



Тихомиров В.А.

# Разработка приложений для UNIGRAPHICS на языке C

ПРОГРАММИРОВАНИЕ  
В NX  
1  
часть

**Владимир Тихомиров**

**РАЗРАБОТКА ПРИЛОЖЕНИЙ  
ДЛЯ UNIGRAPHICS  
НА ЯЗЫКЕ C**

**ПРОГРАММИРОВАНИЕ**

**Издательство  
ФГБОУ ВПО «КНАГТУ»  
2012**

УДК 681.3.06, 30.2-05

Тихомиров В.А. Разработка приложений для Unigraphics на языке С.  
– Издательство: ФГБОУ ВПО «КНАГТУ», 2012. – 462 с.

В учебном пособии рассматриваются вопросы разработки внутренних и внешних приложений для САД системы NX, выполненных программированием на языке С с использованием библиотек Open API. Разбираются алгоритмы программ, обеспечивающих выполнение типовых операций в САД системе NX и форматы используемых при этом функций. Приводится справочный материал, методика и примеры разработки форм (диалоговых окон) графического интерфейса для прикладных приложений, с использованием технологий, предоставляемых САД системой NX. Разбираются примеры реальных приложений для NX, используемые на производстве. Даются задания для закрепления разобранного материала. Предлагаются приемы построения лабораторных работ по изучению программирования в NX.

Учебное пособие предназначено для студентов направления 230100.68 «Информатика и вычислительная техника» (магистратура) по программе «Информационное и программное обеспечение САПР», а также для студентов других направлений, изучающих дисциплины, связанные с автоматизацией работ в САПР. Пособие также может быть использовано соответствующими инженерами, научными работниками и программистами при выполнении практических работ на производстве при создании приложений для САД системы NX.

*Ключевые слова:* автоматизация САПР, программирование в NX, функции Open API NX.

Рецензенты:

Доктор технических наук, профессор В.А. Комаров,  
Самарский государственный аэрокосмический университет;

Доктор технических наук, профессор С.Н. Падалко,  
Московский авиационный институт.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	7
1. Подготовка среды разработки программных модулей NX на базе библиотек Open API.....	10
1.1. Установка и настройка среды компиляции проектов NX.....	10
1.1.1. Подготовка среды разработки по варианту – 1. (Сначала MS VS, затем NX).....	11
1.1.2. Подготовка среды разработки по варианту – 2. (Сначала NX, затем MS VS).....	12
1.1.3. Подготовка среды разработки по варианту – 3. (Ручное создание проекта NX).....	13
1.2. Выполнение модулей DLL в среде NX.....	19
1.3. Шаблон внутренних прикладных модулей NX.....	19
1.4. Отладка внутренних прикладных модулей NX.....	25
1.4.1. Трассировочные сообщения.....	27
1.5. Функции Open API NX языка C.....	31
1.5.1. Порядок использования функций Open API (на примере построение точки и окружности).....	33
1.5.2. Источники информации по вопросам программирования для NX.....	36
1.6. Лицензионная подпись разработанных приложений.....	37
2. Моделирование кривых функциями OPEN API NX.....	41
2.1. Построение прямой.....	43
2.2. Построение конических кривых.....	44
2.3. Построение сплайнов.....	49
2.4. Сплайны в Open API NX.....	50
2.5. Матрицы в компьютерной графике Open API NX.....	66
2.6. Координатные функции NX Open API.....	79
3. Моделирование объектов и действий над ними функциями Open API NX.....	86
3.1. Цилиндр.....	86
3.2. Тело вращения.....	88
3.3. Удаление объектов.....	91
3.4. Копирование объектов.....	96
3.5. Копирование составного объекта.....	100
3.6. Анализ 3D тела в NX.....	104
3.7. Управление памятью в NX.....	115
3.7.1. Списки объектов.....	116
3.7.2. Выделение динамической памяти.....	117
3.7.3. Строковые массивы.....	118
4. Графический интерфейс приложений для NX.....	120

