI_MakeWire makeW;
```

```
    makeW.Add(aEdge1);
    makeW.Add(aEdge2);
    makeW.Add(aEdge3);
    makeW.Add(aEdge4);
    TopoDS_Wire Wc = makeW.Wire();

    TopoDS_Face F = BRepBuilderAPI_MakeFace(Wc);
    gp_Vec move(gp_Dir(0, 0, 1));
    move *= 50;
    TopoDS_Shape shape = BRepPrimAPI_MakePrism(F, move);
    Handle (AIS_Shape) aAISShape = new AIS_Shape(shape);
    aAISShape->SetMaterial(Graphic3d_NOM_CHROME);
    aAISShape->SetDisplayMode(AIS_Shaded);
    myOccView->getContext()->Display(aAISShape, true );
```

Результаты показаны на рис. 12.

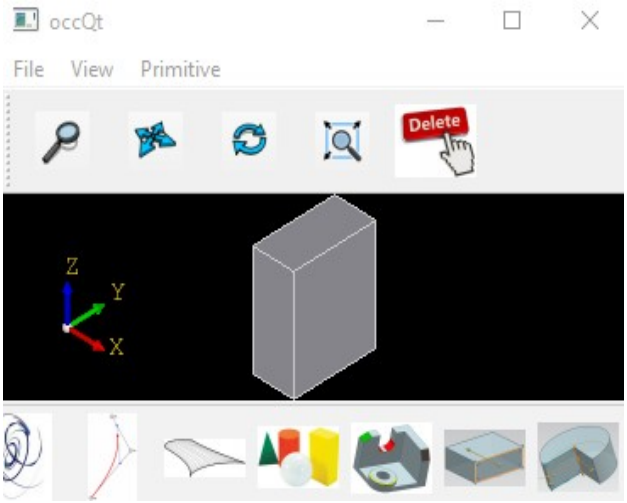

Рис. 12. Кинематическая операция построения модели выдавливанием

Листинг по построению фрагмента спирали вращением (рис. 13):

```
    #include <BRepPrimAPI_MakeRevol.hxx>
void occQt::buildRotate()
{
    gp_Circ c1 = gp_Circ(gp_Ax2(gp_Pnt(100, 0, 0),
gp_Dir(0.,1.,0.)), 10);
    TopoDS_Edge Ec1 = BRepBuilderAPI_MakeEdge(c1);
    TopoDS_Wire Wc1 = BRepBuilderAPI_MakeWire(Ec1);
    TopoDS_Face F1 = BRepBuilderAPI_MakeFace(Wc1);
    gp_Ax1 ax1(gp_Pnt(0, 0, 0), gp_Dir(0, 0, 1));
```

```
    TopoDS_Shape shape = BrepPrimAPI_MakeRevol
(F1, ax1, (270 * M_PI / 180));
    Handle (AIS_Shape) aAISShape = new AIS_Shape(shape);
    aAISShape->SetMaterial(Graphic3d_NOM_CHROME);
    aAISShape->SetDisplayMode(AIS_Shaded);
    myOccView->getContext()->Display(aAISShape, true );
```

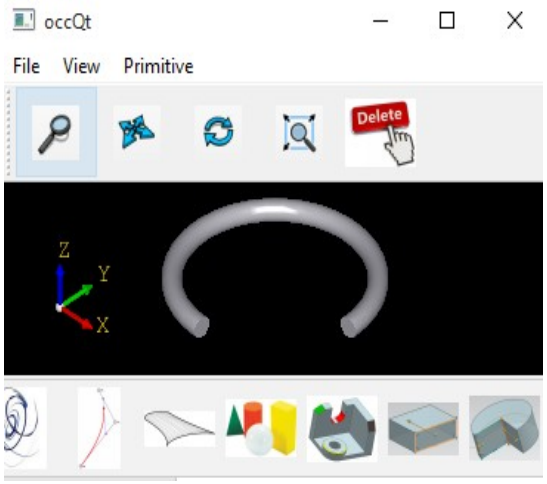

Рис. 13. Кинематическая операция построения модели вращением

Построение тела по траектории (ход моделирования)

Создаётся трасса и используется в качестве траектории:

```
   gp_Pnt pnt1(0, 0, 0);
    gp_Pnt pnt2(0, 0, 350);
    gp_Pnt pnt3(-20, 0, 370);
    gp_Pnt pnt4(-100, 0, 370);
    gp_Circ myCircle;
    myCircle.SetRadius(20);
    gp_Pnt circleCenter(-20.,0.,350);
    myCircle.SetAxis(gp_Ax1(circleCenter,gp_Dir(0,1,0)));
    Handle(Geom_TrimmedCurve) geometricArc = GC_MakeArcOfCircle(myCircle,
1.5*M_PI,2*M_PI,true);
    TopoDS_Edge aEdge1 = BRepBuilderAPI_MakeEdge(pnt1, pnt2);
    TopoDS_Edge aEdge2 = BRepBuilderAPI_MakeEdge(geometricArc);
    TopoDS_Edge aEdge3 = BRepBuilderAPI_MakeEdge(pnt3, pnt4);
    BRepBuilderAPI_MakeWire makeWire;
    makeWire.Add(aEdge1);
    makeWire.Add(aEdge2);
```

```
makeWire.Add(aEdge3);
```

Реализуется построение сечения, которое будет выдавлено по траектории:

```
gp_Circ c1b = gp_Circ(gp_Ax2(gp_Pnt(0,0,0), gp_Dir(0,0,1)), 50);
TopoDS_Edge E1b = BRepBuilderAPI_MakeEdge(c1b);
TopoDS_Wire Wb = BRepBuilderAPI_MakeWire(E1b);
TopoDS_Face F1 = BRepBuilderAPI_MakeFace(Wb);
```

Процедура получения топологического тела по траектории:

```
TopoDS_Shape S = BRepOffsetAPI_MakePipe(makeWire, F1);
```

Результаты построения показаны на рис. 14.

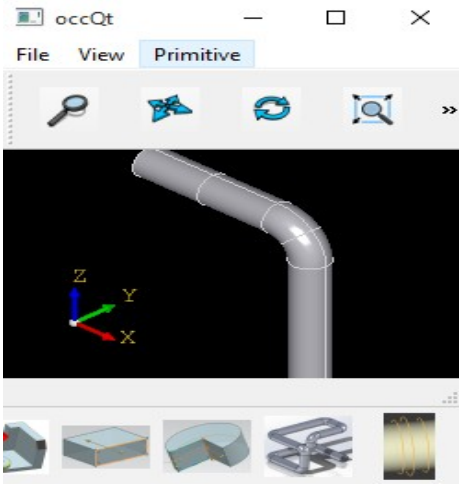

Рис. 14. Кинематическая операция построения модели по траектории

Реализация в OpenCascade (Tsections):

```
void occQt::buildTSection()
{
//1
gp_Circ c1b = gp_Circ(gp_Ax2(gp_Pnt(0,0,0), gp_Dir(0,0,1)), 50);
TopoDS_Edge E1b = BRepBuilderAPI_MakeEdge(c1b);
TopoDS_Wire W1b = BRepBuilderAPI_MakeWire(E1b);
//2
gp_Circ c2b = gp_Circ(gp_Ax2(gp_Pnt(0,0,150), gp_Dir(0,0,1)), 30);
TopoDS_Edge E2b = BRepBuilderAPI_MakeEdge(c2b);
```

