**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**   
**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ**

**ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**



Расчетно-пояснительная записка

К курсовому проекту по  
дисциплине: «Программирование в САПР»

Вариант № 4

Группа:   
221-325

Выполнил:   
Типтев В.К.

Подпись:

\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель:   
Джунковский А.В.

Дата выполнения:  
20.12.2023

г. Москва

**Оглавление**

[ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 2](#_Toc155044242)

[ПРАКТИЧЕСКАЯ ЧАСТЬ 9](#_Toc155044243)

[СПИСОК ЛИТЕРАТУРЫ: 33](#_Toc155044244)

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Сальниковое устройство, также известное как сальник, является одним из видов уплотнительных механизмов, используемых для герметизации подвижных компонентов различных установок и механизмов. Благодаря своей простой конструкции, это одно из наиболее распространенных и давно известных уплотнительных решений. Названия “сальниковая набивка”, “сальниковый узел” и т.д. сохранились с тех времен, когда в качестве уплотнителя использовалась пропитанная маслом пенька; сегодня же используются другие материалы, особенно фторопласты и асбестовое волокно.

Сальники особенно широко применяются в системах трубопровода, где они служат для уплотнения подвижных компонентов арматуры относительно внешней среды. В них используется уплотнительный элемент, создающий необходимое напряжение для обеспечения требуемой степени герметичности.

Кроме того, сальниковые механизмы, работающие по тому же принципу, широко используются в различных промышленных, корабельных и автомобильных механизмах. Сальники могут также применяться для герметизации неподвижного оборудования, такого как трубные и кабельные проходки.



Рисунок 1 — Виды сальников

Суть сальникового устройства в том, что на внешней стороне крышки или корпуса в том месте, где через них проходит шток или шпиндель, создаётся сальниковая камера (сальниковая коробка, сальница), в которую укладывается уплотнительный материал — сальниковая набивка. При помощи специальных устройств набивка поджимается вдоль оси шпинделя (штока), упираясь в стенки сальниковой камеры и уплотняя набивку. При сжатии набивки в ней создаются усилия, под действием которых она прижимается с одной стороны к стенке сальниковой камеры, а с другой — к цилиндрической поверхности шпинделя (штока). Таким образом создаётся герметичность и рабочая среда не проникает за пределы корпуса оборудования. В механизмах малых диаметров поджатие набивки производится накидной гайкой, больших — крышкой сальника (прижимной втулкой, грундбуксой) — при помощи откидных или анкерных болтов с гайками (обычно двух).

Сила трения, возникающая между сальниковой набивкой и штоком, препятствует последнему совершать необходимые перемещения, а при чрезмерных усилиях затяжки сальника делают их невозможными, поэтому для сальников имеют большое значение конструкторские и технологические решения, обеспечивающие их нормальную работу, среди них:

* материал набивки;
* размеры сальниковой камеры;
* конструкция деталей сальникового узла;
* материал штока (шпинделя), чистота обработки его поверхности и другие.

В некоторых случаях (среди арматуры, как правило, в регулирующих клапанах) для снижения трения применяются сальники со смазкой, которая подаётся извне через специальную маслёнку, в тяжелонагруженных механизмах применяется орошение штока водой, например в буровых насосах.

Современная сальниковая набивка представляет собой, как правило, шнур или кольца из асбеста с графитовой пропиткой. Также используются безасбестовые уплотнительные материалы из фторопласта или на основе графита. Вместо набивки иногда применяются манжеты из резины. Также применяются и многие другие материалы, что определяется конкретными условиями эксплуатации.

Сальники используются в различных промышленных и технических приложениях для предотвращения проникновения или выхода жидкостей, газов или пыли между двумя поверхностями, которые двигаются относительно друг друга. Они являются частью многих механических устройств и машин. Вот несколько областей, где сальники широко применяются:

1. Механизмы двигателей и трансмиссий автомобилей: Сальники используются в различных частях автомобиля, таких как коленчатые валы, распределительные валы, валы передач, чтобы предотвратить утечку масла и других жидкостей.
2. Промышленные насосы и компрессоры: В центробежных насосах, поршневых насосах и компрессорах сальники применяются для предотвращения утечек рабочих жидкостей.
3. Механизмы вращения и подшипники: Сальники могут использоваться в подшипниках и механизмах вращения для защиты от пыли и влаги.
4. Гидравлические и пневматические системы: В гидравлических и пневматических системах сальники предотвращают утечку рабочих жидкостей.
5. Промышленные насосы и компрессоры: Сальники применяются в широком спектре промышленных насосов и компрессоров для предотвращения утечек жидкостей или газов.
6. Электродвигатели: В некоторых электродвигателях используются сальники для предотвращения утечек смазочных материалов.
7. Пищевая и фармацевтическая промышленность: Сальники, соответствующие стандартам, применяются в оборудовании, используемом в пищевой и фармацевтической промышленности, чтобы обеспечивать герметичность и безопасность продукции.

Сальники могут различаться по своей конструкции и материалам в зависимости от конкретного применения.

Существует несколько различных типов сальников, и выбор конкретного типа зависит от условий эксплуатации, типа машины или оборудования, а также от того, какие жидкости или газы необходимо удерживать. Вот несколько основных типов сальников:

1. Одинарные сальники:

Это самый базовый тип сальников, состоящий из одного уплотнительного элемента, который предотвращает утечку между двумя поверхностями. Одинарные сальники просты в конструкции и применяются в различных машинах и механизмах.

1. Двойные сальники:

Двойные сальники включают два уплотнительных элемента, расположенных друг за другом. Между ними создается пространство, в которое может поступать дополнительное давление, что улучшает эффективность сальника. Этот тип часто используется в приложениях с высоким давлением.

1. Сальники с барьерной жидкостью:

Эти сальники включают систему барьерной жидкости, которая предотвращает проникновение вредных веществ в сальник и обеспечивает дополнительное уплотнение. Такие сальники применяются в условиях, где требуется особенно высокая степень герметичности.

1. Сальники с тросовой оплеткой:

Этот тип сальников обычно используется в поворотных механизмах, таких как валы. Он имеет тросовую оплетку вокруг уплотнительного элемента, обеспечивая дополнительную прочность и устойчивость к вращающимся движениям.

1. Сальники на основе материалов:

Существуют различные материалы, используемые для создания уплотнительных элементов сальников, такие как резина, полиуретан, фторопласт и другие полимеры. Выбор материала зависит от температурных условий, типа жидкости или газа, а также химической совместимости.

1. Пружинные сальники:

Эти сальники включают пружину, которая обеспечивает постоянное давление уплотнительного элемента к контактирующей поверхности. Применяются в местах, где нужно компенсировать износ или деформацию.

1. Сальники для высоких и низких температур:

Существуют специальные сальники, адаптированные для экстремальных температурных условий. Это могут быть сальники с теплоизоляцией или специальные материалы, устойчивые к низким или высоким температурам.

Выбор конкретного типа сальника зависит от конкретных требований приложения, и важно учитывать условия эксплуатации, химическую совместимость и другие факторы.

Сальник СКРО – это элемент, применяющийся для ввода кабельных изделий в электрооборудование и аппаратуру водонепроницаемого и герметичного исполнения. С одной стороны в сальник СКРО вводится кабель, а вторая сторона сальника развальцовывается с внутренней стороны электрооборудования.

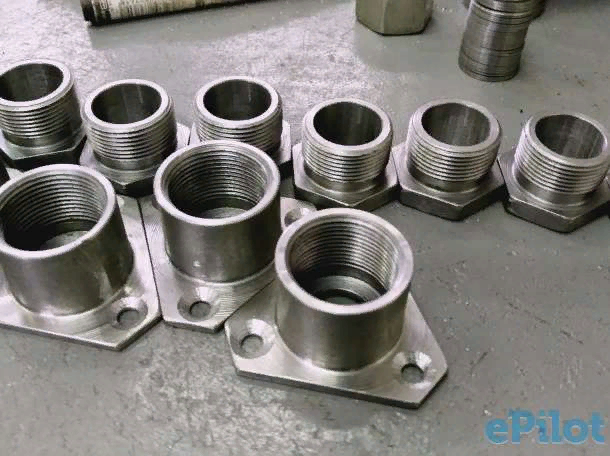


Рисунок 2 — Сальники кабельные по ГОСТ

Аббревиатура СКРО обозначает: С – сальник, К – кабельный, Р – развальцованный, О – односторонний. Возможные типоразмеры сальников СКРО-16, СКРО-27, СКРО-42, СКРО-60, СКРО-76, СКРО-90. При необходимости, можно изготовить нестандартный типоразмер сальника. Стандарт на сальники: ГОСТ 4860.1-83, ГОСТ 4860.2-83.

