СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 2](#_Toc165493982)

[1. Создание приложения Компас-3D с использованием Visual C++ и API 4](#_Toc165493983)

[1.1 Подготовка шаблона для разработки 4](#_Toc165493984)

[1.2 Проектирование компонентов изделия средствами API 6](#_Toc165493985)

[1.3 Создание вариантов исполнения сборки средствами API 14](#_Toc165493986)

[1.4 Создание различных исполнений компонентов сборки 21](#_Toc165493987)

[1.5 Подключение и разработка формы для приложения 23](#_Toc165493988)

[2. Создание приложения Компас-3D с использованием Компас-макро 32](#_Toc165493989)

[2.1 Инструменты используемые для разработки приложения 32](#_Toc165493990)

[2.2 Описание графического интерфейса программы 33](#_Toc165493991)

[2.3 Объяснение процесса генерации функций построения 36](#_Toc165493992)

[2.4 Тестирование 38](#_Toc165493993)

[3. Сравнение подходов к автоматизации проектирования 42](#_Toc165493994)

[3.1 Доступный функционал и интеграция в САПР-систему 42](#_Toc165493995)

[3.2 Проблемы возникшие в ходе разработки программ 45](#_Toc165493996)

[3.3 Итоговое сравнение подходов к автоматизации в САПР Компас-3D 46](#_Toc165493997)

[ЗАКЛЮЧЕНИЕ 48](#_Toc165493998)

# ВВЕДЕНИЕ

Компас 3D является интегрированным средством разработки и проектирования, предназначенным для создания трехмерных моделей механизмов, деталей, сборочных единиц и других объектов. Эта программа обладает широким функционалом, который позволяет создавать и редактировать 3D-модели, производить анализ и симуляцию моделей, а также создавать чертежи. Компас 3D используется в различных отраслях промышленности, в том числе в машиностроении, авиации, судостроении и других.

Однако, для многих задач необходима автоматизация процессов и интеграция с другими программами. Существуют разные способы автоматизировать работу в САПР-системе Компас 3D, одним из них является запись и возпроизведение макросов, написанных при помощи приложения Компас-макро на языке программирования Python. Помимо этого, есть возможность создавать собственные приложения, при помощи различных языков программирования, посредством API.

API (Application Programming Interface) - это интерфейс программирования приложений, который обеспечивает взаимодействие между программными системами. API Компас 3D позволяет программистам и разработчикам использовать функционал Компас 3D в своих собственных программах, автоматизировать процессы создания и редактирования 3D-моделей, а также интегрировать Компас 3D с другими программами.

Компас-макро – интегрерированная в САПР-систему Компас-3D среда разработки, использующая Python и поддерживающая запись большого количества операций, выполняемых пользователем. Предназачена для автоматизации рутинных задач и расширения функциональности.

В ходе выполнения данной работы будут созданы два приложения. Первое будет реализовано при помощи API версии 5, на языке C++, с интерфейсом, описанным в MFC-классах. Функционал, предоставляемый этим приложением позволит создавать различные варианты сборки изделия и его компонентов. Интерфейс включает в себя изображение всех видов изделий, краткое описание особенностей каждого варианта исполнения сборки. Различные элементы интерфейса позволяют пользователю осуществлять выбор нужного варианта и запускать построения. Второе приложение будет написано при помощи Компас-макро, на языке программирования - Python, оно будет объединять в себе макросы, записанные в САПР-системе для каждой детали и их вариантов исполнения. Интерейфс также будет содержать все элементы, позволяющие пользователю осуществить выбор изделия и запустить построения.

Целью данной работы является проведение анализа и выявления схожих черт и отличительных особенностей двух подходов к автоматизации. Для проведения анализа будут разработанны две шаблонные системы автоматизированного проектирования.

# 1. Создание приложения Компас-3D с использованием Visual C++ и API

# 1.1 Подготовка шаблона для разработки

Языком программирования, на котором будет написано API, является C++. САПР КОМПАС 3-D, содержит в своем каталоге файлов множество примеров библиотек на различных языках, помимо них, есть мастер создания библиотек для Visual C++. На рисунке 1 изображен подключенный шаблон в среду разработки.

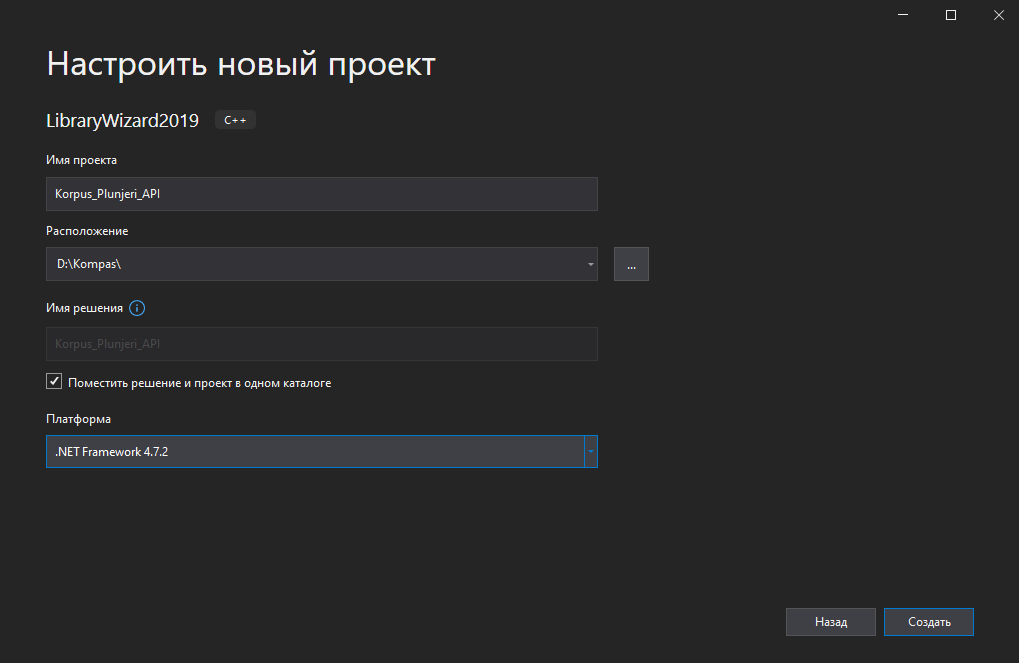


Рисунок 1 – Шаблон для разработки

Выберем этот шаблон, укажем имя проекта и папку для сохранения, после этого этапа перед нами открывается окно настройки шаблона.

Выберем автоматизацию и использование 3D, после выбора этих пунктов в заголовочном файле проекта сразу появятся строки подключения библиотек Kapi5.tlb и ksConstants3d.tlb, в них содержатся все необходимые классы и методы, функции, которые потребуются для разработки. Так как пути будут указаны относительно, следует перенести из SDK Компас-3D необходимые библиотеки в корень проекта. Помимо этого, перед тем, как приступать к разработке, необходимо удалить файлы, которые лежат не на одном уровне с заголовочным файлом stdafx.h. Мастер создания библиотек создает их по умолчанию, но в нашем проекте они не потребуются. На этом этапе, шаблон полностью настроен и можно приступать к созданию геометрии для компонентов изделия. Плюсом создания проекта через Library Wizard является то, что все настройки проекта в среде Visual Studio выставляются так, чтобы проект собирался в .rtw файл, который можно добавить в КОМПАС-3D в качестве приложения. Помимо этого, проект содержит большую часть настроек для того, чтобы в дальнейшем подключить форму к проекту.

# 1.2 Проектирование компонентов изделия средствами API

Создаем void функцию CreateDetails, в ней будут размещаться все операции, требуемые для создания деталей сборки. Перед созданием деталей, следует написать следующие строки:

KompasObjectPtr kompas;

HRESULT hRes = kompas.GetActiveObject(L"Kompas.Application.5");

if (FAILED(hRes))

kompas.CreateInstance(L"Kompas.Application.5");

kompas->Visible = true;

ksDocument3DPtr iDoc3D = kompas->Document3D();

iDoc3D->Create(false, true);

iDoc3D = kompas->ActiveDocument3D();

ksPartPtr Part1;

Part1 = iDoc3D->GetPart(pTop\_Part);

Здесь, мы объявляем указатель на Компас-3D, проверяем есть ли экземпляр программы, которая сейчас активна, если нет, то запускаем САПР. Делаем Компас-3d видимым для пользователя, создаем новую деталь, получаем на нее ссылку, объявляем указатель на эту деталь. Теперь, когда у нас есть пустое окно детали и доступ к нему, можем перейти к созданию эскиза, для построения окружности в плоскости XOY, используем следующие строки:

ksEntityPtr IsketchEntity = Part1->NewEntity(o3d\_sketch);

ksSketchDefinitionPtr sketchDef = IsketchEntity->GetDefinition();

sketchDef->SetPlane(Part1->GetDefaultEntity(o3d\_planeXOY));

IsketchEntity->Create();

ksDocument2DPtr Doc2D = sketchDef->BeginEdit();

Doc2D->ksCircle(0, 0, 40, 1);

sketchDef->EndEdit();

Здесь, мы создаем эскиз, используя ранее объявленный указатель, затем получаем определение эскиза, при помощи GetDefinition, указываем плоскость, в котрой будет располагаться наш эскиз, создаем его и приступаем к построениям, метод ksCircle имеет несколько параметров, 3 координаты точки центра окружности и радиус этой окружности, в этом методе они идут по порядку. Завершаем создание эскиза и переходим к выполнению операции вытягивания, для этой окружности:

ksEntityPtr Extr1 = Part1->NewEntity(o3d\_bossExtrusion);

ksBossExtrusionDefinitionPtr Extr1Def = Extr1->GetDefinition();

Extr1Def->SetSideParam(TRUE, etBlind, 20, 0, FALSE);

Extr1Def->directionType\* (dtNormal);

Extr1Def->SetSketch(IsketchEntity);

Extr1->Create();

В этом фрагменте мы создаем новую сущность вытягивания, получаем определение тем же методом GetDefifnition, указываем параметры операции, TRUE отвечает за направление вытягивания.