4.1. Организация интерфейса с помощью специализированных функций.....	120
4.1.1. Вывод текстовых сообщений на информационных линейках ....	120
4.1.2. Получение информации о координатах точек .....	121
4.1.3. Диалог выбора объектов заданного типа.....	122
4.1.4. Выбор эскизов, имеющихся в текущей сцене .....	125
4.1.5. Диалог построения плоскости. ....	126
4.1.6. Диалог построения вектора.....	129
4.1.7. Диалог построения точки (конструктор точек) .....	130
4.1.8. Функции работы с информационным окном .....	131
4.1.9. Окна сообщений .....	132
4.1.10. Диалог задания точки на экране визуальным позиционированием .....	136
4.1.11. Диалоги работы с именами файлов и рабочей сцены.....	141
4.1.12. «Старые» функции графического интерфейса .....	143
4.2. Визуальный конструктор диалоговых окон Open User Interface Styler .....	149
4.2.1. Атрибуты и методы формы диалога .....	156
4.2.2. Поля цифрового ввода и движки .....	162
4.2.3. Группа, кнопки, и элемент выбора цвета .....	168
4.2.4. Использование списков .....	173
4.2.5. Элементы управления с графическими полями.....	182
4.2.6. Вспомогательные элементы управления .....	190
4.2.6.1. Закладки и группы закладок .....	190
4.2.6.2. Прочие вспомогательные элементы управления .....	192
4.2.7. Выбор объектов на рабочей сцене при работе диалога .....	193
4.3 Локализация интерфейса .....	197
4.4 Сопровождение созданного диалога контекстной помощью .....	202
4.5. Новый инструмент создания диалогов - Блок Styler .....	204
4.6. Использование форм и диалогов операционной системы и третьих фирм.....	209
5. Инструменты запуска прикладных приложений .....	211
5.1. Технология MenuScript.....	211
5.1.1. Настройка каталогов Menuscript.....	211
5.1.2. Синтаксис файла Menuscript .....	213
5.2. Инструментальные линейки пользователя.....	219
5.3. Работа с инструментальными линейками из приложения.....	221
6. Виды представлений объектов в NX и функции работы с ними .....	223
6.1. Фасетное представление тел .....	223
6.2. Контурно-поверхностное представление (boundary representation) тел .....	238
7. Функции для работы со сборками .....	258

7.1. Сборка при позиционировании по абсолютным координатам .....	258
7.2. Анализ элементов сборки для автоматизированного позиционирования её компонентов .....	269
7.3. Использование сопряжений при программной сборке .....	277
8. Примеры программирования внутренних модулей NX .....	284
8.1. Работа со слоями .....	284
8.2. Разработка модуля поиска внутренних ребер тела .....	289
8.3. Модуль определения минимальных габаритов тела .....	305
8.4. Построение отрезка, перпендикулярного грани тела .....	315
8.4.1. Построение отрезка нормали к поверхности из точки на поверхности .....	315
8.4.2. Построение отрезка нормали к поверхности из точки вне поверхности .....	319
8.5. Построение отверстия на произвольной грани .....	320
8.6. Определение длины произвольной кривой .....	322
8.7. Автоматизированное построение кривой, соединяющей две точки на гранях тела .....	325
8.8. Редактирование геометрии и положения объекта сцены .....	332
8.9. Редактирование дуг и окружностей на модели под заданный радиус .....	334
8.10. Программное создание тел типа пружина .....	336
8.11. Программирование в модуле «ЧЕРЧЕНИЕ» (DRAFTING) .....	348
8.12. Работа с размерами и надписями .....	354
9. Разработка внешних (External) приложений NX .....	365
9.1. Внешние приложения NX консольного типа .....	365
9.2. Внешние приложения NX в стиле Win 32 API .....	371
10. Ответы на задачи упражнений для закрепления материала .....	379
10.1. Задача п. 1.5.1 (№ 1) .....	379
10.2. Задача п. 7.3 (№ 1) .....	380
10.3. Задача п. 7.3 (№ 2) .....	385
11. ПРИЛОЖЕНИЕ .....	400



## **ВВЕДЕНИЕ**

CAD (Computer Aided Design) системы к настоящему времени заняли прочную позицию в соответствующих нишах машиностроительного производства в нашей стране. Одной из лидирующих систем в этом направлении является NX.