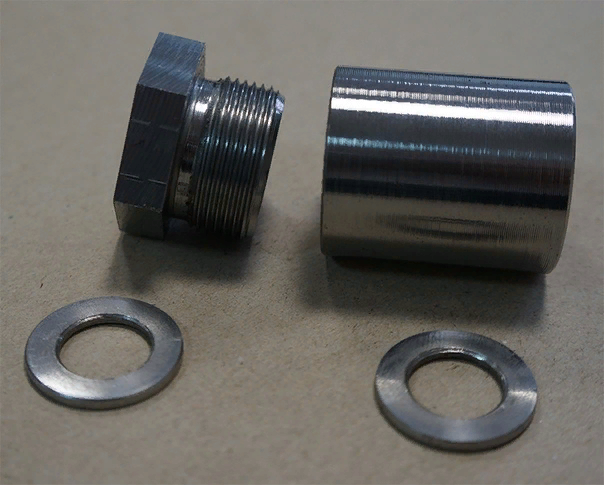


Рисунок 3 — Внутреннее устройство сальника

Односторонние сальники типа СКРО представляют собой разнообразную группу уплотнительных устройств. В зависимости от конкретных требований, условий эксплуатации и особенностей применения, существует несколько вариаций односторонних сальников СКРО, предназначенных для различных отраслей и типов технических устройств. Рассмотрим основные виды односторонних сальников типа СКРО:

1. Сальники для насосов. Эти сальники широко применяются в насосных системах для предотвращения утечек рабочих жидкостей. Они обеспечивают герметичное соединение между вращающимся валом насоса и его корпусом, что особенно критично в случае насосов, используемых в химической промышленности или в системах водоснабжения.

2. Сальники для компрессоров. В компрессорных установках односторонние сальники СКРО применяются для предотвращения утечек газов. Они обеспечивают герметичность внутренних камер компрессора, что важно для эффективной работы систем сжатия газов.

3. Сальники для турбин. В энергетической отрасли сальники данного типа используются в турбинах для обеспечения герметичности и предотвращения утечек рабочих сред. Они поддерживают эффективность турбинных установок и предотвращают потери энергии.

4. Сальники для вентиляторов. В системах вентиляции и кондиционирования воздуха односторонние сальники СКРО играют роль в обеспечении герметичности вращающихся вентиляторов. Это важно для предотвращения утечек воздуха и поддержания эффективной работы системы.

5. Сальники для автомобильных двигателей. В автомобильной промышленности эти сальники применяются для уплотнения вращающихся элементов внутри двигателя, предотвращая утечку масел и жидкостей.

6. Сальники для оборудования пищевой промышленности. В оборудовании, используемом в производстве пищевых продуктов, односторонние сальники СКРО обеспечивают герметичность вращающихся частей, предотвращая контаминацию продуктов.

7. Сальники для химической промышленности. В условиях, где существует воздействие агрессивных химических веществ, эти сальники применяются для предотвращения утечек и обеспечения безопасности процессов.

Разнообразие односторонних сальников СКРО позволяет выбирать оптимальный вариант в зависимости от конкретных требований каждого технического приложения. Это делает их востребованными и универсальными элементами в различных отраслях промышленности и техники.

API (Application Programming Interface) КОМПАС-3D представляет собой программный интерфейс, предназначенный для взаимодействия внешних приложений с программным обеспечением САПР (система автоматизированного проектирования) "КОМПАС-3D". Этот мощный инструмент позволяет разработчикам интегрировать и автоматизировать процессы проектирования, управления данными и взаимодействия с геометрическими моделями в программе "КОМПАС-3D".

API КОМПАС-3D позволяет создавать скрипты и приложения для автоматизации повторяющихся задач проектирования. Это может включать в себя создание деталей, сборок, генерацию отчетов и многое другое.

Разработчики могут использовать API для интеграции "КОМПАС-3D" с другими программами и системами, обеспечивая единый поток данных и управление проектами.

API позволяет создавать специализированные приложения, расширяя функциональность "КОМПАС-3D" в соответствии с уникальными потребностями пользователя.

В рамках моей работы я использовал API 5 КОМПАС-3D для реализации автоматизированных процессов проектирования и создания пользовательских приложений.

Применение API 5 существенно ускорило процессы проектирования, уменьшило вероятность ошибок и обеспечило более гибкий и эффективный подход к работе с программой "КОМПАС-3D". Это позволило мне сосредоточиться на более сложных и творческих аспектах моей работы, обеспечивая при этом высокую точность и качество проектирования.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

Цель работы: создание SDI приложения, позволяющее пользователю выбрать вариант размера интересующих его деталей, а затем самостоятельно строящее 3д модели выбранных деталей в программе КОМПАС.

План работы:

1. Создание интерфейса, соответствующего требованиям.
2. Написание логики чтения и переключения вариантов размера и исполнений, их взаимосвязи.
3. Написание кода построения деталей и сборки в зависимости от выбранных параметров.

**Создание интерфейса**

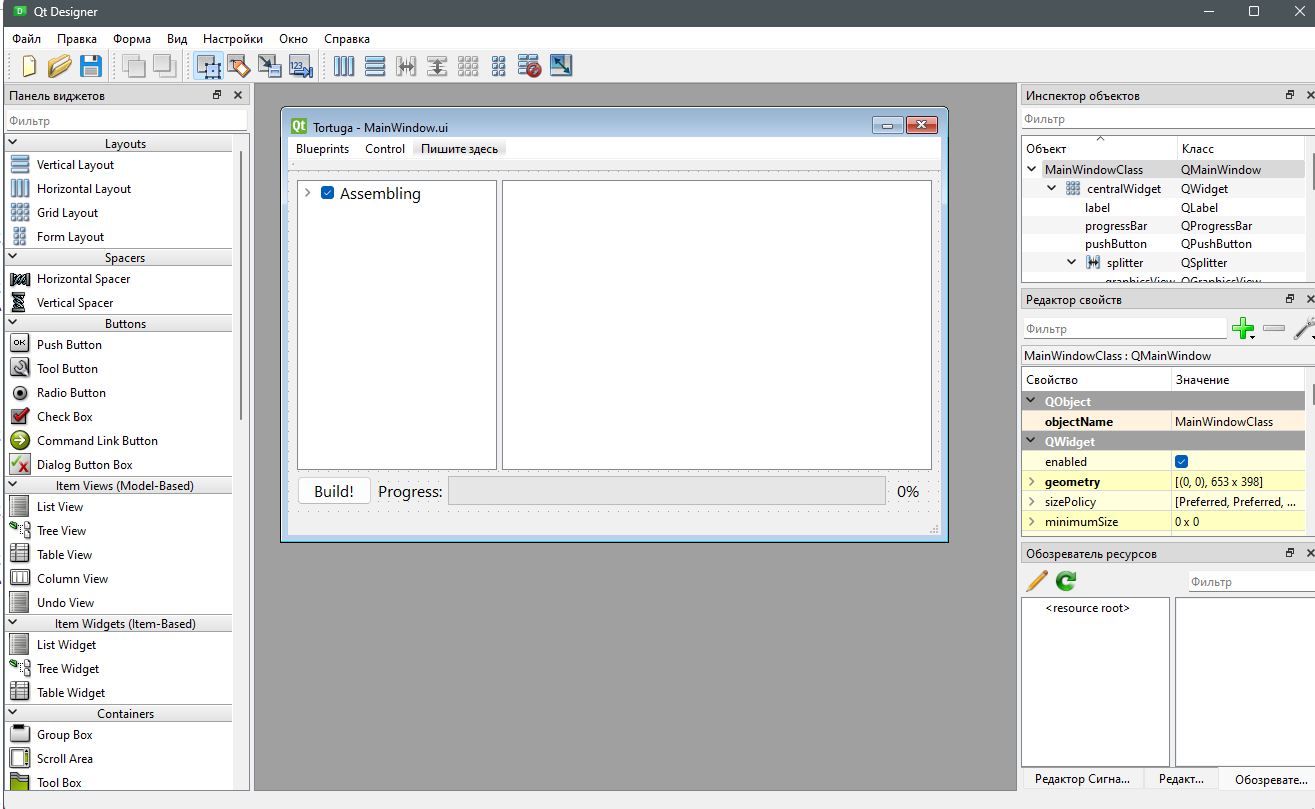
Было решено создавать интерфейс на базе фреймворка Qt. В редакторе создана основа окна программы, элементы расположены при помощи grid label, что позволило элементам адаптивно изменять размер под масштаб окна. Так же древовидный список элементов и поле для графического отображения чертежей были разделены сплиттером, так же для удобства масштабирования.

Рисунок 1 Создание интерфейса в редакторе

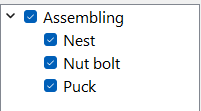
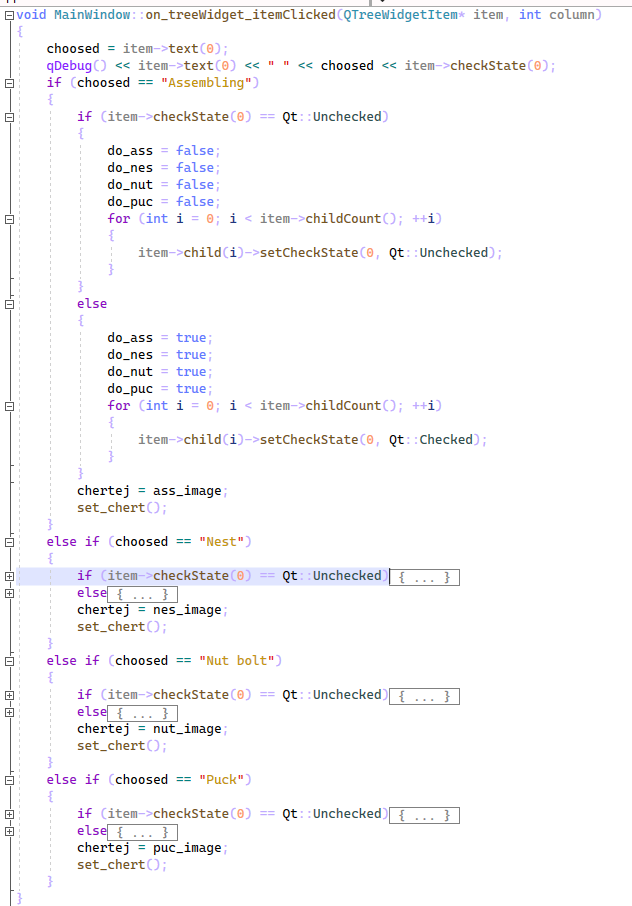
Каждый элемент древовидного списка обозначает свою часть модели – сборка, гнездо, гайка и шайба.