Для построения дальнейших эскизов и вытягиваний были созданы плоскости с различными смещениями по осям. Процесс создания плоскости со смещением описан ниже:

ksEntityPtr planeEntity5 = Part1->NewEntity(o3d\_planeOffset);

ksPlaneOffsetDefinitionPtr planeDef5 = planeEntity5->GetDefinition();

//// Устанавливаем базовую плоскость и смещение по оси Z

planeDef5->SetPlane(Part1->GetDefaultEntity(o3d\_planeYOZ)); planeDef5->direction = true; // true для смещения в положительном направлении оси Z

planeDef5->offset = 120; // Смещение на 10 единиц по оси Z

//// Создаем плоскость

planeEntity5->Create();

Для того, чтобы эскиз размещался в этой плоскости, следует в его параметре SetPlane задать указатель на смещенную плосость. Далее рассмотрим создание фасок и скруглений.

ksEntityPtr Fillet1 = Part1->NewEntity(o3d\_fillet);

ksFilletDefinitionPtr FilletDef2 = Fillet1->GetDefinition();

FilletDef2->tangent = FALSE;

FilletDef2->radius = 15;

ksEntityCollectionPtr EnColChamfer2 = FilletDef2->array();

// Выбираем грани и добавляем их в массив

ksEntityCollectionPtr EnColPart55 = Part1->EntityCollection(o3d\_edge);

EnColPart55->SelectByPoint(-40, 0, 0);

EnColChamfer2->Add(EnColPart55->First());

Fillet1->Create();

Выше расположен листинг создания скругления на одной грани. Здесь мы создаем объект скругления, задаем радиус.

Фаска строится похожим способом, заполняем массив граней, после создания объекта фаски и задания его параметров. Листинг создания фаски:

ksEntityPtr Chamfer2 = Part2->NewEntity(o3d\_chamfer);

ksChamferDefinitionPtr ChamferDef2 = Chamfer2->GetDefinition();

ChamferDef2->tangent = FALSE;

ChamferDef2->SetChamferParam(TRUE, 10, 10);

ksEntityCollectionPtr EnColPart3 = Part2->EntityCollection(o3d\_edge);

ksEntityCollectionPtr EnColChamfer3 = ChamferDef2->array();

EnColChamfer3->Clear();

EnColPart3->SelectByPoint(-180, 0, 376);

EnColChamfer3->Add(EnColPart3->GetByIndex(0));

Chamfer2->Create();

Для построения эскизов тел различной геометрической формы применяются принципы геометрии и тригонометрии. Листинг построения эскиза шестиугольника:

ksDocument2DPtr Doc2D1201 = sketchDef1201->BeginEdit();

Doc2D1201->ksCircle(-449, -218, 35, 1);

double sideLength8 = 15;

double xCenter8 = -449, yCenter8 = -218; // координаты центра шестиугольника

double angle8 = 3.14 / 6; // начальный угол

double x8[6], y8[6]; // координаты вершин шестиугольника

for (int i = 0; i < 6; i++) {

x8[i] = xCenter8 + sideLength8 \* cos(angle8);

y8[i] = yCenter8 + sideLength8 \* sin(angle8);

angle8 += 3.14 / 3;}

// Создаем линии для каждой стороны шестиугольника

for (int i = 0; i < 6; i++) {

Doc2D1201->ksLineSeg(x8[i], y8[i], x8[(i + 1) % 6], y8[(i + 1) % 6], 1);

}

sketchDef1201->EndEdit();

Все, описанные выше операции используются для создания каждой детали. Изменение их параметров, позволяет добиться получения нужных форм. Построения каждой детали завершаются сохранением в файл и закрытием активного документа. Для каждой детали, будет создан новый документ и отдельный файл. Чтобы упростить открытие проекта из разных мест на диске, было решено добавить переменную, которая будет содержать в себе путь. При сложении этой переменной и названия детали, будем получать абсолютный путь до нужной детали. Листинг описанных выше операций:

// Сохраняем текущую деталь

CString fullPathKorpus = m\_selectedPath + \_T("\\1.Korpus.m3d");

\_bstr\_t bstrPathKorpus(fullPathKorpus);

iDoc3D->SaveAs(bstrPathKorpus);

// Закрываем текущий документ

iDoc3D->close();

Также, для правильной работы сохранения, следует преобразовать строку в нужный формат BSTR. Для изменения цвета детали создана функция, SetPartColor, в которую передаются два параметра: цвет в формате RGB и объект детали.

void SetPartColor(ksPartPtr part, long color)

{

double ambient = 0.5; // Общий свет

double diffuse = 0.6; // Диффузия

double specularity = 0.8; // Зеркальность

double shininess = 0.8; // Блеск

double transparency = 1.0; // Прозрачность

double emission = 0.5; // Излучение

part->SetAdvancedColor(color, ambient, diffuse, specularity, shininess, transparency, emission);

part->Update();}

Метод SetAdvancedColor имеет множество параметры, отвечающих за визуальную часть детали. В функции, созданной для покраски деталей эти параметры имеют те же значения, что и в Компас-3d заданы по умолчанию, меняется только цвет. На рисунке 3 представлена деталь, при создании которой использовался весь описанный выше листинг.

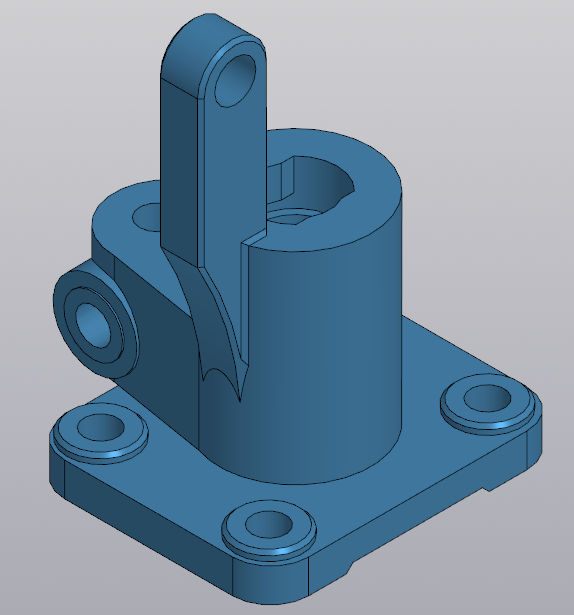


Рисунок 3 – Деталь 1.Korpus

Для создания геометрии пружины и шплинта используется операция вытягивания по траектории. На примере создания пружины, рассмотрим процесс реализации этой операции. Для начала создаем спираль в нулевой точке эскиза на базовой плоскости XOY.

ksEntityPtr Prujina1 = Part8->NewEntity(o3d\_cylindricSpiral);

ksCylindricSpiralDefinitionPtr PrujinaDef1 = Prujina1->GetDefinition();

PrujinaDef1->buildDir = TRUE;

PrujinaDef1->buildMode = 2;

PrujinaDef1->diam = 10;

PrujinaDef1->diamType = 0;

PrujinaDef1->firstAngle = 0;

PrujinaDef1->heightAddHow = TRUE;

PrujinaDef1->heightAdd = 0;

PrujinaDef1->height = 20;

PrujinaDef1->heightType = 0;

PrujinaDef1->step = 4;

PrujinaDef1->turnDir = TRUE;

PrujinaDef1->turn = 5;

PrujinaDef1->SetPlane(Part8->GetDefaultEntity(o3d\_planeXOY));

PrujinaDef1->SetLocation(0, 0);

Prujina1->Create();

Для вытягивания по направляющей, требуется эскиз сечения кинематического элемента. Создадим окружность диаметром 2 и приступим к описанию операции вытягягивания по направляющей:

ksEntityPtr Extr103 = Part8->NewEntity(o3d\_bossEvolution);

ksBossEvolutionDefinitionPtr Extr103Def = Extr103->GetDefinition();

Extr103Def->sketchShiftType = 1;

ksEntityCollectionPtr EnColPruj1 = Part8->EntityCollection(o3d\_edge);

Extr103Def->SetSketch(IsketchEntity103);

ksEntityCollectionPtr pathPartArray = Extr103Def->PathPartArray();

pathPartArray->Add(Prujina1);

ksThinParamPtr iThinParam = Extr103Def->ThinParam();

iThinParam->thin = false;

Extr103->Create();

Для завершения построения эскиза, остается сделать плоские части на витках при помощи выреза выдавливанием. На рисунке 4 изображена пружина, созданная при помощи вышеописанных операций.

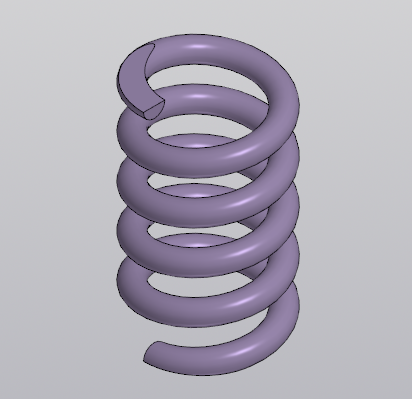


Рисунок 4 – 10.Prujina

Ранее представленный листинг используется для построения всех составных частей сборки. После завершения создания всех компонентов, переходим к созданию сборки изделия и позиционированию элементов.

# 1.3 Создание вариантов исполнения сборки средствами API

Для сборки объявим новую функцию CreateSborka, в этой функции мы также, как и с деталями получаем активный документ, если его нет, то создаем.

CoInitialize(NULL);

KompasObjectPtr kompas;

HRESULT hRes = kompas.GetActiveObject(L"Kompas.Application.5");

if (FAILED(hRes))

kompas.CreateInstance(L"Kompas.Application.5");

kompas->Visible = true;

ksDocument3DPtr iDoc3Dsb = kompas->Document3D();

iDoc3Dsb->Create(false, false);

iDoc3Dsb = kompas->ActiveDocument3D();

ksPartPtr Partsb = iDoc3Dsb->GetPart(pTop\_Part);

Ссылка на каждую деталь будет хранится в переменной, поэтому создадим 28 переменных с названиями деталей. На примере детали 1.Korpus рассмотрим процесс добавления компонента в сборку:

iDoc3Dsb->SetPartFromFile(bstrPathKorpus, Partsb, VARIANT\_FALSE);

Korpus = iDoc3Dsb->GetPart(0);

Korpus->fixedComponent = TRUE;

В методе SetPartFromFile есть 3 параметра, первый это абсолютный путь до детали, в нашем случае используем переменную, описанную ранее и которую мы использовали для сохранения деталей. Вторым параметром является указатель на деталь, в которую будет загружена деталь из файла, третий параметр отвечает за замену имеющейся детали на загружаемую, если

они совпадают, мы отключаем это поведение САПР системы, указав VARIANT\_FALSE.

После загрузки всех деталей в сборку, мы получаем внешне практически готовое изделие (Рисунок 5), так как детали строились в абсолютной системе координат и их не надо позиционировать при помощи сопряжений.