```
TopoDS_Wire W2b = BRepBuilderAPI_MakeWire(E2b);
//3
gp_Circ c3b = gp_Circ(gp_Ax2(gp_Pnt(0,0,300), gp_Dir(0,0,1)), 75);
TopoDS_Edge E3b = BRepBuilderAPI_MakeEdge(c3b);
TopoDS_Wire W3b = BRepBuilderAPI_MakeWire(E3b);
BRepOffsetAPI_ThruSections generatorb(Standard_False,Standard_False);
generatorb.AddWire(W1b);
generatorb.AddWire(W2b);
generatorb.AddWire(W3b);
generatorb.Build();
TopoDS_Shape S2 = generatorb.Shape();
Handle(AIS_Shape) aAISShape = new AIS_Shape(S2);
aAISShape->SetMaterial(Graphic3d_NOM_CHROME);
aAISShape->SetDisplayMode(AIS_Shaded);
myOccView->getContext()->Display(aAISShape, true );
```

Результаты построения показаны на рис. 15.

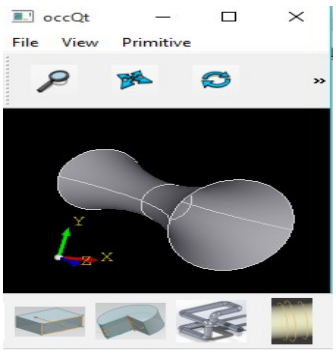

Рис. 15. Кинематическая операция построения модели по сечениям

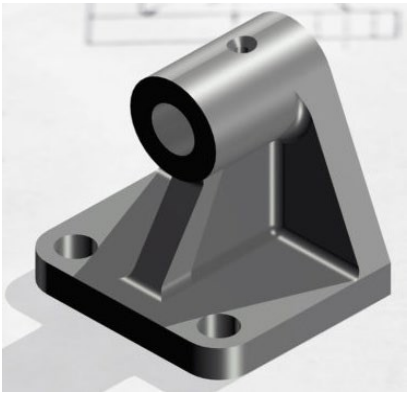

Рис. 16. Модель упора

Задания для самостоятельной работы:

1. Выполнить построение детали (рис. 16), используя приёмы моделирования предыдущих работ и операций по вращению с выдавливанием.

2. Выполнить параметризацию всех отверстий в созданной твердотельной модели на ImGui.

ЛАБОРАТОРНАЯ РАБОТА № 5
РАБОТА С ЭКСПОРТНЫМИ ФОРМАТАМИ ДАННЫХ

Цель работы: работа с импортированной (экспортируемой) геометрией объектов в Open Cascade.

Задачи и требования к выполнению:

1. Изучить приёмы работы с обменными форматами файлов в Open Cascade .

2. Ознакомиться с примерами из документации на Open Cascade.

3. Собрать и подготовить к выполнению программное решение.

Теоретические сведения

Open Cascade содержит большой набор трансляторов. Возможен импорт из наиболее распространённых нейтральных форматов, таких как STEP, IGES, Brep, STL (рис. 17).

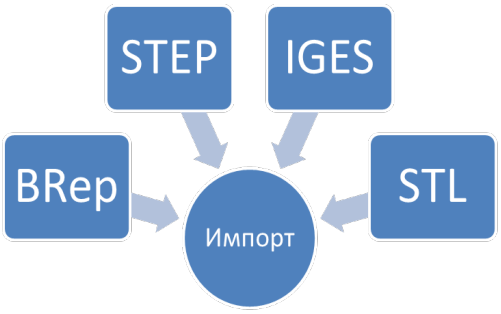

Рис. 17. Экспортные форматы Open Cascade

Далее будут рассмотрены методы Open Cascade, с помощью которых появится возможность просмотреть импортированные данные нейтральных форматов. Для начала необходимо настроить свойства проекта: добавить в дополнительные зависимости проекта 3 статические библиотеки Open Cascade:

TKIGES.lib, TKSTL.lib, TKSTEP.lib. Библиотека TKBRep.lib –требуется для открытия файлов Brep. Так же для импорта конкретного формата файла необходимо включить в состав проекта определённый заголовочный файл Open Cascade: STEPControl_Reader.hxx и STEPControl_Controller.hxx для файлов формата STEP, IGESControl_Reader.hxx и IGESControl_Controller.hxx для файлов IGES, StlAPI_Reader.hxx для файлов STL. Для импорта файлов Brep необходимо включить два заголовочных файла: BRepTools.hxx и Brep_Builder.hxx.

Импорт и экспорт в STL

Для импорта STL файлов также необходимо сначала создать транслятор, указать имя файла, затем отобразить результат на экран:

```
TopoDS_Shape shape;
StlAPI_Reader stlReader;
stlReader.Read(shape, importName);
myOccView->getContext()->Display(shape, true );
```

Экспорт в формат STL

```
myOccView->getContext()-> InitSelected();
StlAPI_Writer stlWriter;
stlWriter.Write(myOccView->getContext()->SelectedShape(),
exportName);
```

Импорт и экспорт в BREP

Импорт данных в формате Brep (открыть файлы) реализуется следующим функционалом:

```
BRep_Builder B;
TopoDS_Shape shape;
BRepTools::Read(shape, importName, B);
Handle(AIS_Shape) aAISShape = new AIS_Shape(shape);
```

Экспорт (запись) в формат Brep может быть реализована так:

```
myOccView->getContext()->InitSelected();
BRepTools::Write(myOccView->getContext()->
SelectedShape(), exportName);
```

Листинг проектного решения по использованию обменных форматов данных. Следует обратить внимание, что в примере используются компоненты интерфейса Qt, а в текущих условиях необходимо реализовать интерфейс с использованием ImGui.