Рисунок 2 Дерево с выбором деталей

Галочками можно отмечать элементы, которые необходимо смоделировать (реализована зависимость постановки галочек – нельзя собрать сборку без всех деталей, но можно выбрать все детали, не строя сборку)



Одинарное нажатие на элемент списка отображает его чертёж в области просмотра – будь то чертёж детали или всей сборки.

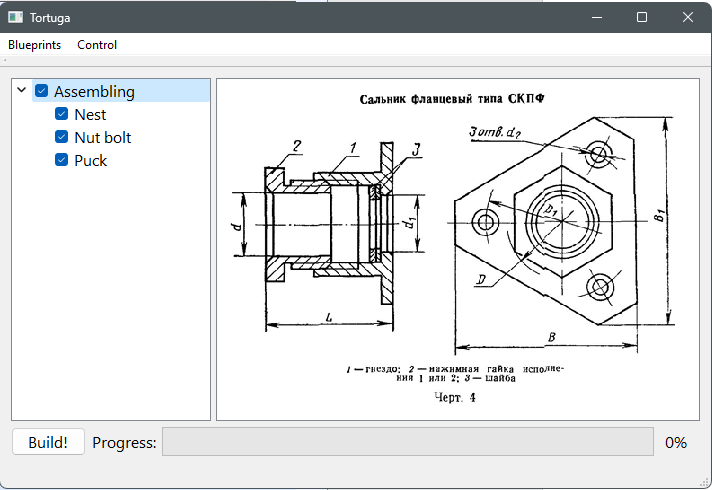


Рисунок 3 Интерфейс основного окна программы

**void** MainWindow::on\_treeWidget\_itemDoubleClicked(QTreeWidgetItem\* item, **int** column)

{

choosed = item->text(0);

qDebug() << item->text(0) << " " << choosed;

**if** (choosed == "Assembling")

{

chertej = ass\_image;

}

**else** **if** (choosed == "Nest")

{

chertej = nes\_image;

}

**else** **if** (choosed == "Nut bolt")

{

chertej = nut\_image;

}

**else** **if** (choosed == "Puck")

{

chertej = puc\_image;

}

set\_chert();

PermDialog dlg;

dlg.choose\_det(choosed);

dlg.show();

dlg.exec();

}

Листинг 1 Переключение между чертежами

Так же переключение между чертежами можно осуществлять при помощи верхнего меню Blueprints

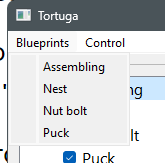


Рисунок 4 Верхнее меню для переключения между чертежами

Так же в верхнем меню есть пункт Reset, который позволяет сбросить все варианты размеров и исполнения к изначальным

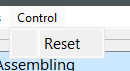
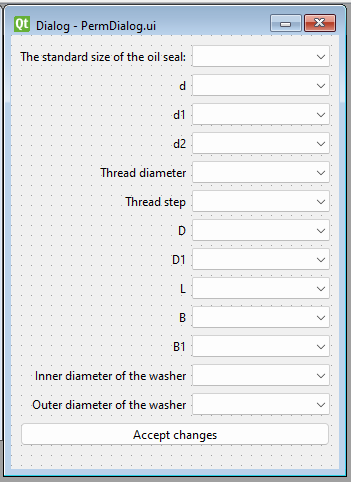


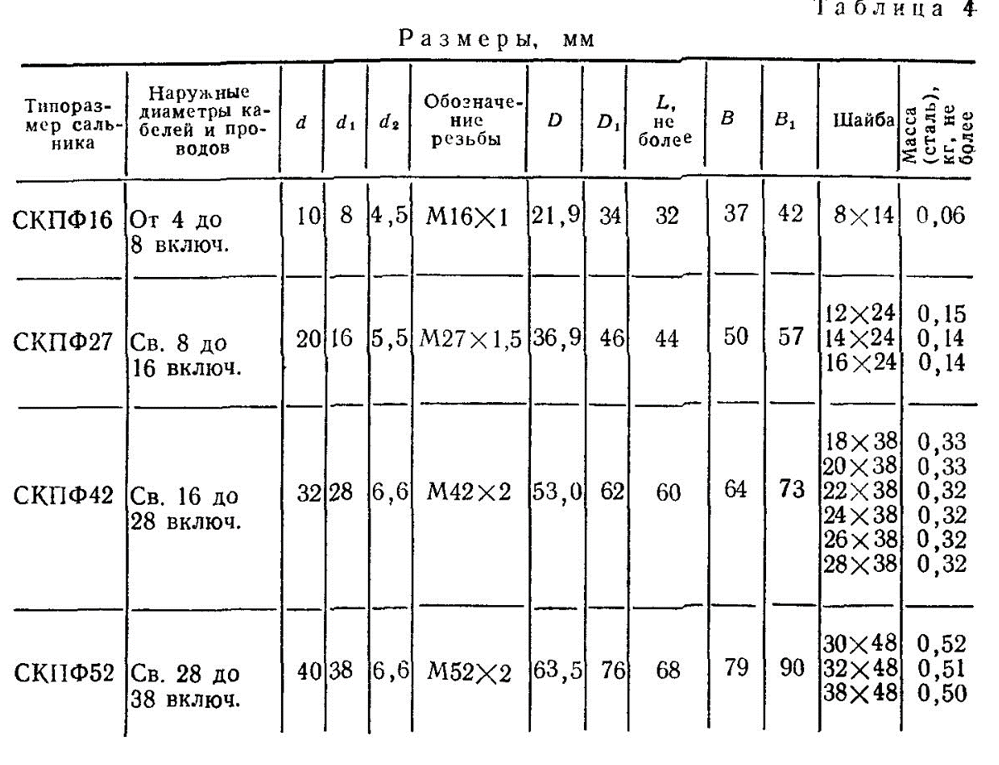
Рисунок 5 Элемент меню для сброса всех настроек

Сам диалог выбора размеров создан в отдельном .ui файле и содержит 13 лейблов и 13 комбобоксов, расположенных про помощи сетки, а так же кнопку подтверждения.

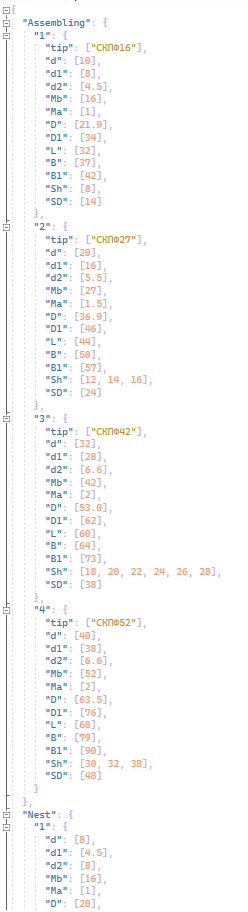
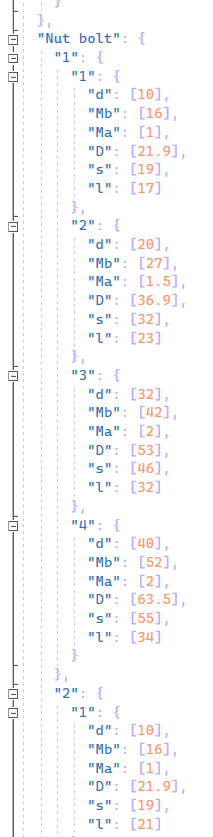


**Написание логики**

Перейдём к системе хранения вариантов размеров и исполнений. Вот в таком формате представлены варианты в моём варианте курсовой:

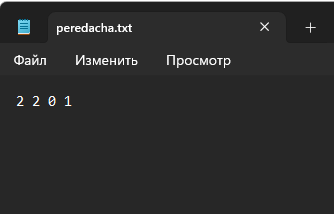


Что бы сделать их читаемыми для программы, я храню их в формате json:

В словаре по названию элемента хранится словарь с вариантами размеров (один вариант соответствует строке в таблице размеров детали/сборки). Гайка – исключение. В словаре Nut bolt хранится словарь с исполнениями (1 и 2), а уже в каждом из них словарь с вариантами размеров.

Так в таблице можно увидеть поля, в которых несколько значений. Они оформлены в json’е как элементы списка в одном из вариантов, выбранный индекс в этом списке и выбранный номер варианта хранятся в отдельном файле в папке с программой.