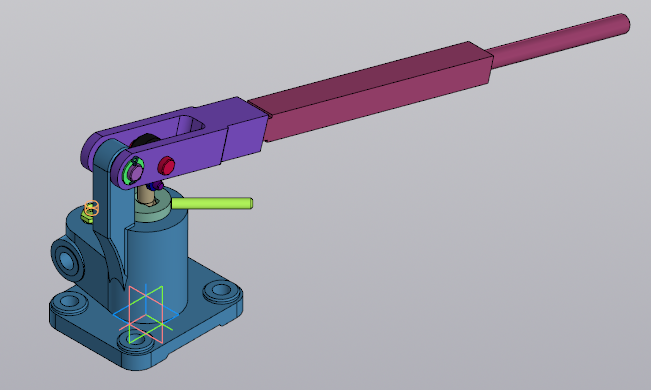


Рисунок 5 – Сборка без сопряжений

Для завершения сборки, остается добавить несколько деталей, которые представляют собой дубликаты уже имеющихся. Так как, для создания этой конструкции мы используем те же детали, мы не можем продублировать построения в абсолютной системе координат для этой части. Повторно загрузим недостающие детали ранее описанным способом и приступим к созданию сопряжений. Для раздачи имен и граням была создана функция, которая будет давать им названия, исходя из имени детали, которой они принадлежат, их площади и ориентации. Листинг функции для назначения имен граням:

void SetFaceNames(ksPartPtr part, const wchar\_t\* partName)

{

ksEntityCollectionPtr faces = part->EntityCollection(o3d\_face);

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

{

ksEntityPtr face = faces->GetByIndex(i);

ksFaceDefinitionPtr faceDef = face->GetDefinition();

double area = faceDef->GetArea(ST\_MIX\_MM);

VARIANT\_BOOL normalOrient = faceDef->GetnormalOrientation();

std::wstringstream ss;

ss << area << "\_" << normalOrient;

std::wstring faceName = std::wstring(partName) + L" Gran" + ss.str();

face->Putname(faceName.c\_str());

face->Update();

}

}

В эту функцию мы передаем деталь и ее имя. Цикл перебирает все грани детали, определяет их площадь и ориентацию, имя назначается в таком формате: 1.Korpus Gran24457.9\_-1.

Для фрагментов кода содержащих построения корпуса и плунжера добавим вызов функции именования граней. Теперь можно переходить к созданию сопряжений. Листинг задания сопряжения «Концентричность»:

col = Korpus->EntityCollection(o3d\_face);

ksEntityPtr granKorpus\_2 = col->GetByName("1.Korpus Gran24457.9\_-1", true, true);

col = Plunjer6->EntityCollection(o3d\_face);

ksEntityPtr granPlunjer6\_2 = col->GetByName("6.Plunjer6 Gran1985.47\_-1", true, true);

iDoc3Dsb->AddMateConstraint(mc\_Concentric, granKorpus\_2, granPlunjer6\_2, 1, 1, 0);

Сначала создадим переменную, которая будет содержать в себе грани для сопряжения, затем из коллекции граней каждого тела вы выбираем нужные грани по имени, которое мы задали при помощи функции. Далее воспользуемся методом AddMateConstraint, в него передаем тип сопряжения, две сопрягаемые грани тел, а также позиционирование в рамках сопряжения.

Для сопряжения типа «Совпадение» процесс аналогичный, получаем грани, которые будут участвовать в сопряжении и передаем их в метод AddMateConstraint. Меняем тип сопряжения на ms\_Coincidence, передаем переменные с нужными гранями и указываем, какое тело будет смещаться к другому телу для совпадения. Здесь, -1 соответствует фиксированному положению, а 1 смещению. Листинг задания сопряжения типа «Совпадение» представлен ниже:

ksEntityCollectionPtr col = Korpus->EntityCollection(o3d\_face);

ksEntityPtr granKorpus\_1 = col->GetByName("1.Korpus Gran4196.55\_-1", true, true);

col = Plunjer6->EntityCollection(o3d\_face);

ksEntityPtr granPlunjer\_1 = col->GetByName("6.Plunjer6 Gran1033.58\_0", true, true);

iDoc3Dsb->AddMateConstraint(mc\_Coincidence, granKorpus\_1, granPlunjer\_1, -1, 1, 0);

На этом этапе создание геометрии изделия завершено, можно подключать проект в Компас 3D в качестве приложения и получать спроектированное изделие при запуске (Рисунок 6).

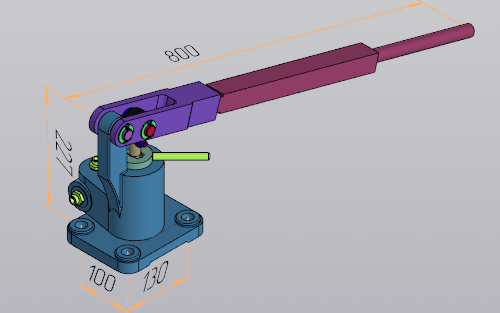


Рисунок 6 – Базовый вариант изделия

Для разработки различных вариантов исполнения потребуется переменная, в зависимости от значения которой будут выполняется те или иные блоки кода, обернутые в условные операторы. Ниже представлен листинг применения этой переменной и условных оператор для выполнения дополнительных построений на корпусе:

if (VariantSborki == 1) {

ksEntityPtr planeEntity700 = Part1->NewEntity(o3d\_planeOffset);

ksPlaneOffsetDefinitionPtr planeDef700 = planeEntity700->GetDefinition();

planeDef700->SetPlane(Part1->GetDefaultEntity(o3d\_planeXOY));

planeDef700->direction = true;

planeDef700->offset = -20;

//// Создаем плоскость

planeEntity700->Create();

}

ksDocument2DPtr Doc2D = sketchDef->BeginEdit();

Doc2D->ksLineSeg(-87.5, -72, 87.5, -72, 1);

sketchDef->EndEdit();

ksEntityPtr Extr1 = Part1->NewEntity(o3d\_bossExtrusion);

ksBossExtrusionDefinitionPtr Extr1Def = Extr1->GetDefinition();

Extr1Def->SetSideParam(TRUE, etBlind, 40, 0, FALSE);

Extr1Def->directionType\* (dtNormal);

Extr1Def->SetSketch(IsketchEntity);

Extr1->Create();

Тем же способом отсекаем построения, которые не будут использоваться в конкретнов варианте исполнения.

Грани, участвующие в сопряжениях, не были затронуты, поэтому для каждой сборки будут применимы сопряжения из базового варианта исполнения. Меняем переменную, отвечающую за построения в условных операторах от 0 до 2, собираем проект и запускаем построения. На рисунке 7, 8 изображены различные варианты исполнения изделия.

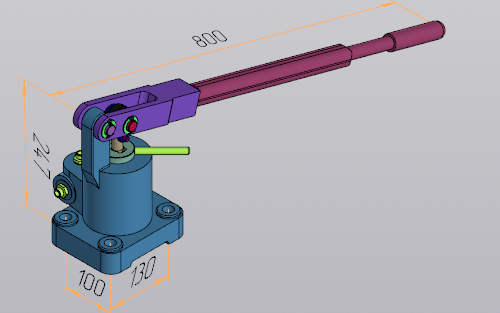


Рисунок 7 – Второй вариант изделия

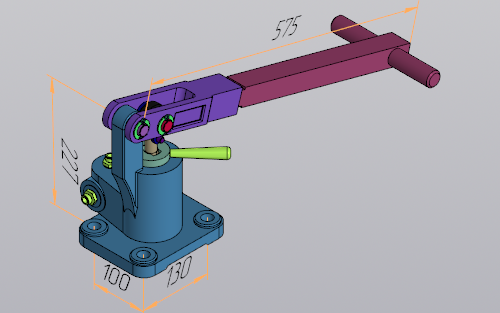


Рисунок 8 – Третий вариант изделия

# 1.4 Создание различных исполнений компонентов сборки

По аналогии, с созданием вариантов сборки, создадим переменную, которая будет отвечать за построения для каждой детали. Для каждой детали реализованы 3 варианта исполнения. Ниже, рассмотрим фрагмент кода, отвечающий за дополнительные построения для одного из вариантов исполнения рукоятки:

if (VariantDetali == 2) {

ksEntityPtr Extr122 = Part13->NewEntity(o3d\_bossExtrusion);

ksBossExtrusionDefinitionPtr Extr122Def = Extr122->GetDefinition();

Extr122Def->SetSideParam(TRUE, etBlind, 87, 2.5, FALSE);

Extr122Def->directionType = dtNormal;

Extr122Def->SetSketch(IsketchEntity122);

Extr122->Create();

ksEntityPtr Fillet780 = Part13->NewEntity(o3d\_fillet);

ksFilletDefinitionPtr FilletDef780 = Fillet780->GetDefinition();

FilletDef780->tangent = FALSE;

FilletDef780->radius = 5;

ksEntityCollectionPtr EnColChamfer780 = FilletDef780->array();

// Выбираем грани и добавляем их в массив

ksEntityCollectionPtr EnColPart780 = Part13->EntityCollection(o3d\_edge);

EnColPart780->SelectByPoint(-71.982411102196, 58.125236636129, 0);

EnColChamfer780->Add(EnColPart780->Last());

Fillet780->Create();

}

Для всех деталей применим аналогичный подход. Получаем 23 функции, содержащие построения каждой детали. Отличие от функции для сборки, заключается в том, что здесть мы не будем применять функцию для именования граней деталей, а также не будем сохранять их в файл. Сохранение нужного варианта пользователь будет делать по своему усмотрению. На рисунках 9,1 0 изображены 2 варианта исполнения рукоятки.

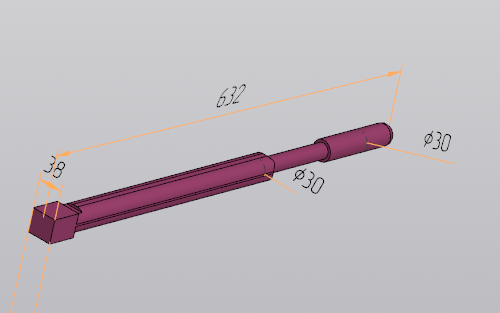


Рисунок 9 – 17.Rukouatka исполнение 1

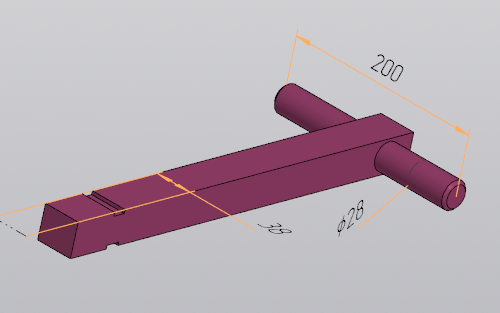


Рисунок 10– 17.Rukouatka исполнение 1

# 1.5 Подключение и разработка формы для приложения

Перед тем, как приступать к разработке интерфейса следует проверить, содержатся ли строки \_UNICODE и UNICODE в определении препроцессора для текущей конфигурации. Для сборки проекта под UNICODE должны быть такие настройки (Рисунок 6).