NX прошла довольно долгий и сложный путь развития. Корни её лежат в системе Unigraphics, и базовый язык, на котором писалась система (еще тогда в среде UNIX), был язык C. Именно на нем были созданы первые библиотеки, предоставленные пользователям для разработки собственных приложений для этой системы, и до сих пор имеющие исключительно высокую функциональность. Именно через логику работы этих библиотек наиболее наглядно изучается внутренняя логика работы системы, структура и логика программной реализации в ней создаваемых моделей. Отдавая дань истории, данное пособие было названо «Разработка приложений для Unigraphics на языке C», хотя все приведенные в пособии примеры и методики полностью соответствуют, существующей сейчас, системе NX.

Программный пакет NX – система нового поколения, предназначенная для цифровой разработки изделий. Обладая самым широким в отрасли набором интегрированных и полностью взаимосвязанных CAD/CAM/CAE приложений, NX обеспечивает процессы разработки, инженерного анализа и подготовки производства.

Являясь законченным решением для цифрового создания изделия, NX предлагает интегрированную систему для выполнения задач проектирования, инженерного анализа, создания документации, оснастки и подготовки производства любой сложности для всех областей промышленности. К настоящему моменту в разных издательствах вышли полезные книги по работе в системе NX [3, 20], но, в основном, они посвящены конструкторскому инструментарию системы.

Инструменты системы NX обеспечивают выполнение большинства требований и запросов конструкторов и производственников, а для тех запросов, которые оказались не охваченными, система предоставляет универсальный механизм, позволяющий пользователю создавать собственные инструменты и приложения для решения своих запросов. Активное использование инструментария по разработке собственных приложений для CAD/CAM/CAE систем позволяет существенно повысить автоматизацию

работ в этих системах, снизить временные затраты на разработку и технологическое сопровождение производственных процессов и изделий. Однако по вопросам программирования в NX русскоязычной литературы, в открытом доступе, всё еще недостаточно.

Данное учебное пособие посвящено обучению начальным приемам использования библиотек и инструментария NX для создания собственных приложений прикладными программистами.

Для разработки собственных инструментов и приложений система NX предлагает четыре модуля [4]:

1. Модуль NX/Open API - обеспечивает прямой программный интерфейс к системе NX, позволяя пользователю создавать приложения на наиболее известных на сегодняшний день языках программирования Basic, C и Java. Модуль обеспечивает свободно расширяемую модель данных. Вы можете определить собственный объект на базе стандартных объектов NX. Этот объект будет изображаться, управляться и храниться в базе данных NX, как и любой стандартный объект. Вы можете написать приложение, которое будет работать как полностью интегрированная «внутренняя» функция NX или как самостоятельно выполняемая программа.
2. Модуль NX/Open++ - содержит объектно-ориентированный интерфейс к NX. Написанный на языке C++, интерфейс использует все преимущества объектно-ориентированного программирования: наследование свойств, инкапсуляцию и полиморфизм. Модуль обеспечивает полный доступ к иерархической модели классов, разрешая пользователю перегружать операции, выводить свои собственные классы и создавать целиком новые объекты в NX. NX/Open++ полностью совместим с модулем NX/Open API и обеспечивает легкое решение сложных программных задач.
3. Модуль NX/Knowledge Fusion - содержит среду для программирования на языке инженерных знаний (Knowledge Based Engineering, КВЕ). Конструкторы и дизайнеры могут заниматься непосредственно проектированием изделия, не занимаясь трансляцией данных между системами КВЕ и MCAD. Язык КВЕ является промышленным стандартом де-факто. Многие из ведущих производителей в аэрокосмической, автомобильной промышленности и общем машиностроении используют язык инженерных знаний КВЕ в целях: уменьшения времени изготовления изделия вследствие автоматизации повторяющихся процессов и наложении определенных правил во время проектирования и улучшению качества проектирования, благодаря применению инженерных знаний, заложенных в систему.