Так же среди деталей есть гайка, у которой есть два исполнения – под пластик и под металл. От выбора исполнения так же зависит список доступных вариантов размеров. Исполнение гайки так же сохраняется в файле.

Сам диалог, вызываемый двойным нажатием по нужной детали, универсален для всех четырёх элементов и меняет содержимое лэйблов и комбобоксов под переменные, которые есть у этого элемента, а ненужные элементы диалога скрывает. Все варианты переменных грузятся при помощи библиотеки nlohmann json, выбор в комбобоксе ставится в зависимости от сохранённого в фейле значения варианта, а при его изменении в одном из комбобоксов изменяет его у всех остальных. При подтверждении изменений вариант перезаписывается в файле на выбранный.

**void** PermDialog::choose\_det(QString det)

{

std::fstream myfile;

myfile.open("peredacha.txt", std::ios\_base::in);

myfile >> variant >> nut\_isp >> puckind >> combo\_puckind;

tmp\_var = variant;

myfile.close();

**if** (det == "Assembling")

{

varnames = { "tip", "d", "d1", "d2", "Mb", "Ma", "D", "D1", "L", "B", "Sd", "SD"};

labelnames = {

"The standard size of the oil seal:",

"d",

"d1",

"d2",

"Thread diameter",

"Thread step",

"D",

"D1",

"L",

"B",

"Inner diameter of the washer",

"Outer diameter of the washer"

};

}

**else** **if** (det == "Nest")

{

varnames = {"d", "d1", "d2", "Mb", "Ma", "D", "D1", "D2", "B1", "l" };

labelnames = {

"d",

"d1",

"d2",

"Thread diameter",

"Thread step",

"D",

"D1",

"D2",...

В зависимости от выбранной детали создаётся список имён для лэйблов, а также список с именами переменных, по которым нужно обращаться к json словарю.

После этого изменяются сами лэйблы и добавляются варианты в комбобоксы.

**this**->setWindowTitle(det);

std::vector <**double**> allvals = {};

std::vector <std::string> sallvals = {};

**this**->setFixedHeight(varnames.size() \* 35 + 40);

**for** (**int** i = 0; i < labels.size(); i++)

{

**if** (i < varnames.size())

{

labels[i]->setText(QString::fromStdString(labelnames[i]));

qDebug() << varnames[i];

**if** (!(/\*varnames[i] == "tip" || varnames[i] == "Sd" || \*/det == "Nut bolt"))

{

**if** (varnames[i] != "tip")

{

allvals = get\_all\_vals(det, QString::fromStdString(varnames[i]));

qDebug() << allvals.size();

**for** (**int** j = 0; j < allvals.size(); j++)

{

comboxes[i]->addItem(QString::number(allvals[j]));

//comboxes[i]->setItemText(i, QString::fromStdString(std::to\_string(allvals[i])));

}

**if** (labels[i]->text() == "Inner diameter of the washer")

{

comboxes[i]->setCurrentIndex(combo\_puckind);

}

**else**

{

comboxes[i]->setCurrentIndex(variant - 1);

}

}

**else**

{

sallvals = get\_all\_string\_vals(det, QString::fromStdString(varnames[i]));

**for** (**int** j = 0; j < sallvals.size(); j++)

{

qDebug() << j << " " << sallvals[j];

comboxes[i]->addItem(QString::fromStdString(sallvals[j]));

//comboxes[i]->setItemText(i, QString::fromStdString(std::to\_string(allvals[i])));

}

comboxes[i]->setCurrentIndex(variant - 1);

}

}

**else**

{

**if** (varnames[i] == "isp")

{

comboxes[i]->addItem(QString::fromStdString("1"));

comboxes[i]->addItem(QString::fromStdString("2"));

comboxes[i]->setCurrentIndex(nut\_isp - 1);

}

**else**

{

allvals = get\_all\_vals(det, QString::fromStdString(varnames[i]), nut\_isp);

qDebug() << allvals.size();

**for** (**int** j = 0; j < allvals.size(); j++)

{

comboxes[i]->addItem(QString::number(allvals[j]));

//comboxes[i]->setItemText(i, QString::fromStdString(std::to\_string(allvals[i])));

}

comboxes[i]->setCurrentIndex(variant - 1);

}

}

}

**else**

{

labels[i]->setVisible(**false**);

comboxes[i]->setVisible(**false**);

}

}

Варианты переменных загружаются при помощи универсальной функции, возвращающей вектор с элементами.

std::vector <**double**> get\_all\_vals(QString det, QString pname, **int** nut\_isp = 0)

{

std::vector <**double**> allvals = {};

**if** (pname != "tip")

{

**for** (**int** i = 1; i < 5; i++)

{

//vals = peremsDoc[det.toStdString()][std::to\_string(i)][pname.toStdString()].get<std::vector<double>>();

**if** (nut\_isp == 0)

{

vals = peremsDoc[det.toStdString()][std::to\_string(i)][pname.toStdString()].get<std::vector<**double**>>();

}

**else**

{

vals = peremsDoc[det.toStdString()][std::to\_string(nut\_isp)][std::to\_string(i)][pname.toStdString()].get<std::vector<**double**>>();

}

allvals.insert(allvals.end(), vals.begin(), vals.end());

}

}

**return** allvals;

}

При изменении варианта размера, шайба, у которой есть переменная внутреннего диаметра с несколькими вариациями на один вариант размера – принимает первую вариацию из выбранного варианта размера, а при изменении вариации диаметра изменяется вариант размера для всех остальных переменных, соответствующий данной вариации (написаны отдельные функции для поиска вариации по варианту и наоборот)

**int** get\_shaib\_var(QString val)

{

**for** (**int** i = 1; i < 5; i++)

{

//vals = peremsDoc[det.toStdString()][std::to\_string(i)][pname.toStdString()].get<std::vector<double>>();

vals = peremsDoc["Puck"][std::to\_string(i)]["d"].get<std::vector<**double**>>();

qDebug() << vals[0] << val.toDouble();

**for** (**int** j = 0; j < vals.size(); j++)

{

**if** (val.toDouble() == vals[j])

{

**return** i;

}

}

}

}

**int** get\_shaib\_index(**int** variant)

{

**int** ind = 0;

**for** (**int** i = 1; i < variant; i++)

{

vals = peremsDoc["Puck"][std::to\_string(i)]["d"].get<std::vector<**double**>>();

ind += vals.size();

}

**return** ind;

}

**int** get\_shaib\_personal\_index(**int** index)

{

**int** ind = 0;

**for** (**int** i = 1; i < 5; i++)

{

vals = peremsDoc["Puck"][std::to\_string(i)]["d"].get<std::vector<**double**>>();

ind += vals.size();

**if** (ind > index)

{

**return** vals.size() - (ind - index);

}

}

**return** ind;

}

**Написание кода построения деталей и сборки**

Построение деталей начинается при нажатии на кнопку “Build!” В главном окне. Сначала все переменные читаются из json’а в зависимости от выбранного варианта размера и исполнения:

std::fstream myfile;

myfile.open("peredacha.txt", std::ios\_base::in);

myfile >> variant >> nut\_isp >> puckind;

myfile.close();

json peremsDoc{ json::parse(read\_json\_main()) };

// Load for Assembling

get\_perem(&Asd, "Assembling", "d", peremsDoc);

get\_perem(&Asd1, "Assembling", "d1", peremsDoc);

get\_perem(&Asd2, "Assembling", "d2", peremsDoc);

get\_perem(&AsMb, "Assembling", "Mb", peremsDoc);

get\_perem(&AsMa, "Assembling", "Ma", peremsDoc);

get\_perem(&AsD, "Assembling", "D", peremsDoc);

get\_perem(&AsD1, "Assembling", "D1", peremsDoc);

get\_perem(&AsL, "Assembling", "L", peremsDoc);

get\_perem(&AsB, "Assembling", "B", peremsDoc);

get\_perem(&AsB1, "Assembling", "B1", peremsDoc);

get\_perem(&AsSh1, "Assembling", "Sd", peremsDoc);

get\_perem(&AsSh2, "Assembling", "SD", peremsDoc);

// Load for Nest

get\_perem(&Ned, "Nest", "d", peremsDoc);

get\_perem(&Ned1, "Nest", "d1", peremsDoc);

get\_perem(&Ned2, "Nest", "d2", peremsDoc);

get\_perem(&NeMb, "Nest", "Mb", peremsDoc);

get\_perem(&NeMa, "Nest", "Ma", peremsDoc);

get\_perem(&NeD, "Nest", "D", peremsDoc);

get\_perem(&NeD1, "Nest", "D1", peremsDoc);

get\_perem(&NeD2, "Nest", "D2", peremsDoc);

get\_perem(&NeB1, "Nest", "B1", peremsDoc);

get\_perem(&Nel, "Nest", "l", peremsDoc);

// Load foor Nut bolt

get\_perem(&Nud, "Nut bolt", "d", peremsDoc);

get\_perem(&NuMb, "Nut bolt", "Mb", peremsDoc);

get\_perem(&NuMa, "Nut bolt", "Ma", peremsDoc);

get\_perem(&NuD, "Nut bolt", "D", peremsDoc);

get\_perem(&Nus, "Nut bolt", "s", peremsDoc);

get\_perem(&Nul, "Nut bolt", "l", peremsDoc);