```
#include <QFileDialog>
#include <StlAPI_Reader.hxx>
```

```
#include <StlAPI_Writer.hxx>
void occQt::openFiles()
{
    QString file = QFileDialog::getOpenFileName(this,
                                QString::fromUtf8("Открыть файл"),
                                QDir::currentPath(),
                                "STL format(*.stl)");
    if (!file.isEmpty()) OpenSTL(qPrintable(file));
}
Standard_Boolean occQt::OpenSTL(const Standard_CString&  aFileName)
{
  TopoDS_Shape shape;
  StlAPI_Reader stlReader;
  stlReader.Read(shape, aFileName);
  Handle(AIS_Shape) aAISShape = new AIS_Shape(shape);
  myOccView->getContext()->Display(aAISShape, true );
  return Standard_True;
}
void occQt::saveFiles()
{
    QString  file  =  QFileDialog::getSaveFileName(this,  tr("Сохранить
файл"),
                                QDir::currentPath(),
                                tr("STL format(*.stl)"));
     if (!file.isEmpty()) SaveSTL(qPrintable(file));
}
Standard_Boolean occQt::SaveSTL(const Standard_CString&  aFileName)
{
    AIS_ListOfInteractive myListObj;
    myOccView->getContext()->DisplayedObjects(myListObj);
    //qDebug()<<aListOfObjects.Size();
    Handle(AIS_Shape) S = Handle(AIS_Shape)::DownCast(myListObj.First());
    if (S.IsNull()){
     QMessageBox::critical(this,"Errors","Empty object, no save stl");
     return Standard_False;
    }
    TopoDS_Shape Sh=S->Shape();
    StlAPI_Writer stlWriter;
    stlWriter.Write(Sh, aFileName);
    return Standard_True;
}
```

На рис. 18 показана модель после ее открытия в приложении в формате STL.

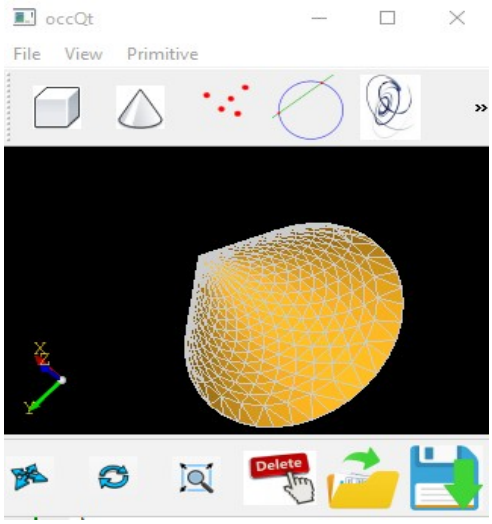

Рис. 18. Экспортные форматы Open Cascade

Задания для самостоятельной работы:

1. Подготовить запись и чтение данных в форматах (IGES, BREP и STEP) для любой из моделей, созданной ранее.

2. Разработать диалоговое окно по работе с файлами на ImGui.

ЛАБОРАТОРНАЯ РАБОТА № 6