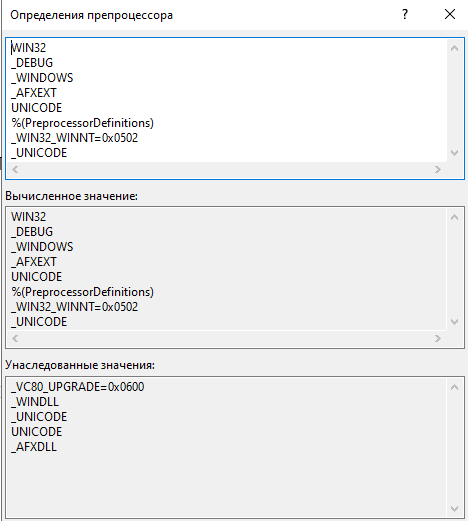


Рисунок 6 – Определения препроцессора для использования MFC

Сборка проекта под UNICODE требуется для того, чтобы не возникали вылеты при запуске приложения в КОМПАС-3D, при использовании MFC – классов. Также, при работе со строками следует использовать методы, которые будут корректно работать под UNICODE. Например, перед строковыми значениями нужно добавлять литерал L.

Переходим к созданию класса MFC для нашей формы. При добавлении класса MFC нажатием на кнопку «добавить» и выбором из списка доступных заготовок возникает ошибка. Поэтому создаем вручную файл MainWindow.h и переносим его в проект. Ниже представлен листинг класса, который потребуются для реализации всего запланированного функционала:

class TestDlg : public CDialog

{

// Construction

public:

TestDlg(CWnd\* pParent = NULL); // standard constructor

// Dialog Data

//{{AFX\_DATA(TestDlg)

enum { IDD = IDD\_DIALOG1 };

CStatic m\_show;

CStatic m\_selectedPathCtrl;

CComboBox m\_myCombobox;

CComboBox m\_myCombobox1;

//}}AFX\_DATA

// Overrides

// ClassWizard generated virtual function overrides

//{{AFX\_VIRTUAL(TestDlg)

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

virtual BOOL OnInitDialog();

//}}AFX\_VIRTUAL

// Implementation

protected:

// Generated message map functions

//{{AFX\_MSG(TestDlg)

afx\_msg void ButtonCreateIzd();

afx\_msg void OnBnClickedButtonBrowse();

afx\_msg void OnBnClickedRadio1();

afx\_msg void OnBnClickedRadio2();

afx\_msg void OnBnClickedRadio3();

afx\_msg void OnCbnSelchangeMyComboboxForDetails();

afx\_msg void OnCbnSelchangeSelectVariantDetali();

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP() };

TestDlg - это класс диалогового окна, производный от CDialog. Он содержит элементы управления, такие как CStatic, CComboBox, которые представляют собой статический текст и выпадающий список соответственно. В этом классе также определены некоторые функции-обработчики сообщений, такие как ButtonCreateIzd(), OnCbnSelchangeMyCombobox(), OnBnClickedButtonBrowse(), которые вызываются при определенных событиях пользовательского интерфейса.

Для создания кнопок, изображений и выпадающих списков на форме, требуется в .rc файле добавить их описание, это выглядит следующим образом:

IDD\_DIALOG1 DIALOGEX 0, 0, 463, 300

STYLE DS\_SYSMODAL | DS\_SETFONT | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION

"Шаблонная система автоматизированного проектирования насоса двухплунжерного"

FONT 8, "Tahoma", 0, 0, 0x1

BEGIN

PUSHBUTTON "Создать",IDC\_MY\_BUTTON,377,254,61,25

LTEXT "Выбранный путь:",IDC\_SELECTED\_PATH,7,268,161,8

PUSHBUTTON "Путь сохранения деталей...",IDC\_BUTTON\_BROWSE,7,279,106,14

CONTROL "",IDC\_SHOW\_STATIC,"Static",SS\_BITMAP,7,7,150,150,WS\_EX\_CLIENTEDGE

CONTROL "Базовый вариант исполнения",IDC\_RADIO1,"Button",BS\_AUTORADIOBUTTON,7,225,116,10

CONTROL "Расширенный корпус и штуцеры, конструкция рукоятки изменена",IDC\_RADIO2,

"Button",BS\_AUTORADIOBUTTON,7,239,241,10

CONTROL "Двухстороняя рукоятка, пазы на вилке, рукоятка для спуска усилена",IDC\_RADIO3,

"Button",BS\_AUTORADIOBUTTON,7,253,257,10

LTEXT "Выберите вариант исполнения изделия:",IDC\_STATIC,7,212,138,8

COMBOBOX IDC\_COMBO1,345,50,113,30,CBS\_DROPDOWN | CBS\_SORT | WS\_VSCROLL | WS\_TABSTOP

LTEXT "Выбор детали",IDC\_STATIC,345,40,71,8

COMBOBOX IDC\_COMBO2,346,88,112,30,CBS\_DROPDOWN | CBS\_SORT | WS\_VSCROLL | WS\_TABSTOP

LTEXT "Создание компонентов сборки",IDC\_STATIC,346,21,105,8

LTEXT "Выбор варианта исполнения",IDC\_STATIC,345,79,98,8

END

В этом коде мы описываем типы элементов, которые будут расположены на форме, а также их положения. Помимо этого, для каждого элемента следует добавить идентификатор ресурсов, которые будут служить для связи элементов формы с нашим приложением. Описание идентификаторов выполняем в файле Resource.h:

#define IDI\_Korpus\_Plunjeri\_API 1

#define IDS\_STRING4 2

#define IDS\_STRING5 6

#define IDR\_LIBID 100

#define IDR\_AUTO\_LIB 101

#define IDC\_RADIO1 1000

#define IDC\_RADIO2 1001

#define IDC\_RADIO3 1002

#define IDC\_COMBO1 1004

#define IDC\_COMBO2 1005

Переходим к реализации функций обработчиков для окна приложения. Ниже приведен листинг функции, которая отвечает за отрисовку окна:

void TestShowDialog() {

// по умолчанию ресурсы берутся из .exe файла, для того чтобы их достать

// из нашей dll вызовем этот define

AFX\_MANAGE\_STATE(AfxGetStaticModuleState());

TestDlg\* pDialog = new TestDlg();

if (pDialog) {

pDialog->Create(IDD\_DIALOG1, NULL);

pDialog->ShowWindow(SW\_SHOW);

}

}

Вызов этой функции необходимо разместить в точке входа в наш проект, для того, чтобы окно вызывалось сразу после запуска:

void WINAPI LIBRARYENTRY(unsigned int comm) {

// по умолчанию ресурсы берутся из .exe файла, для того чтобы их достать

// из нашей dll вызовем этот define

AFX\_MANAGE\_STATE(AfxGetStaticModuleState());

switch (comm)

{

case 1:

{

GetKompas();

if (kompas)

TestShowDialog();

break;

}}}

Теперь, при запуске приложения из КОМПАС-3D, будет открываться окно программы, содержащее все описанные ранее элементы. Для того чтобы, у этих элементов появились обработчики событий необходимо связать их с программой при помощи следующего кода:

void TestDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialog::DoDataExchange(pDX);

//{{AFX\_DATA\_MAP(TestDlg)

DDX\_Control(pDX, IDC\_COMBO1, m\_myCombobox);

DDX\_Control(pDX, IDC\_COMBO2, m\_myCombobox1);

DDX\_Control(pDX, IDC\_SHOW\_STATIC, m\_show);

DDX\_Control(pDX, IDC\_SELECTED\_PATH, m\_selectedPathCtrl);

//}}AFX\_DATA\_MAP

}

BEGIN\_MESSAGE\_MAP(TestDlg, CDialog)

//{{AFX\_MSG\_MAP(TestDlg)

ON\_BN\_CLICKED(IDC\_MY\_BUTTON, ButtonCreateIzd)

ON\_WM\_PAINT()

ON\_BN\_CLICKED(IDC\_BUTTON\_BROWSE, &TestDlg::OnBnClickedButtonBrowse)

ON\_BN\_CLICKED(IDC\_RADIO1, &TestDlg::OnBnClickedRadio1)

ON\_BN\_CLICKED(IDC\_RADIO2, &TestDlg::OnBnClickedRadio2)

ON\_BN\_CLICKED(IDC\_RADIO3, &TestDlg::OnBnClickedRadio3)

ON\_CBN\_SELCHANGE(IDC\_COMBO1, &TestDlg::OnCbnSelchangeMyComboboxForDetails)

ON\_CBN\_SELCHANGE(IDC\_COMBO2, &TestDlg::OnCbnSelchangeSelectVariantDetali)

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

DoDataExchange(CDataExchange\* pDX)- это функция, которая используется для установления связи между элементами управления в диалоговом окне и переменными членами класса. Это делается с помощью макросов DDX\_Control. Каждый из этих макросов связывает элемент управления, идентифицированный по его идентификатору ресурса.

В функцию-обработчик загрузки окна программы мы размещаем код отрисовки изображения в формате .bmp на Control из .rc файла, для этого предварительно необходимо добавить изображение в проект и связать с ним идентификатор ресурсов, при помощи этой строки:

IDB\_MY\_BITMAP BITMAP "res\\14\_Sedlo3.bmp"

Помимо этого, в функции обрабатывающей запуск окна, реализуем заполнение CCombobox значениями и выбор первого Radiobutton при запуске программы.

RadioButton реализованы в программе для того, чтобы пользователь мог переключатся между вариантами исполненя изделия. Нажатие на них вызывает функцию, переключающее изображение на Control и меняющее значение переменной.

Для упрощения запуска приложения на других компьютерах, а также для возможности самому указывать путь до файлов изделия создадим функцию-обработчик на кнопку «Обзор», здесь, мы по нажатию на эту кнопку вызываем диалоговое окно «проводника», после выбора папки, путь до нее сохраняется в переменную, которую мы ранее использовали для указания пути сохранения и загрузки деталей, также, чтобы пользователь мог видеть путь в окне программы, мы передаем значение пути в переменную текстового поля. Листинг обработчика указания папки для сохранения и загрузки:

void TestDlg::OnBnClickedButtonBrowse()

{

BROWSEINFO bi = { 0 };

bi.lpszTitle = \_T("Выберите папку:");

LPITEMIDLIST pidl = SHBrowseForFolder(&bi);

if (pidl != 0)

{

// Получаем путь из выбранного ID списка элементов

TCHAR path[MAX\_PATH];

if (SHGetPathFromIDList(pidl, path))

{

m\_selectedPath = path; // Сохраняем путь в переменной

m\_myCombobox.AddString(path);

m\_selectedPathCtrl.SetWindowText(path);

}

// Освобождаем память

IMalloc\* imalloc = 0;

if (SUCCEEDED(SHGetMalloc(&imalloc)))

{

imalloc->Free(pidl);

imalloc->Release();}}}

На этом этапе разработка приложения можно считать завершенной, собираем проект, подключаем его в КОМПАС-3D. На рисунках 11, 12 изображен результат работы программы, а также окно, создание которого описано выше.