// Load for Puck

get\_perem(&Pud, "Puck", "d", peremsDoc);

get\_perem(&PuD, "Puck", "D", peremsDoc);

get\_perem(&Pus, "Puck", "s", peremsDoc);

void MainWindow::get\_perem(double\* perem, std::string det, std::string pname, json peremsDoc)

{

if (det == "Nut bolt")

{

\*perem = peremsDoc[det][std::to\_string(nut\_isp)][std::to\_string(variant)][pname][0].get<double>();

}

else if ((det == "Assembling" && pname == "Sd") || (det == "Puck" && pname == "d"))

{

\*perem = peremsDoc[det][std::to\_string(variant)][pname][puckind].get<double>();

}

else

{

\*perem = peremsDoc[det][std::to\_string(variant)][pname][0].get<double>();

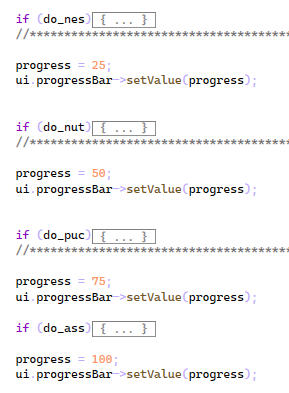
}

}

Далее задаём пути для сохранения будущих деталей и инициализируем компас:

|  |
| --- |
| std::string nest\_pth = fs::current\_path().string() + "\\Output detals\\Nest.m3d";  std::string nut\_pth = fs::current\_path().string() + "\\Output detals\\Nut bolt.m3d";  std::string puck\_pth = fs::current\_path().string() + "\\Output detals\\Puck.m3d";  std::string assembling\_pth = fs::current\_path().string() + "\\Output detals\\assembling.a3d";  CComPtr<IUnknown> pKompasAppUnk = nullptr;  if (!pKompasApp5)  {  // Получаем CLSID для Компас  CLSID InvAppClsid;  HRESULT hRes = CLSIDFromProgID(L"Kompas.Application.5", &InvAppClsid);  if (FAILED(hRes)) {  pKompasApp5 = nullptr;  return;  }  // Проверяем есть ли запущенный экземпляр Компас  //если есть получаем IUnknown  hRes = ::GetActiveObject(InvAppClsid, NULL, &pKompasAppUnk);  if (FAILED(hRes)) {  // Приходится запускать Компас самим так как работающего нет  // Также получаем IUnknown для только что запущенного приложения Компас  hRes = CoCreateInstance(InvAppClsid, NULL, CLSCTX\_LOCAL\_SERVER, \_\_uuidof(IUnknown), (void\*\*)&pKompasAppUnk);  if (FAILED(hRes)) {  pKompasApp5 = nullptr;  return;  }  }  // Получаем интерфейс приложения Компас  hRes = pKompasAppUnk->QueryInterface(\_\_uuidof(KompasObject), (void\*\*)&pKompasApp5);  if (FAILED(hRes)) {  return;  }  }  pKompasApp5->Visible = true; |

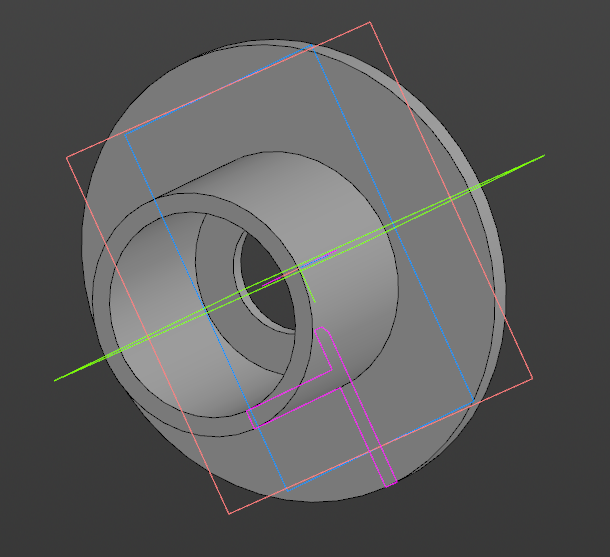
Далее происходит моделирование деталей. Перед началом создания каждого элемента программа проверяет, стоит ли галочка рядом с этим элементом в древовидном списке и только после делает. После окончания работы над деталью прогресс бар обновляется.



Разберём построение каждой из деталей и сборку поэтапно.

Начинаем с гнезда. Создаём эскиз по точкам и выдавливаем элемент вращения:

|  |
| --- |
| pDoc = pKompasApp5->Document3D();  pDoc->Create(false, true);  pPart = pDoc->GetPart(pTop\_Part);  pSketch = pPart->NewEntity(o3d\_sketch);  pSketchDef = pSketch->GetDefinition();  pSketchDef->SetPlane(pPart->GetDefaultEntity(o3d\_planeXOY));  pSketch->Create();  p2DDoc = pSketchDef->BeginEdit();  double point[9][2];  point[0][0] = Nel;  point[0][1] = NeMb / 2;  point[1][0] = Nel;  point[1][1] = NeD / 2;  point[2][0] = 3;  point[2][1] = NeD / 2;  point[3][0] = 3;  point[3][1] = NeD2 / 2;  point[4][0] = 0;  point[4][1] = NeD2 / 2;  point[5][0] = 0;  point[5][1] = Ned / 2 + 1;  point[6][0] = 1;  point[6][1] = Ned / 2;  point[7][0] = 3;  point[7][1] = Ned / 2;  point[8][0] = 3;  point[8][1] = NeMb / 2;  p2DDoc->ksLineSeg(point[0][0], point[0][1], point[1][0], point[1][1], 1);  p2DDoc->ksLineSeg(point[1][0], point[1][1], point[2][0], point[2][1], 1);  p2DDoc->ksLineSeg(point[2][0], point[2][1], point[3][0], point[3][1], 1);  p2DDoc->ksLineSeg(point[3][0], point[3][1], point[4][0], point[4][1], 1);  p2DDoc->ksLineSeg(point[4][0], point[4][1], point[5][0], point[5][1], 1);  p2DDoc->ksLineSeg(point[5][0], point[5][1], point[6][0], point[6][1], 1);  p2DDoc->ksLineSeg(point[6][0], point[6][1], point[7][0], point[7][1], 1);  p2DDoc->ksLineSeg(point[7][0], point[7][1], point[8][0], point[8][1], 1);  p2DDoc->ksLineSeg(point[8][0], point[8][1], point[0][0], point[0][1], 1);  p2DDoc->ksLineSeg(-10, 0, 10, 0, 3);  pSketchDef->EndEdit();  pRotate = pPart->NewEntity(o3d\_bossRotated);  pRotDef = pRotate->GetDefinition();  pRotDef->SetSketch(pSketch);  pRotDef->SetSideParam(TRUE, 360);  pRotate->Create(); |



Затем обрезаем основание до формы треугольника с обрезанными углами:

pSketch2 = pPart->NewEntity(o3d\_sketch);

pSketchDef = pSketch2->GetDefinition();

pSketchDef->SetPlane(pPart->GetDefaultEntity(o3d\_planeYOZ));

pSketch2->Create();

p2DDoc = pSketchDef->BeginEdit();

p2DDoc->ksCircle(0, 0, 100, 1);

double point1[6][2];

point1[0][0] = sin(asin(NeB1 / NeD2) - PI / 3) \* NeD2 / 2;

point1[0][1] = cos(asin(NeB1 / NeD2) - PI / 3) \* NeD2 / 2;

point1[1][0] = NeB1 / 2;

point1[1][1] = -cos(asin(NeB1 / NeD2)) \* NeD2 / 2;

point1[2][0] = sin((PI / 3 - (asin(NeB1 / NeD2) - PI / 3))) \* NeD2 / 2;

point1[2][1] = -cos((PI / 3 - (asin(NeB1 / NeD2) - PI / 3))) \* NeD2 / 2;

point1[3][0] = -sin((PI / 3 - (asin(NeB1 / NeD2) - PI / 3))) \* NeD2 / 2;

point1[3][1] = -cos((PI / 3 - (asin(NeB1 / NeD2) - PI / 3))) \* NeD2 / 2;

point1[4][0] = -NeB1 / 2;

point1[4][1] = -cos(asin(NeB1 / NeD2)) \* NeD2 / 2;

point1[5][0] = -sin(asin(NeB1 / NeD2) - PI / 3) \* NeD2 / 2;

point1[5][1] = cos(asin(NeB1 / NeD2) - PI / 3) \* NeD2 / 2;

p2DDoc->ksLineSeg(point1[0][0], point1[0][1], point1[1][0], point1[1][1], 1);

p2DDoc->ksLineSeg(point1[1][0], point1[1][1], point1[2][0], point1[2][1], 1);

p2DDoc->ksLineSeg(point1[2][0], point1[2][1], point1[3][0], point1[3][1], 1);

p2DDoc->ksLineSeg(point1[3][0], point1[3][1], point1[4][0], point1[4][1], 1);

p2DDoc->ksLineSeg(point1[4][0], point1[4][1], point1[5][0], point1[5][1], 1);

p2DDoc->ksLineSeg(point1[5][0], point1[5][1], point1[0][0], point1[0][1], 1);

pSketchDef->EndEdit();

pExtrude1 = pPart->NewEntity(o3d\_cutExtrusion);