4. Модуль NX/Open GRIP - содержит простой язык для автоматизации простых задач в CAD/CAM/CAE. Пользователь может создать приложения для автоматизации создания геометрических моделей, программ для станков ЧПУ, черчения. С помощью GRIP вы можете полностью автоматизировать процесс создания модели и ее чертежа.

В данном пособии (которое следует считать первой частью общего проекта «NX программирование») рассматривается только первый модуль - программирование прикладных программ в среде NX на языке C с использованием библиотек Open API NX. Для тех читателей, кто предпочитает языки Basic или Java (рассматриваемый модуль поддерживает и эти языки) данное пособие также может быть полезным, поскольку алгоритмы и идеология создания прикладных приложений для NX одинакова для этих трех языков.

С другой стороны – автор не ставил себе задачу сделать справочник по всем функциям Open API NX. Задача учебного пособия – донести до читателя «дух» использования функций Open API NX в собственных программных разработках. Поняв общую идею программирования для NX, дополнительную справочную информацию по функциям Open API читатель всегда сможет найти самостоятельно в справочной системе NX в тех файлах, которые описаны в пособии.

Пособие ориентировано на читателей, владеющих языком C и C++, и работающих в среде CAD NX по созданию 3D моделей и 2D чертежей.

Все вопросы и замечания по поводу данной книги вы можете направить автору по адресу:

Россия, 631013, Комсомольск-на-Амуре, пр.Ленина, д.27, ФГБОУ ВПО «КНАГТУ», каф. МОПЭВМ

Тел.: [\(4212\) 54-68-54](tel:(4212)54-68-54)

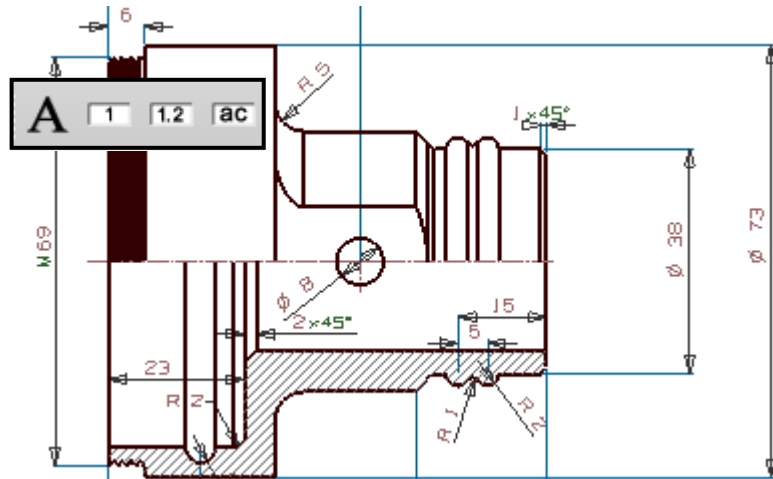
Факс: [\(4212\) 53-61-50](tel:(4212)53-61-50)

URL: <http://www.knastu.ru>

e-mail: [kmopevm@knastu.ru](mailto:kmopevm@knastu.ru)

# 1

## Подготовка среды разработки программных модулей NX на базе библиотек Open API



Для доступа к трехмерной модели в среде NX и построения различных автоматизированных систем, управляющих, как самими моделями, так и операциями их изменения, предусмотрен механизм разработки прикладных программных модулей на языке C с использованием библиотек встроенных функций NX Open API. Прикладной модуль может быть внешнего типа и выполнен, как отдельное приложение в формате EXE файла, или внутреннего типа – DLL файл, который запускается из запущенной CAD системы NX. Инструментарием для создания таких модулей рекомендуется использовать MS Visual Studio. В учебном процессе, при изучении программирования в среде NX, можно ограничиться бесплатным пакетом MS Visual Studio Express, который позволяет реализовать практически все технологии, описанные в данном пособии.