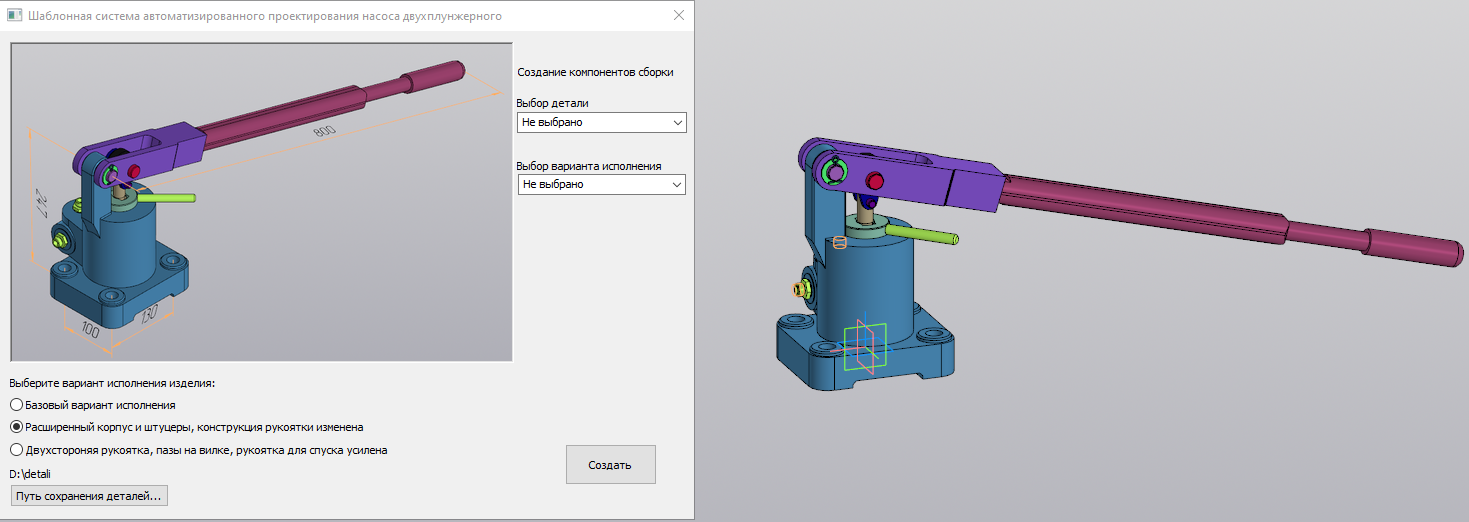


Рисунок 11 – Проектирование изделия средствами API

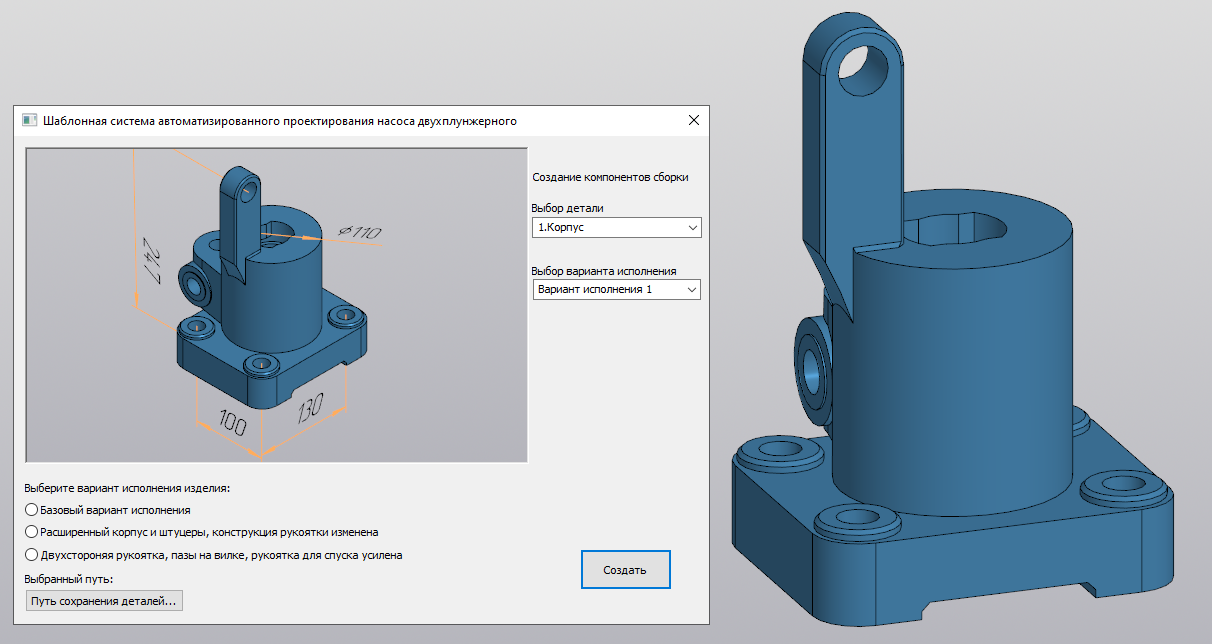


Рисунок 12 – Проектирование детали средствами API

# 2. Создание приложения Компас-3D с использованием Компас-макро

# 2.1 Инструменты используемые для разработки приложения

Для начала написания программы первым делом следует выбрать среду разработки. Поскольку в качестве языка программирования был выбран Python, следует использовать программное обеспечение Pycharm. Среда разработки PyCharm обладает рядом особенностей, таких как интеллектуальное завершение кода, проверка ошибок на лету, быстрые исправления и многое другое. Она также обеспечивает поддержку различных фреймворков, таких как Django, Flask, Google App Engine, Pyramid и других, что делает ее универсальным инструментом для веб-разработки.

PyCharm позволяет разработчикам работать с Python на различных операционных системах, включая Linux, Windows и MacOS. Благодаря своим возможностям, PyCharm является популярным выбором для разработки в области науки о данных и веб-разработки.

Использование PyCharm в совместной работе с Python обеспечивает удобную и эффективную среду для разработки, тестирования и развертывания приложений.

Для создания графического интерфейса в программе была использована библиотека для python, под названием tkinter, которая предоставляет следующие возможности:

* создание графических окон и элементов интерфейса;
* использование виджетов, таких как метки, кнопки, поля ввода, флажки, переключатели и другие;
* организация сложных макетов с использованием рамок (frame);
* работа с изображениями и графическими элементами;
* обработка событий, таких как нажатия кнопок и ввод данных;
* интеграция с другими библиотеками и инструментами разработки.

# 2.2 Описание графического интерфейса программы

Для создания графического интерфейса был создан экземпляр окна Window. Далее в этом окне были размещены элементы рамок Frame, которые помогают размещать элементы относительно верхнего левого угла рамки, а не угла основного окна. Рамка помогает облегчить размещение элементов в поле главного окна программы.

Было создано две рамки, каждая рамка содержит свои элементы. Рамка «Выбор детали» обеспечивает пользователя возможностью выбора детали, которую ему необходимо построить. Далее происходит выбор стиля, с которым будет построена та или иная деталь. Для каждой детали есть ровно по три стиля построения. Либо это изменение размеров деталей, либо это изменение формы самой детали. Чтобы сделать выбор достаточно раскрыть первый Combobox и выбрать название детали. Далее следует открыть второй Combobox, расположенный в верхнем правом углу программы и выбрать стиль построения выбранной детали. Рядом с двумя элементами выбора располагаются ярлыки Label, которые подсказывают пользователю действия.

Вторая рамка «Размеры» содержит в себе 20 ячеек, расположенных в виде таблицы. Сама ячейка является элементом Label. Содержимое ячеек пользователю менять не предоставляется возможности, то есть данная таблица содержит информацию о размерах и доступна только для чтения. У каждой ячейки была задана граница черного цвета толщиной в 2 пикселя, чтобы было визуальное разграничение информации.

Значительную правую часть окна занимает изображение. По умолчанию при запуске будет выбрано фото, в котором черным текстом по белому фону написано о том, что деталь не выбрана. Это можно увидеть на рисунке 13. При смене детали в первом Combobox происходит смена изображения на соответствующее детали.

Также графическому окну было дано название и выбрано изображение значка в виде компаса, располагаемом в верхней рамке слева.

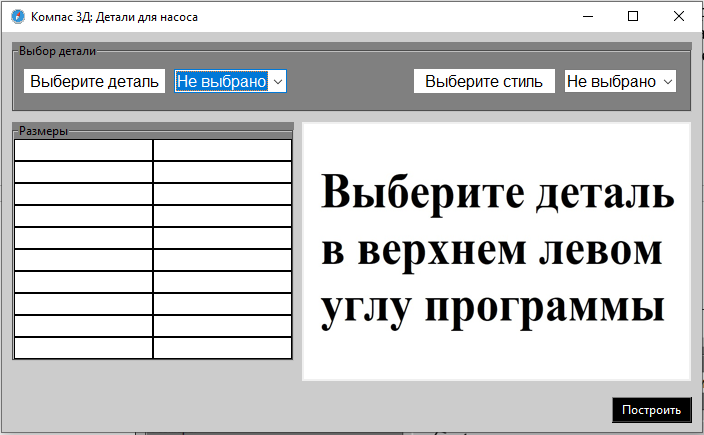


Рисунок 13 – Графическое окно программы

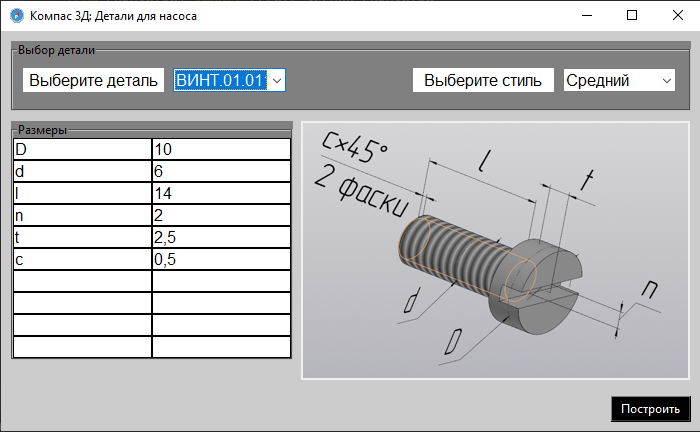


Рисунок 14 – Графическое окно с выбранной деталью «Винт.01.011»

На рисунке 14 представлена деталь, для которой существует три размера детали, а именно маленький, средний и большой. Смысл названия стилей исполнения данной детали заключается в том, что при построении меняется масштаб исполнения, но все формы идентичны друг другу. На рисунках 15 и 16 представлены два стиля исполнения другой детали. Стандартный вариант предназначен для рожкового или накидного ключа размером 24 мм, а второй стиль исполнения предназначен для ключа на 32 мм.