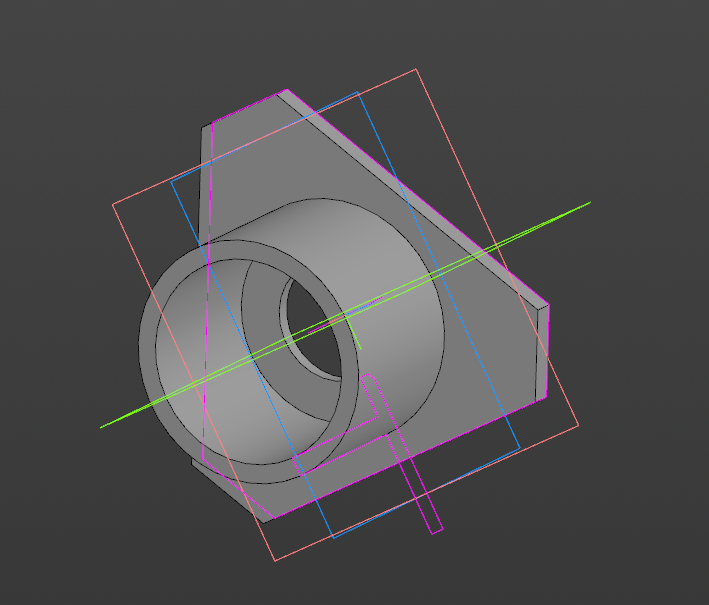
pExDef1 = pExtrude1->GetDefinition();

pExDef1->directionType = dtNormal;

pExDef1->SetSketch(pSketch2);

pExDef1->SetSideParam(true, etThroughAll, 0, 0, false);

pExtrude1->Create();



Далее вырезаем отверстия и размножаем их массивом:

pSketch1 = pPart->NewEntity(o3d\_sketch);

pSketchDef = pSketch1->GetDefinition();

pSketchDef->SetPlane(pPart->GetDefaultEntity(o3d\_planeXOY));

pSketch1->Create();

p2DDoc = pSketchDef->BeginEdit();

double point2[9][2];

point2[0][0] = 0;

point2[0][1] = -NeD1 / 2;

point2[1][0] = 0;

point2[1][1] = -NeD1 / 2 - Ned1 / 2;

point2[2][0] = 3 - Ned2 / 2 + Ned1 / 2;

point2[2][1] = -NeD1 / 2 - Ned1 / 2;

point2[3][0] = 3;

point2[3][1] = -NeD1 / 2 - Ned2 / 2;

point2[4][0] = 3;

point2[4][1] = -NeD1 / 2;

p2DDoc->ksLineSeg(point2[0][0], point2[0][1], point2[1][0], point2[1][1], 1);

p2DDoc->ksLineSeg(point2[1][0], point2[1][1], point2[2][0], point2[2][1], 1);

p2DDoc->ksLineSeg(point2[2][0], point2[2][1], point2[3][0], point2[3][1], 1);

p2DDoc->ksLineSeg(point2[3][0], point2[3][1], point2[4][0], point2[4][1], 1);

p2DDoc->ksLineSeg(point2[4][0], point2[4][1], point2[0][0], point2[0][1], 1);

p2DDoc->ksLineSeg(-10, -NeD1 / 2, 10, -NeD1 / 2, 3);

pSketchDef->EndEdit();

pExtrude = pPart->NewEntity(o3d\_cutRotated);

pRotDef1 = pExtrude->GetDefinition();

pRotDef1->SetSketch(pSketch1);

pRotDef1->SetSideParam(TRUE, 360);

pExtrude->Create();

pCircCopy = pPart->NewEntity(o3d\_circularCopy);

pCircDef = pCircCopy->GetDefinition();

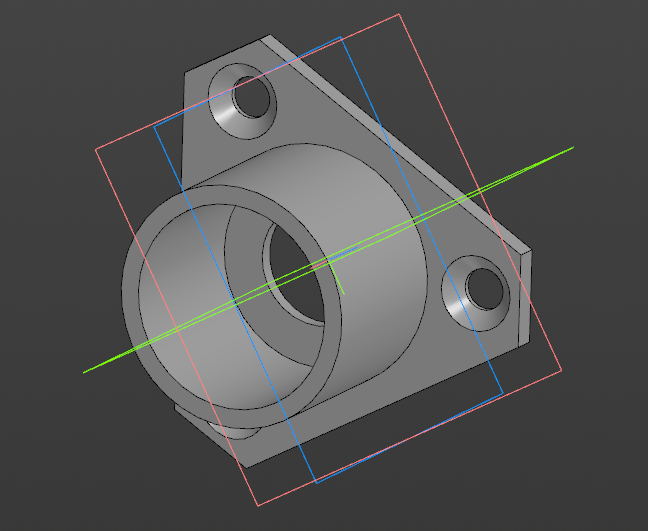
pCircDef->Putcount2(3);

pCircDef->SetAxis(pPart->GetDefaultEntity(o3d\_axisOX));

fl = pCircDef->GetOperationArray();

fl->Clear();

fl->Add(pExtrude);

pCircCopy->Create();

После идёт поиск нужного нам рёбра перебором с условиями. Оно нам понадобится для создания резьбы на детали и для сопряжения деталей в сборке.

flFaces = pPart->EntityCollection(o3d\_face);

for (int i = 0; i < flFaces->GetCount(); i++)

{

face = flFaces->GetByIndex(i);

def = face->GetDefinition();

if (def->GetOwnerEntity() == pRotate)

{

if (def->IsCylinder())

{

def->GetCylinderParam(&h, &r);

if (r == NeMb / 2.f)

{

col = def->EdgeCollection();

thredge = col->GetByIndex(0);

face->Putname("NestCilind");

qDebug() << "NestCilind";

face->Update();

}

}

else if (def->IsPlanar())

{

col = def->EdgeCollection();

for (int k = 0; k < col->GetCount(); k++)

{

ksEdgeDefinitionPtr d = col->GetByIndex(k);

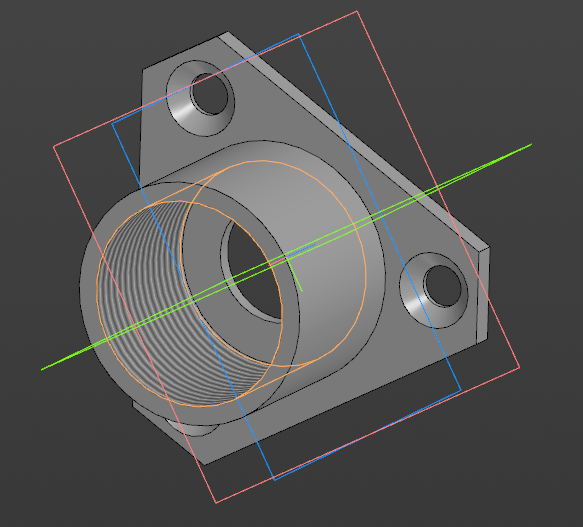
if (d->IsPeriodic())

{

p1 = d->GetVertex(true);

p2 = d->GetVertex(false);

double x1, y1, z1;

 p1->GetPoint(&x1, &y1, &z1);

if (x1 == 0)

{

face->Putname("NestConnect");

qDebug() << "NestConnect";

face->Update();

break;

}

}

}

}

}

}

thrPtr = pPart->NewEntity(o3d\_thread);

thrDef = thrPtr->GetDefinition();

thrDef->SetBaseObject(thredge);

thrDef->PutautoDefinDr(false);

thrDef->dr = NeMb;

thrDef->PutfaceValue(true);

thrDef->Putp(NeMa);

thrDef->length = Nul - Nul / 5;

qDebug() << thrPtr->Create();

pDoc->SaveAs(nest\_pth.c\_str());

Второй создаётся гайка. В зависимости от исполнения, процесс моделирования меняется. Процесс моделирования состоит из двух этапов для обоих исполнений. На первом создается выдавливание вращением:

pSketch = pPart->NewEntity(o3d\_sketch);

pSketchDef = pSketch->GetDefinition();

pSketchDef->SetPlane(pPart->GetDefaultEntity(o3d\_planeXOY));

pSketch->Create();

p2DDoc = pSketchDef->BeginEdit();

double nutpointrot[10][2];

nutpointrot[0][1] = NuD / 2;

nutpointrot[0][0] = 0;

nutpointrot[1][1] = NuD / 2;

nutpointrot[1][0] = -Nul / 5;

nutpointrot[2][1] = (Nud + (NuMb - Nud) / 2) / 2;

nutpointrot[2][0] = -Nul / 5;

nutpointrot[3][1] = (Nud + (NuMb - Nud) / 2) / 2;

nutpointrot[3][0] = -Nul / 5 - Nul / 9;

nutpointrot[4][1] = NuMb / 2;

nutpointrot[4][0] = -Nul / 5 - Nul / 9;

nutpointrot[5][1] = NuMb / 2;

nutpointrot[5][0] = -Nul + (NuMb / 2 - nutpointrot[2][1]);

nutpointrot[6][1] = nutpointrot[2][1];

nutpointrot[6][0] = -Nul;

nutpointrot[7][1] = Nud / 2;

nutpointrot[7][0] = -Nul;

nutpointrot[8][1] = Nud / 2;

nutpointrot[8][0] = -Nul / 7.5;

nutpointrot[9][1] = Nud / 2 + Nul / 7.5;

nutpointrot[9][0] = 0;

for (int i = 0; i < 9; i++)