ТОПОЛОГИЧЕСКИЕ ЭЛЕМЕНТЫ В OPEN CASCADE

Цель работы: работа с топологией в Open Cascade.

Задачи и требования к выполнению:

1. Изучить понятие топологии и что представляют собой топологические элементы в твердотельной модели.

2. Ознакомиться с примерами из документации на Open Cascade.

3. Собрать и подготовить к выполнению программное решение.

Теоретические сведения

Топология твердотельных моделей - область в компьютерной графике, которая изучает геометрическую структуру и связи между вершинами, рёбрами и гранями в трёхмерных моделях. Определяет, какие вершины соединены рёбрами, какие ребра образуют грани, и какие грани образуют объёмные объекты. Модель данных содержит топологические объекты, выделенные

синим цветом и геометрические объекты, выделенные красным цветом (рис. 19, 20 и 21).

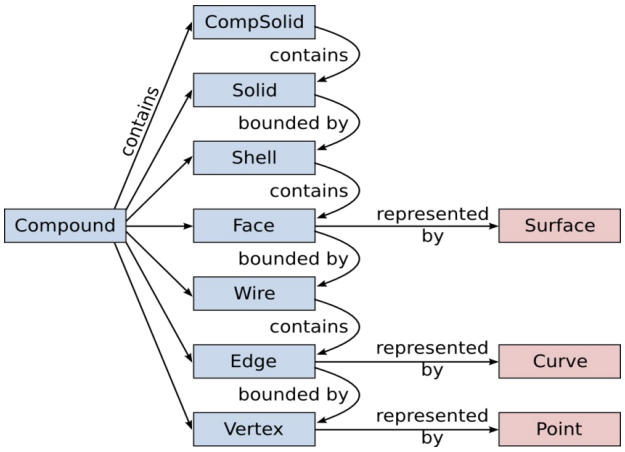

Рис. 19. Топологическая структура модели

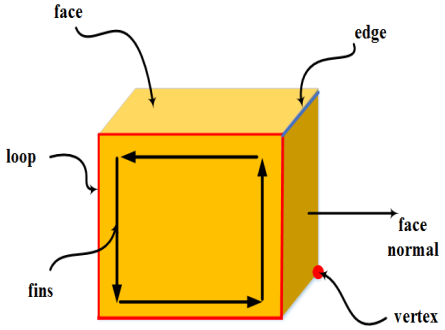

Рис. 20. Топологическое представление

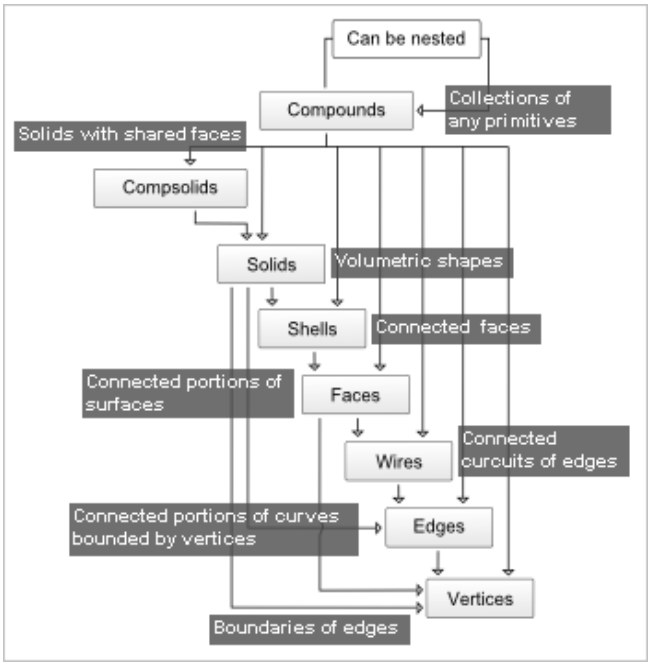

Рис. 21. Топологическое представление в Open Cascade

COMPOUND: группа любых фигур, которые описаны ниже:

COMPSOLID: набор твёрдых тел, соединённых их гранями. Расширяет функционал WIRE и SHELL до твёрдых тел.

SOLID: часть 3D-пространства, ограниченного оболочками.

SHELL: набор граней, соединённых некоторыми краями их границ контуром. Оболочка может быть открыта или закрыта.

FACE: часть плоскости (в 2D-геометрии) или поверхность (в 3D-геометрии), ограниченная замкнутым контуром. Его геометрия ограничена (обрезана) контурами.

WIRE: последовательность ребер, соединённых их вершинами. Она может быть открыта или закрыта в зависимости от того, связаны ли ребра или нет.

EDGE: одномерная форма, соответствующая кривой и связанная вершиной на каждом краю.

VERTEX: нульмерная форма, соответствующая точке в геометрии.

Листинг для анализа топологии блока в Open Cascade:

```
#include <QMessageBox>
void occQt::myTopology()
{
    TopoDS_Shape S = BRepPrimAPI_MakeBox(400.,250.,300.);
    Handle(AIS_Shape) aAISShape = new AIS_Shape(S);
    aAISShape->SetMaterial(Graphic3d_NOM_CHROME);
    aAISShape->SetDisplayMode(AIS_Shaded);
    myOccView->getContext()->Display(aAISShape, true );
    TopoDS_Shape myTopoDS;
    Handle(AIS_Shape) myShape;
    QString strInformation;
    strInformation.clear();
    int countEdge=0,countFace=0,countVertex=0;
    for (TopExp_Explorer ex(S, TopAbs_FACE); ex.More(); ex.Next())
    {
      if ((countFace==0) || (countFace==1)) {
          myTopoDS=ex.Current();
          myShape = new AIS_Shape(myTopoDS);
          myShape->SetColor(Quantity_NOC_YELLOW);
          myOccView->getContext()->Display(myShape, true );
      }
      countFace++;
    }
   TopTools_IndexedMapOfShape m;
   TopExp::MapShapes(S , TopAbs_EDGE, m);
   countEdge=m.Size();
 TopExp_Explorer exNew;
 exNew.Init(S,TopAbs_VERTEX);
 while(exNew.More())
 {
```

```
    myTopoDS=exNew.Current();
    myShape = new AIS_Shape(myTopoDS);
    myShape->SetColor(Quantity_NOC_RED);
    myOccView->getContext()->Display(myShape, true );
    countVertex++;
    exNew.Next();
  }
    strInformation+="Грани:"+QString::number(countFace)+"\n";
    strInformation+="Ребра:"+QString::number(countEdge)+"\n";
    strInformation+="Точки:"+QString::number(countVertex)+"\n";
    QmessageBox::information(nullptr,"Info",strInformation);
}
```

На рис. 22 показаны результаты работы фрагмента кода. Следует обратить внимание на то, что для блока представлена информация о 48 точках, на самом деле точек всего 12 в указанном теле. Объясняется это тем, что в процессе анализа могут использоваться грани и ребра с вершинами, которые входят в смежные грани и ребра данной модели. Избавиться от повторов можно, применив специальную функцию-компаратор, основанную например, множестве std::set.

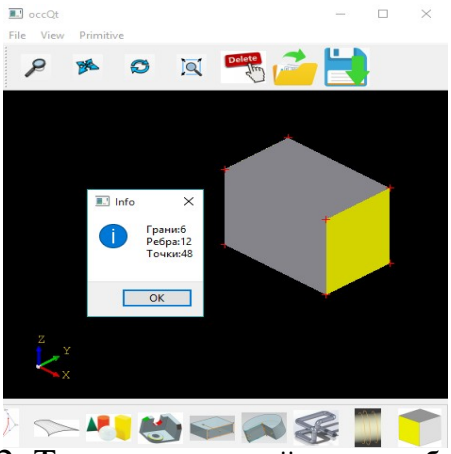

Рис. 22. Топологический анализ блока

Задания для самостоятельной работы:

1. Покрасить в заданный цвет плоские грани некоторой импортируемой геометрии (например, цилиндра).

2. Разработать диалоговое окно по отображению топологической информации в ImGui.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. ГОСТ 19.701–90 (ИСО 5807–85) «Единая система программной документации».

2. Пикус Ф.Г., Идиомы и паттерны проектирования в современном C++ / пер. с анг. А. А. Слинкина. - М.: ДМК Пресс, 2020. - 452 с.: ил.

3. Боресков А. В., Программирование компьютерной графики. Современный OpenGL. – М.:ДМК Пресс, 2019. – 372 с.: ил.

4. Голованов Н.Н. Геометрическое моделирование. Учебное пособие. - М.:КУРС:ИНФРА-М, 2016. - 400 с.

5. Steve Marschner, Fundamentals of Computer Graphics: International Student Edition 5th Edition, Published by A K Peters/CRC Press, 716 Pages 514 Color Illustrations, 2021.

ОГЛАВЛЕНИЕ

ГЕОМЕТРИЧЕСКИЕ ПОСТРОЕНИЯ В 3D

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ
по дисциплине «Геометрическое моделирование»
для студентов направления
09.03.01 «Информатика и вычислительная техника»
очной и заочной форм обучения

Составитель

Юров Алексей Николаевич

Издается в авторской редакции

Подписано к изданию 02.02.2024.

Уч.–изд. л. 1,4.

ФГБОУ ВО «Воронежский государственный технический университет»
394006 Воронеж, ул.20-летия Октября, 84