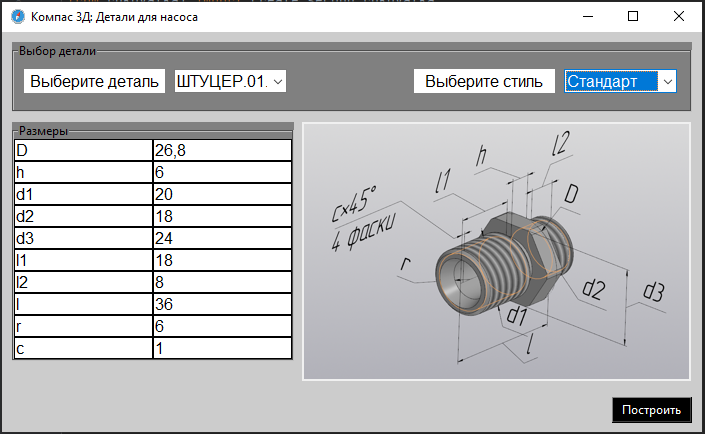


Рисунок 15 – Графическое окно со стандартным вариантом детали

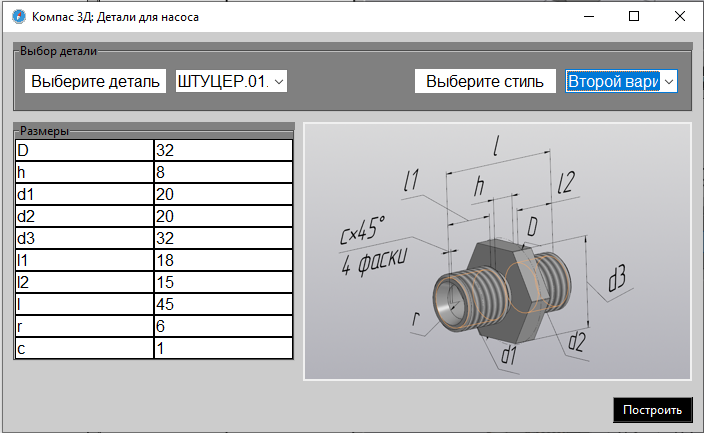


Рисунок 16 – Графическое окно с вторым стилем исполнения детали

# 2.3 Объяснение процесса генерации функций построения

Генерация функций построения с помощью встроенного в Компас приложения КОМПАС-МАКРО представляет собой процесс создания программных макросов для автоматизации различных задач в системе КОМПАС.

Для того чтобы записать макрос необходимо перейти на главную страницу "Компас 3D" и в верхней панели раскрыть контекстное меню «Приложения». Далее следует выбрать "Компас-Макро". С этого момента начинается запись файла макроса, то есть все действия, производимые в приложении, будут записаны в макрос. Поэтому каждую операцию и отдельно взятое действие стоит производить обдуманно, чтобы не увеличивать генерируемый файл на несколько десятков строк кода.

После включения записи макроса необходимо создать деталь. Далее происходит обычная процедура построения тел с применением различных методов и операций. Самыми распространенными являются "вытягивание" и "вращение" эскизов. С помощью булевых операций, которые уже имеются в стандартных операциях, происходит соединение всех тел в одно целое или же производятся необходимые отверстия, вырезы и углубления.

После завершения построения детали нужно произвести остановку записи. Делается это в том же месте, где производился запуск. Далее необходимо узнать директорию, где располагается макрос. Чтобы это сделать необходимо снова перейти на главную страницу приложения "Компас 3D". Снова нажать кнопку записи макроса в контекстном меню. Нажать "Начать запись». После этого откроется окно "Проводника" операционной системы с директорией, в которой сохраняются все макросы. Далее нужно выбрать файл сгенерированного макроса и добавить его в директорию с проектом в среде разработки Pycharm. После нужно изменить расширение файла с ".pym" на ".py". Далее необходимо выделить весь код, сделать табуляцию всех строк с помощью клавиши "Tab" и добавить в начало строку без отступов и табуляций. В строке содержится определение функции, например, "def create\_detail():". После этого следует импортировать получившуюся функцию в главный файл программы на языке Python. Теперь при выполнении данной функции будет происходить повторение всех записанных действий в макросе. Запуск программы должен осуществляться с запущенным окном приложения "Компас 3D".

# 2.4 Тестирование

Проверим работоспособность реализованного приложения. Для этого запустим его и далее выберем модель, которую необходимо построить. Для этого следует нажать на первый выпадающий список и выбрать нужную деталь. Выбор детали сопровождается заполнением таблицы с размерами пустыми величинами, поскольку не был еще выбран стиль построения детали. Результат показан на рисунке 17.

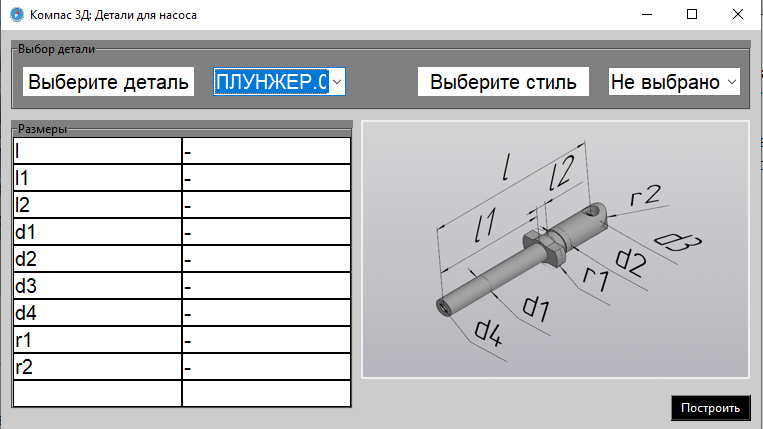


Рисунок 17 – Окно приложения с выбранной деталью

Далее следует выбрать стиль построения детали. Это действие сопровождается изменением размеров в таблице. Был выбран средний размер детали. Результат показан на рисунке 18.

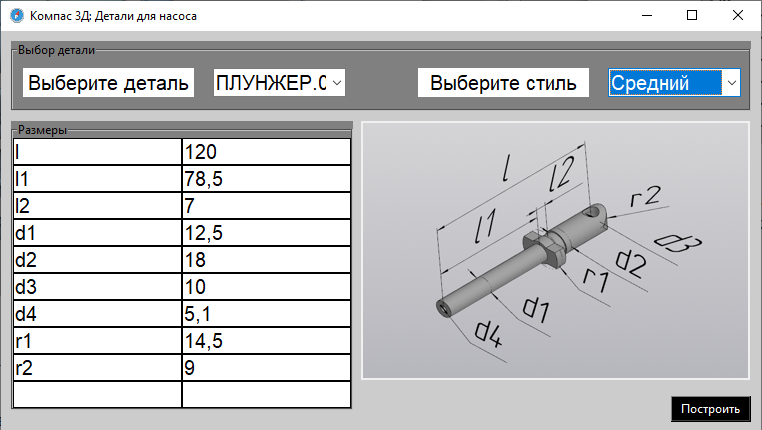


Рисунок 18 – Окно приложения с выбранным стилем построения

Выбранная деталь была построена в приложении Компас 3D. Результат представлен на рисунке 19.

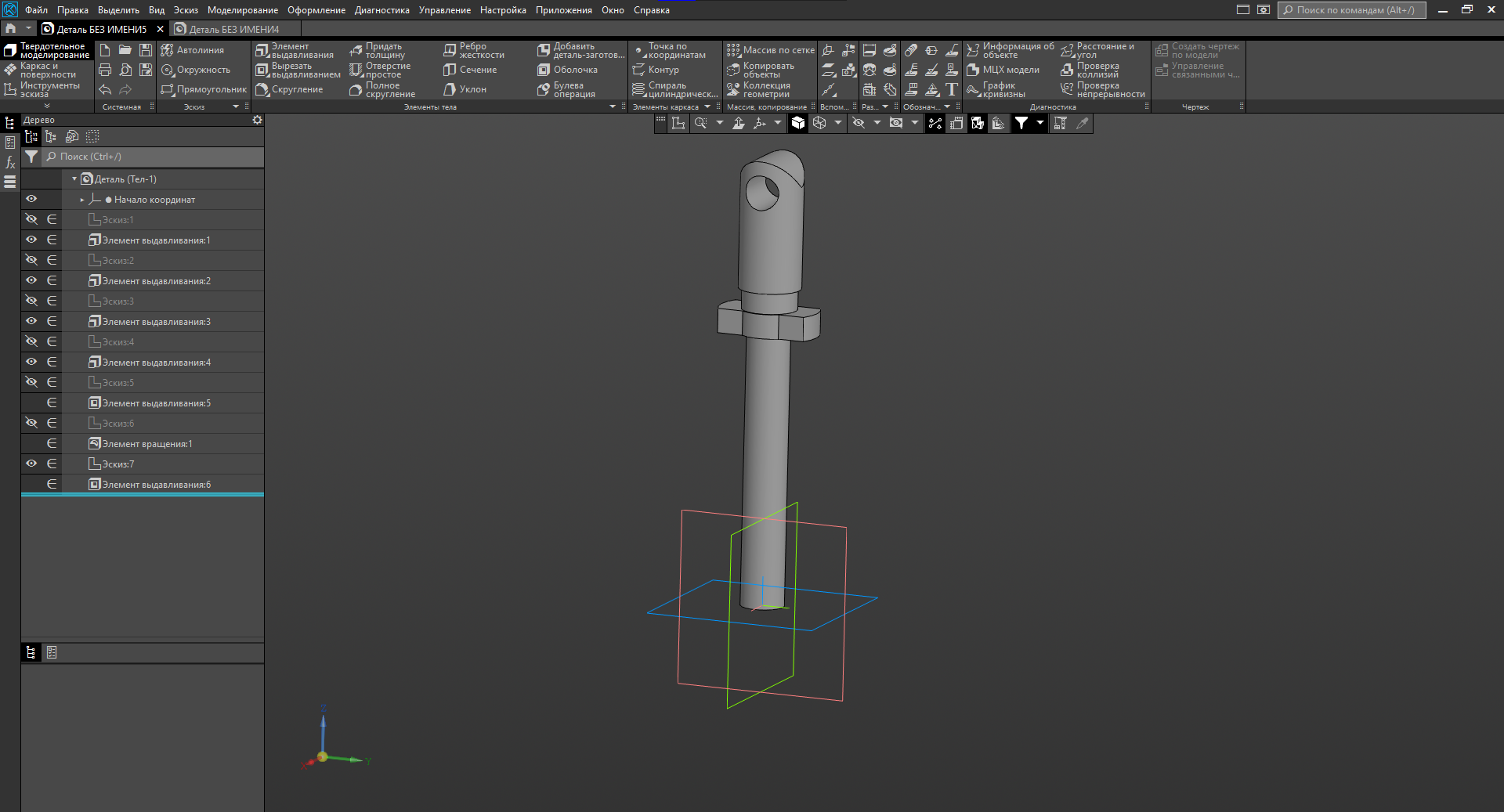


Рисунок 19 – Результат построения выбранной детали в Компас 3D

Теперь выполним построение детали, для которой существуют разные стили построения. Выполним построение корпуса двухплунжерного насоса. Для этого выберем в первом выпадающем списке деталь. Результат представлен на рисунках 20 и 21.