{

p2DDoc->ksLineSeg(nutpointrot[i][0], nutpointrot[i][1], nutpointrot[i + 1][0], nutpointrot[i + 1][1], 1);

}

p2DDoc->ksLineSeg(nutpointrot[9][0], nutpointrot[9][1], nutpointrot[0][0], nutpointrot[0][1], 1);

p2DDoc->ksLineSeg(-10, 0, 10, 0, 3);

pSketchDef->EndEdit();

pRotate = pPart->NewEntity(o3d\_bossRotated);

pRotDef = pRotate->GetDefinition();

pRotDef->SetSketch(pSketch);

pRotDef->SetSideParam(TRUE, 360);

pRotate->Create();

Или

pSketch = pPart->NewEntity(o3d\_sketch);

pSketchDef = pSketch->GetDefinition();

pSketchDef->SetPlane(pPart->GetDefaultEntity(o3d\_planeXOY));

pSketch->Create();

p2DDoc = pSketchDef->BeginEdit();

double nutpointrot[8][2];

nutpointrot[0][1] = NuD / 2;

nutpointrot[0][0] = 0;

nutpointrot[1][1] = NuD / 2;

nutpointrot[1][0] = -Nul / 2.5;

nutpointrot[2][1] = NuMb / 2;

nutpointrot[2][0] = -Nul / 2.5;

nutpointrot[3][1] = NuMb / 2;

nutpointrot[3][0] = -Nul + (NuMb / 2 - (Nud + (NuMb - Nud) / 2) / 2);

nutpointrot[4][1] = (Nud + (NuMb - Nud) / 2) / 2;

nutpointrot[4][0] = -Nul;

nutpointrot[5][1] = Nud / 2;

nutpointrot[5][0] = -Nul;

nutpointrot[6][1] = Nud / 2;

nutpointrot[6][0] = -Nul / 7.5;

nutpointrot[7][1] = Nud / 2 + Nul / 7.5;

nutpointrot[7][0] = 0;

for (int i = 0; i < 7; i++)

{

p2DDoc->ksLineSeg(nutpointrot[i][0], nutpointrot[i][1], nutpointrot[i + 1][0], nutpointrot[i + 1][1], 1);

}

p2DDoc->ksLineSeg(nutpointrot[7][0], nutpointrot[7][1], nutpointrot[0][0], nutpointrot[0][1], 1);

p2DDoc->ksLineSeg(-10, 0, 10, 0, 3);

pSketchDef->EndEdit();

pRotate = pPart->NewEntity(o3d\_bossRotated);

pRotDef = pRotate->GetDefinition();

pRotDef->SetSketch(pSketch);

pRotDef->SetSideParam(TRUE, 360);

pRotate->Create();

Затем обрезка шестигранника:

pSketch = pPart->NewEntity(o3d\_sketch);

pSketchDef = pSketch->GetDefinition();

pSketchDef->SetPlane(pPart->GetDefaultEntity(o3d\_planeYOZ));

pSketch->Create();

p2DDoc = pSketchDef->BeginEdit();

p2DDoc->ksCircle(0, 0, 100, 1);

double nutpointshest[6][2];

nutpointshest[0][0] = 0;

nutpointshest[0][1] = NuD / 2;

nutpointshest[1][0] = cos(PI / 6) \* NuD / 2;

nutpointshest[1][1] = sin(PI / 6) \* NuD / 2;

nutpointshest[2][0] = cos(PI / 6) \* NuD / 2;

nutpointshest[2][1] = -sin(PI / 6) \* NuD / 2;

nutpointshest[3][0] = 0;

nutpointshest[3][1] = -NuD / 2;

nutpointshest[4][0] = -cos(PI / 6) \* NuD / 2;

nutpointshest[4][1] = -sin(PI / 6) \* NuD / 2;

nutpointshest[5][0] = -cos(PI / 6) \* NuD / 2;

nutpointshest[5][1] = sin(PI / 6) \* NuD / 2;

for (int i = 0; i < 5; i++)

{

p2DDoc->ksLineSeg(nutpointshest[i][0], nutpointshest[i][1], nutpointshest[i + 1][0], nutpointshest[i + 1][1], 1);

}

p2DDoc->ksLineSeg(nutpointshest[5][0], nutpointshest[5][1], nutpointshest[0][0], nutpointshest[0][1], 1);

pSketchDef->EndEdit();

pExtrude = pPart->NewEntity(o3d\_cutExtrusion);

pExDef1 = pExtrude->GetDefinition();

pExDef1->directionType = dtReverse;

pExDef1->SetSketch(pSketch);

pExDef1->SetSideParam(true, etThroughAll, 100, 0, false);

pExtrude->Create();

Или

pSketch = pPart->NewEntity(o3d\_sketch);

pSketchDef = pSketch->GetDefinition();

pSketchDef->SetPlane(pPart->GetDefaultEntity(o3d\_planeYOZ));

pSketch->Create();

p2DDoc = pSketchDef->BeginEdit();

p2DDoc->ksCircle(0, 0, 100, 1);

double nutpointshest[6][2];

nutpointshest[0][0] = 0;

nutpointshest[0][1] = NuD / 2;

nutpointshest[1][0] = cos(PI / 6) \* NuD / 2;

nutpointshest[1][1] = sin(PI / 6) \* NuD / 2;

nutpointshest[2][0] = cos(PI / 6) \* NuD / 2;

nutpointshest[2][1] = -sin(PI / 6) \* NuD / 2;

nutpointshest[3][0] = 0;

nutpointshest[3][1] = -NuD / 2;

nutpointshest[4][0] = -cos(PI / 6) \* NuD / 2;

nutpointshest[4][1] = -sin(PI / 6) \* NuD / 2;

nutpointshest[5][0] = -cos(PI / 6) \* NuD / 2;

nutpointshest[5][1] = sin(PI / 6) \* NuD / 2;

for (int i = 0; i < 5; i++)

{

p2DDoc->ksLineSeg(nutpointshest[i][0], nutpointshest[i][1], nutpointshest[i + 1][0], nutpointshest[i + 1][1], 1);

}

p2DDoc->ksLineSeg(nutpointshest[5][0], nutpointshest[5][1], nutpointshest[0][0], nutpointshest[0][1], 1);

pSketchDef->EndEdit();

pExtrude1 = pPart->NewEntity(o3d\_cutExtrusion);

pExDef2 = pExtrude1->GetDefinition();

pExDef2->directionType = dtBoth;

pExDef2->SetSketch(pSketch);

qDebug() << pExDef2->SetSideParam(false, etBlind, Nul / 2.5 - 2, 0, false);

pExtrude1->Create();

Затем, по аналогии с гнездом, идёт поиск ребра для задания резьбы, а так же для сопряжения деталей в сборке.

flFaces = pPart->EntityCollection(o3d\_face);

for (int i = 0; i < flFaces->GetCount(); i++)

{

face = flFaces->GetByIndex(i);

def = face->GetDefinition();

if (def->GetOwnerEntity() == pRotate)

{

if (def->IsCylinder())

{

def->GetCylinderParam(&h, &r);

if (r == NuMb / 2.f)

{

col = def->EdgeCollection();

thredge = col->GetByIndex(1);

face->Putname("NutCilind");

qDebug() << "NutCilind";

face->Update();

}

}

else if (def->IsPlanar())

{

col = def->EdgeCollection();

for (int k = 0; k < col->GetCount(); k++)

{

ksEdgeDefinitionPtr d = col->GetByIndex(k);

if (d->IsPeriodic())

{

p1 = d->GetVertex(true);

p2 = d->GetVertex(false);

double x1, y1, z1;

p1->GetPoint(&x1, &y1, &z1);

if (x1 == 0)

{

face->Putname("NutConnect");

qDebug() << "NutConnect";

face->Update();

break;

}

}

}

}

}

}

thrPtr = pPart->NewEntity(o3d\_thread);

thrDef = thrPtr->GetDefinition();

thrDef->SetBaseObject(thredge);

thrDef->PutautoDefinDr(true);

thrDef->PutfaceValue(true);

thrDef->Putp(NuMa);

thrDef->length = Nul - Nul / 5 - Nul / 9 - (NuMb / 2 - (Nud + (NuMb - Nud) / 2) / 2);

qDebug() << thrPtr->Create();

pDoc->SaveAs(nut\_pth.c\_str());

Создание шайбы, что идёт третьим пунктом, происходит в одно действие – круговое выдавливание:

pDoc = pKompasApp5->Document3D();

pDoc->Create(false, true);

pPart = pDoc->GetPart(pTop\_Part);

pSketch = pPart->NewEntity(o3d\_sketch);

pSketchDef = pSketch->GetDefinition();

pSketchDef->SetPlane(pPart->GetDefaultEntity(o3d\_planeXOY));

pSketch->Create();

p2DDoc = pSketchDef->BeginEdit();

double puckpoint[4][2];

puckpoint[0][0] = 0;

puckpoint[0][1] = PuD / 2;

puckpoint[1][0] = -Pus;

puckpoint[1][1] = PuD / 2;

puckpoint[2][0] = -Pus;

puckpoint[2][1] = Pud / 2;

puckpoint[3][0] = 0;

puckpoint[3][1] = Pud / 2;

for (int i = 0; i < 3; i++)

{

p2DDoc->ksLineSeg(puckpoint[i][0], puckpoint[i][1], puckpoint[i + 1][0], puckpoint[i + 1][1], 1);

}

p2DDoc->ksLineSeg(puckpoint[3][0], puckpoint[3][1], puckpoint[0][0], puckpoint[0][1], 1);

p2DDoc->ksLineSeg(-10, 0, 10, 0, 3);

pSketchDef->EndEdit();

pRotate = pPart->NewEntity(o3d\_bossRotated);

pRotDef = pRotate->GetDefinition();

pRotDef->SetSketch(pSketch);

pRotDef->SetSideParam(TRUE, 360);

pRotate->Create();

Затем идёт поиск грани для дальнейшего сопряжения в сборке:

flFaces = pPart->EntityCollection(o3d\_face);

for (int i = 0; i < flFaces->GetCount(); i++)

{

face = flFaces->GetByIndex(i);

def = face->GetDefinition();

if (def->GetOwnerEntity() == pRotate)

{

if (def->IsCylinder())

{

def->GetCylinderParam(&h, &r);

if (r == PuD / 2.f)

{

face->Putname("PuckCilind");

qDebug() << "PuckCilind";

face->Update();

}

}

else if (def->IsPlanar())

{

col = def->EdgeCollection();

for (int k = 0; k < col->GetCount(); k++)

{

ksEdgeDefinitionPtr d = col->GetByIndex(k);

if (d->IsCircle())

{

p1 = d->GetVertex(true);

p2 = d->GetVertex(false);

double x1, y1, z1;

p1->GetPoint(&x1, &y1, &z1);

if (x1 == 0)

{

face->Putname("PuckConnect");

qDebug() << "PuckConnect";

face->Update();

break;

}

}

}

}

}

}

Заключительный этап – сборка. Для её создания мы загружаем созданные ранее детали из файлов, получаем ранее найденные и поименованные рёбра, сопрягаем и сохраняем сборку.

pDoc = pKompasApp5->Document3D();

pDoc->Create(false, false);

pPart = pDoc->GetPart(pTop\_Part);

ksPartPtr pNest, pNut, pPuck;

pDoc->SetPartFromFile(nest\_pth.c\_str(), pPart, true);

pDoc->SetPartFromFile(nut\_pth.c\_str(), pPart, true);

pDoc->SetPartFromFile(puck\_pth.c\_str(), pPart, true);

pNest = pDoc->GetPart(0);

pNut = pDoc->GetPart(1);

pPuck = pDoc->GetPart(2);

entcol = pNest->EntityCollection(o3d\_face);

ksEntityPtr NestCilind = entcol->GetByName("NestCilind", true, true);

ksEntityPtr NestConnect = entcol->GetByName("NestConnect", true, true);

entcol = pNut->EntityCollection(o3d\_face);

ksEntityPtr NutCilind = entcol->GetByName("NutCilind", true, true);

ksEntityPtr NutConnect = entcol->GetByName("NutConnect", true, true);

entcol = pPuck->EntityCollection(o3d\_face);

ksEntityPtr PuckCilind = entcol->GetByName("PuckCilind", true, true);

ksEntityPtr PuckConnect = entcol->GetByName("PuckConnect", true, true);

pDoc->AddMateConstraint(mc\_Concentric, NestCilind, NutCilind, -1, 1, 0);

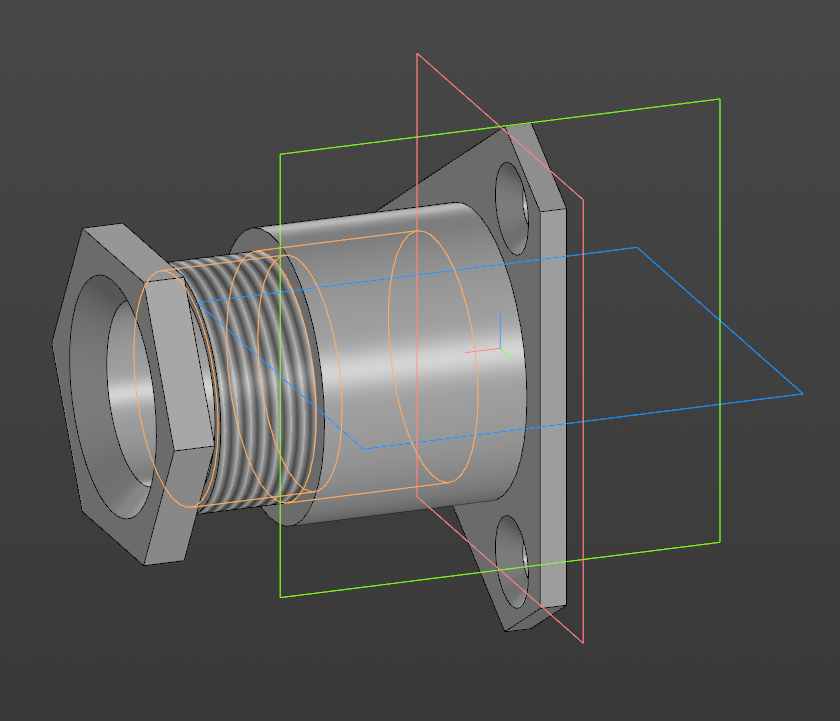
pDoc->AddMateConstraint(mc\_Concentric, NestCilind, PuckCilind, 1, 1, 0);

pDoc->AddMateConstraint(mc\_Distance, NestConnect, PuckConnect, 1, 1, -3);

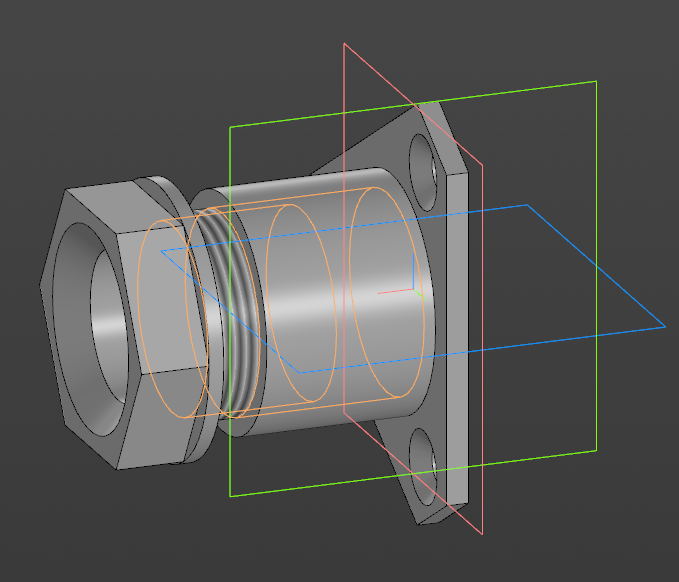
pDoc->AddMateConstraint(mc\_Distance, NestConnect, NutConnect, -1, 1, -AsL);

pDoc->SaveAs(assembling\_pth.c\_str());

Сборка с гайкой исполнения 1:



Сборка с гайкой исполнения 2:



# СПИСОК ЛИТЕРАТУРЫ:

1. *Детали машин* : атлас конструкций: учеб. пособие для студентов машиностроительных специальностей вузов / Б.А. Байков, В.Н. Богачев, А.В. Буланже и др.; под общ. ред. д-ра техн. наук проф. Д.Н. Решетова. – 5-е изд., переработ. и доп. – Москва : 1992 . –296 с.
2. *Норсеев Сергей.* Разработка приложений под компас в delphi : 2013. – 346 с.
3. *ООО «УСА».* Сальники одинарного и двойного действия : сайт. – Москва, 2023– . – URL: <https://goo.su/wbvn> (дата обращения 16.12.2023).
4. *Прогресс-м.* Муфты – сальники сксо-16, сксо-27, сксо-42, сксо-60, сксо-76, сксо-90 гост 4860.1–83, гост 4860.2-83 : сайт. – Москва, 2023. – URL: https://prog-m.ru/salniki-skso/ (дата обращения 16.12.2023).
5. *Сиал*. Манжета армированная (сальник) 190х230х15 : сайт. – Москва, 2023– . – URL: [https://goo.su/emn3qy](https://goo.su/EMn3Qy) (дата обращения 16.12.2023).
6. *Импульс*. Сальники - назначение, монтаж и эксплуатация : сайт. – Москва, 2023– . – URL: <https://impuls-ek.ru/info/articles/salniki_naznachenie_montazh_i_ekspluatatsiya/> (дата обращения 16.12.2023).
7. *Nvph*. Сальники и армированные манжеты : сайт. – Москва, 2023– . – URL: https://nvph.ru/slaniki-manjeti/ (дата обращения 16.12.2023).