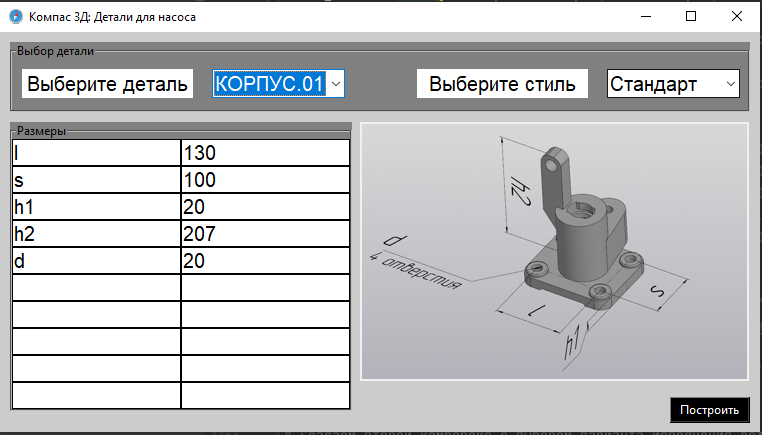


Рисунок 20 – Окно приложения со стандартной деталью Корпус.01.001

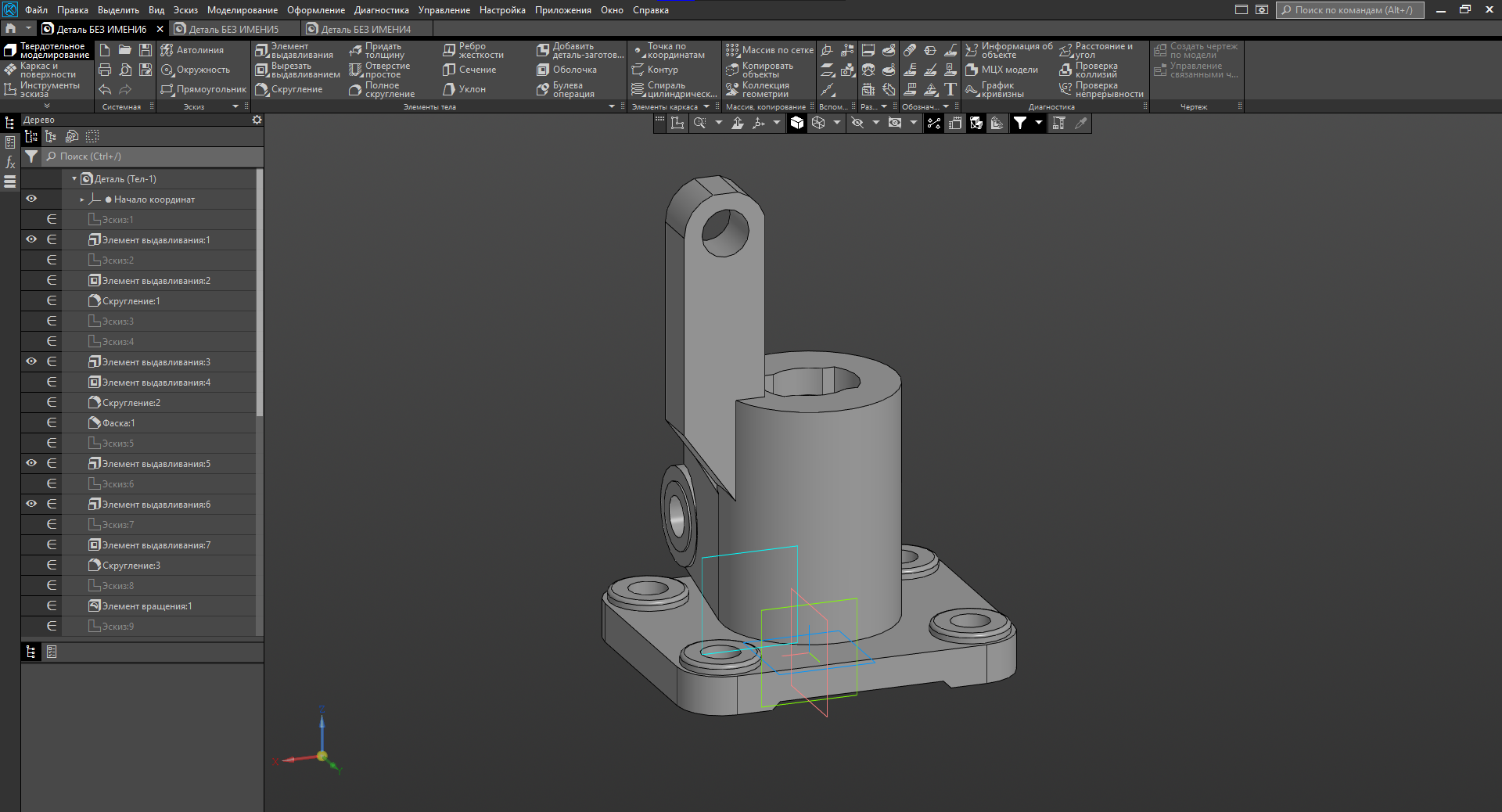


Рисунок 21 – Результат построения стандартной детали Корпус.01.001

Теперь выполним построение этой же детали, то есть оставляем выбор в первом выпадающем списке без изменения. Во втором списке выбираем второй вариант исполнения. Как видно на рисунке 23, изменился рисунок детали в окне пользовательского интерфейса. Соответственно будет построена такая же деталь в приложении Компас 3D. Результат построения показан на рисунке 24.

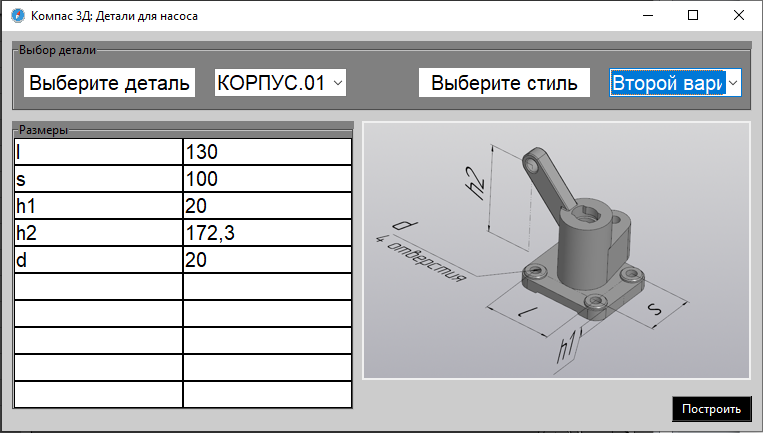


Рисунок 23 – Окно приложения со вторым вариантом детали Корпус.01.001

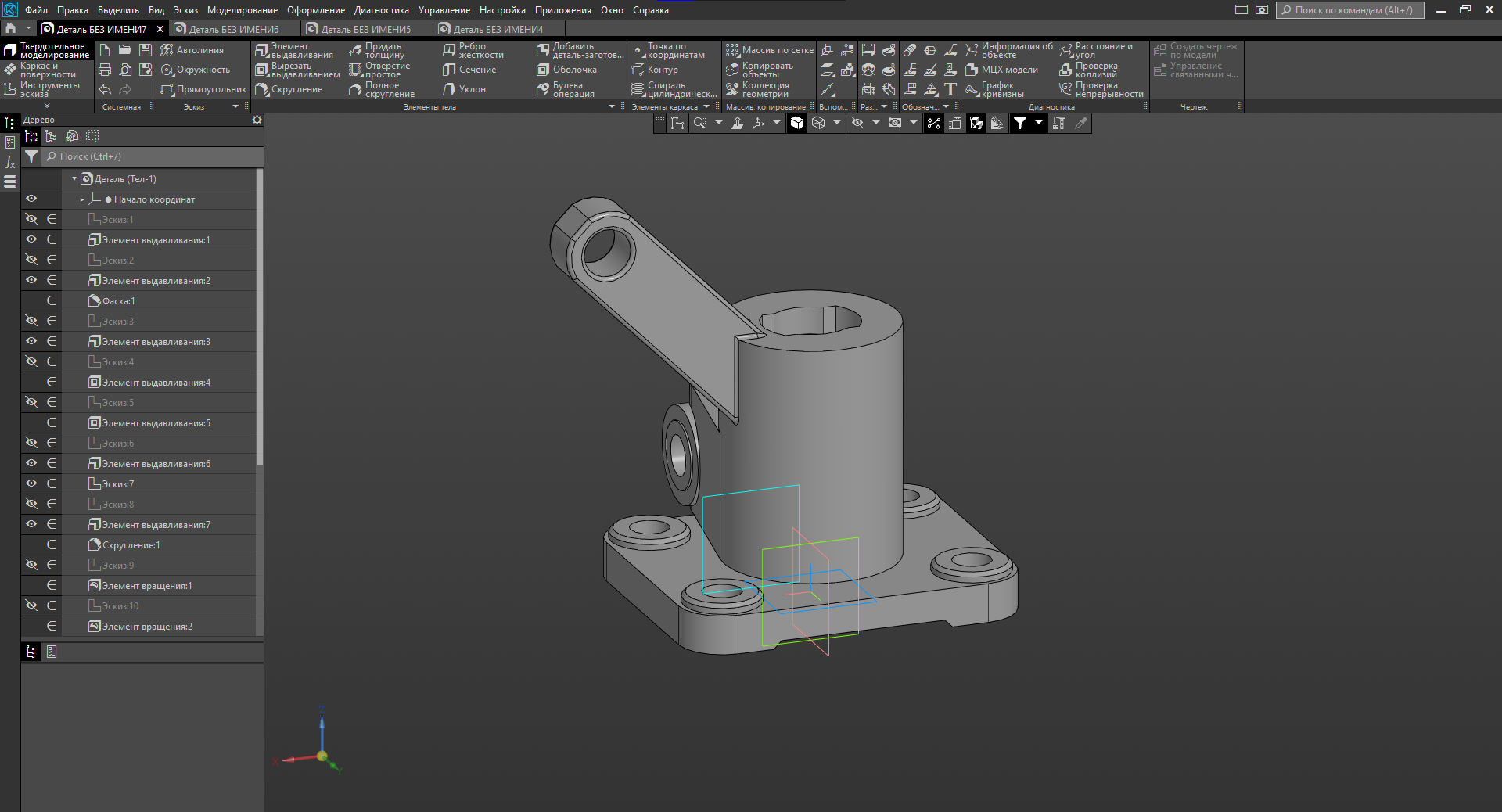


Рисунок 24 – Результат построения второго варианта детали Корпус.01.001

# 3. Сравнение подходов к автоматизации проектирования

# 3.1 Доступный функционал и интеграция в САПР-систему

Для начала, стоит рассмотреть функционал, который предоставляют оба инструмента для реализации шаблонной системы автоматизированного проектирования. Компас-макро записывает довольно обширный список операций, выполняемых в САПР системе, основные операции, запись которых выполнялась: базовая операция вращения, базовая операция выдавливания, базовая операция по сечениям, вырез в листовом теле, вырезать вращением,вырезать выдавливанием, вырезать кинематически, диаметральный размер, дуга, кинематическая операция, линейный размер, операция скругления кривых, операция соединения кривых, отрезок, плоскость под углом, поверхность вращения, поверхность выдавливания, поверхность по пласту точек, поверхность по сети точек, поверхность по сечениям, радиальный размер, ребро жесткости, сечение по эскизу, скругление, условное изображение резьбы, фаска, цилиндрическая спираль, эскиз. Применение этих операций позволяет создать все компоненты сборки, что и было сделано. Однако, при работе в окне сборки, Компас-макро не записывает операции загрузки детали из файла и позиционирования их при помощи сопряжений. Из этого следует, что для создания всего функционала, имеющегося в приложении, написанном на Visual C++, с использованием API, требуется вручную дорабатывать макросы, записанные при помощи Компас-макро. Для пользователей, обладающими знаниями о написаннии программ с использованием API это не создаст большой проблемы, ведь макросы изпользуют те же библиотеки, такие как kAPI5.tlb и kAPI7.tlb и ksKonstants3D.tlb. Однако, пользователи, обладающие только навыками работы с САПР, могут столкнуться с трудностями.

Написание приложений в отдельной среде разработки, в нашем случае Visual Studio 2019 имеет основное преимущество над Компас-Макро – это намного более обширный список поддерживаемых операций, что позволяет создавать более масштабные проекты.

Приложения, написанные на Visual C++, могут быть интегрированы в САПР Компас-3D. Для этого требуется, собрать проект в .rtw или .dll файл, в дальшнейшем его можно будет запускать из UI САПР-системы. На рисунке 25 изображен запуск проекта из раздела «Приложения».

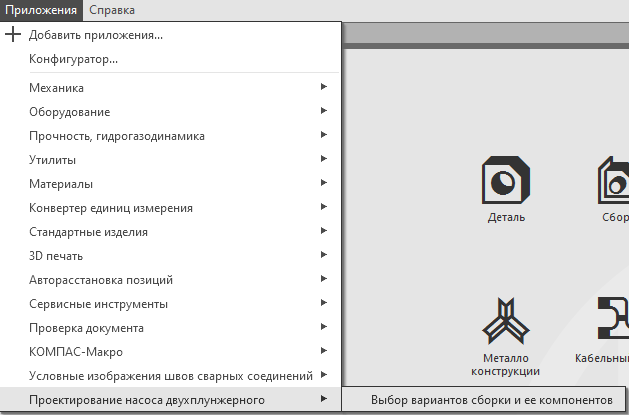


Рисунок 25 – Запуск проекта на Visual C++

Для Макросов доступен другой способ запуска, также переходим в раздел «Приложения», затем «Компас-макро» -> «Макросы». Перед нами открывается окно, из которого добавить записанный макрос, а затем запустить его. Начиная с Компас-3D V19, возможен запуск макросов через PyScripter с интерфейсом, написанным с использованием библиотеки tkinter для Python.

Программа, написанная с использованием Компас-макро представляет собой основной main.py файл, который объединяет в себе все макросы, записанные для каждого варианта детали, а также описание графического интерфейса. Запуск осуществлятся из другой среды разработки, под названием Visual Studio Code и для своей работы требует предварительно запущенного Компас-3D. Этот подход аналогичен написанию скриптов в PyScripter и требует промежуточного звена виде среды разработки. На рисунке 26 изображен запуск программы из VS Code.

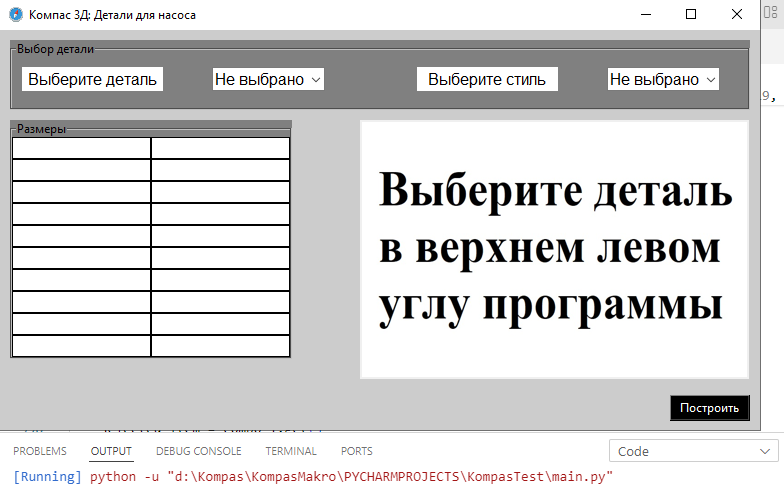


Рисунок 26 – Запуск проекта с применением Компас-макро из VS Code

В свою очередь, для запуска приложений, собранных в .rtw файл, достаточно двойного нажатия на него. Это вызовет запуск Компаса, а далее он будет доступен в разделе «Приложения».

# 3.2 Проблемы возникшие в ходе разработки программ

Компас-макро предоставляет достаточно удобный и простой спобос, реализовать построения отдельных деталей, однако, в ходе записи макросов, следует безошибочно строить геометрию эскизов. Редактирование эскизов при помощи усечения или продления линий, добавления скруглений, дуг может повлечь за собой неверные строки кода в макросах. Узнать об ошибке можно лишь после запуска готового макроса, это приводит к тому, что приходится перестраивать деталь до тех пор, пока точные свойства геометрии (длина, угол, диаметр, расположение) не будут запомнены, чтобы построить эскиз без использования ранее описанных операций. Некоторые из этих проблем можно решить путем редактирования макроса, путем добавления угла или смещения у определенного эскиза, однако избавится таким способом от лишних построений не представляется возможным. Если в ходе построения детали, возникают какие-либо сложные построения, достаточно проблематично отыскать строку в записанном макросе, отвечающую за определенную часть эскиза.

Для приложений, написанных на Visual C++ и имеющих интерфейс требуется предварительная настройка, описанная в разделе 1.1, это стало проблемой до нахождения примеров кода, содержащих интерфейс, созданный при помощи MFC-классов. Большим плюсом является то, на любом этапе написания API можно собрать проект и проверить не возникло ли в нем каких-либо ошибок. Но некоторых ошибок не удалось избежать, например, в ходе создания сборки был выбран вариант задания сопряжений по индексам граней, однако, стало известно о том, что при перестроении деталей их грани нумеруются иначе, только после того, как все сопряжения были расставлены. Аналогичная ошибка, возникла в ходе создания фасок и скруглений. Для них правильным решением было задавать грань по координатам точки, пересекающей ее.

# **3.3 Итоговое сравнение подходов к автоматизации в САПР Компас-3D**

В рамках разработки шаблонной системы автоматизированного проектирования было проведено сравнительное исследование двух подходов: использования API версии 5 Компас 3D для C++ и Компас-макро. Выбор наиболее подходящего подхода к автоматизации проектирования в Компас-3D (Компас-макро и приложения Visual C++) зависит от ряда факторов:

Отличительные черты автоматизации проектирования с использованием Компас-макро:

* Легкость использования: Запись макроса осуществляется путем фиксации действий пользователя в САПР-системе, не требуя навыков программирования;
* Быстрая разработка: Подходит для простых задач, таких как создание базовых элементов, выполнение повторяющихся операций;
* Ограниченный функционал: Не подходит для сложных задач, нестандартных операций, автоматизации больших сборок;
* Низкая производительность: Макросы могут работают медленнее, чем приложения C++, особенно при работе со сложными моделями;
* Невозможность интеграции с другими системами: Ограниченные возможности взаимодействия с внешними API;
* Подходит: Для пользователей без опыта программирования, которым требуется быстрая разработка простых решений;
* Быстрая разработка: Подходит для ситуаций, когда требуется быстрое решение;

Отличительные черты автоматизации проектирования путем создания приложений, написанных с использованием API на Visual C++:

* Расширенный функционал: Подходит для сложных задач, нестандартных операций, автоматизации больших сборок.
* Высокая производительность: C++ позволяет оптимизировать код для достижения максимальной производительности;
* Возможность интеграции с другими системами: Широкие возможности взаимодействия с внешними API;
* Масштабируемость: Подходит для долгосрочных проектов с перспективой расширения функциональности;
* Сложность разработки: требуется предварительная настройка для приложений, имеющих интерфейс;
* Для опытных программистов, которым требуется создавать сложные решения с высокой производительностью и возможностью интеграции с другими системами;
* Требует больше времени на разработку: Не подходит для ситуаций, когда требуется быстрое решение;
* Необходимость интеграции с другими системами: C++ предлагает больше возможностей для взаимодействия с внешними API.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были разработаны две программы, представляющие собой шаблонные системы автоматизированного проектирования изделия.

Для программного продукта на Visual C++ разработана библиотека для КОМПАС-3D с использованием API версии 5. На основе полученных знаний было реализовано построение геометрии компонентов изделия, выполнена сборка различных вариантов изделия с сопряжениями, а также созданы исполнения для ее составных частей, был разработан интерфейс приложения при помощи MFC-классов, который предоставляет удобный и понятный функционал для взаимодействия с САПР системой и позволяет проектировать изделие.

Для программного продукта, написанного при помощи Компас-макро был реализован интерфейс на python, с использованием библиотеки tkinter, который позволяет запускать макросы, записанные для каждого из вариантов компонентов сборки.

На основе опыта разработки двух шаблонных систем автоматизированного проектирования, был проведен анализ отличительных черт, преимуществ и недостатков, проблем, возникших в ходе разработки. Проведенный анализ двух подходов к автоматизации, позволит более тщательно подходит к созданию программных продуктов, нацеленных на проектирование изделий.