### 1.1. Установка и настройка среды компиляции проектов NX

В повседневной практике возможны три пути установки и настройки среды Microsoft Visual Studio для компиляции проектов внутренних и внешних модулей для NX:

1. настройка, когда CAD система NX устанавливается после того, как установлена система MS Visual Studio;
2. настройка, когда MS Visual Studio ставится после установки CAD системы NX;
3. ручная настройка проекта, в случае, если ни первый, ни второй путь не позволил получить требуемого результата.

Штатным (соответствующим рекомендациям разработчиков NX) следует считать первый вариант. При нем вся установка и настройка выполняется в автоматическом режиме.

### 1.1.1. Подготовка среды разработки по варианту – 1. (Сначала MS VS, затем NX)

Следует установить (или убедиться, что на компьютере уже установлена) MS Visual Studio. Соотношение версий CAD системы NX и MS Visual Studio для правильного их автоматического интегрирования представлено в таблице 2.1.

MS Visual Studio 2010 не может быть автоматически интегрирована с указанными в таблице версиями NX, но это не мешает ее ручной настройке по варианту 3 (см. ниже).

Процесс установки MS Visual Studio не входит в рамки рассмотрения в данном пособии, и если он Вам не знаком или вызывает затруднения, то обратитесь к соответствующей литературе [19]. Мы же будем считать, что на текущий момент MS Visual Studio нужной версии у нас на компьютере уже установлена.

Таблица 1.1

Соответствие версий систем NX и MS Visual Studio

№№	Версия CAD системы NX	Версия MS Visual Studio
1	NX v.4.xx , NX v.5.xx	MS Visual Studio 2003
2	NX v.6.xx	MS Visual Studio 2005
3	NX v.7.0	MS Visual Studio 2008
4	NX v.7.5	MS Visual Studio 2008, SP1

Теперь установим сам NX. При наличии лицензионного программного продукта и документации по его установке это не вызывает каких либо затруднений. Первая и последняя экранные формы, сопровождающие успешную установку CAD системы, представлены на рисунке 1.1.

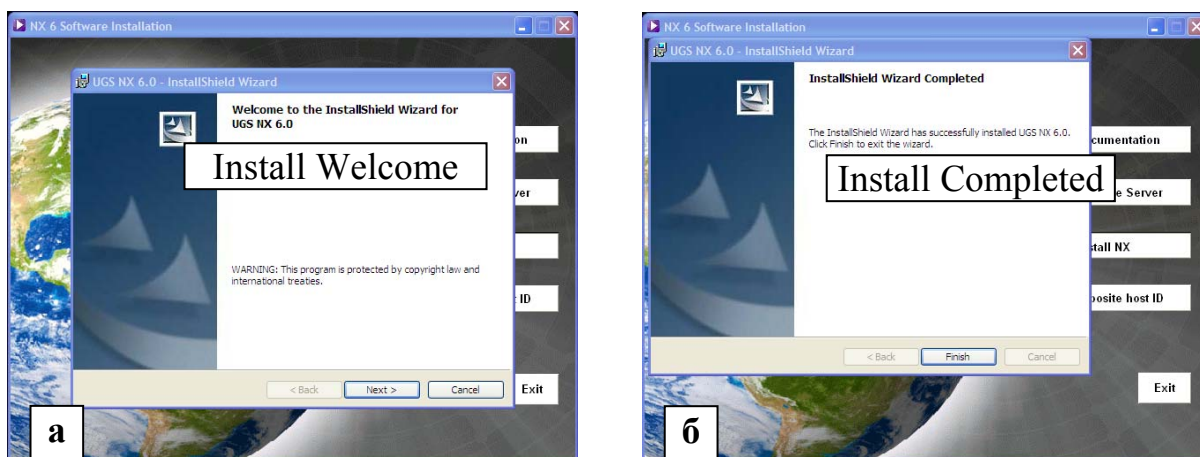
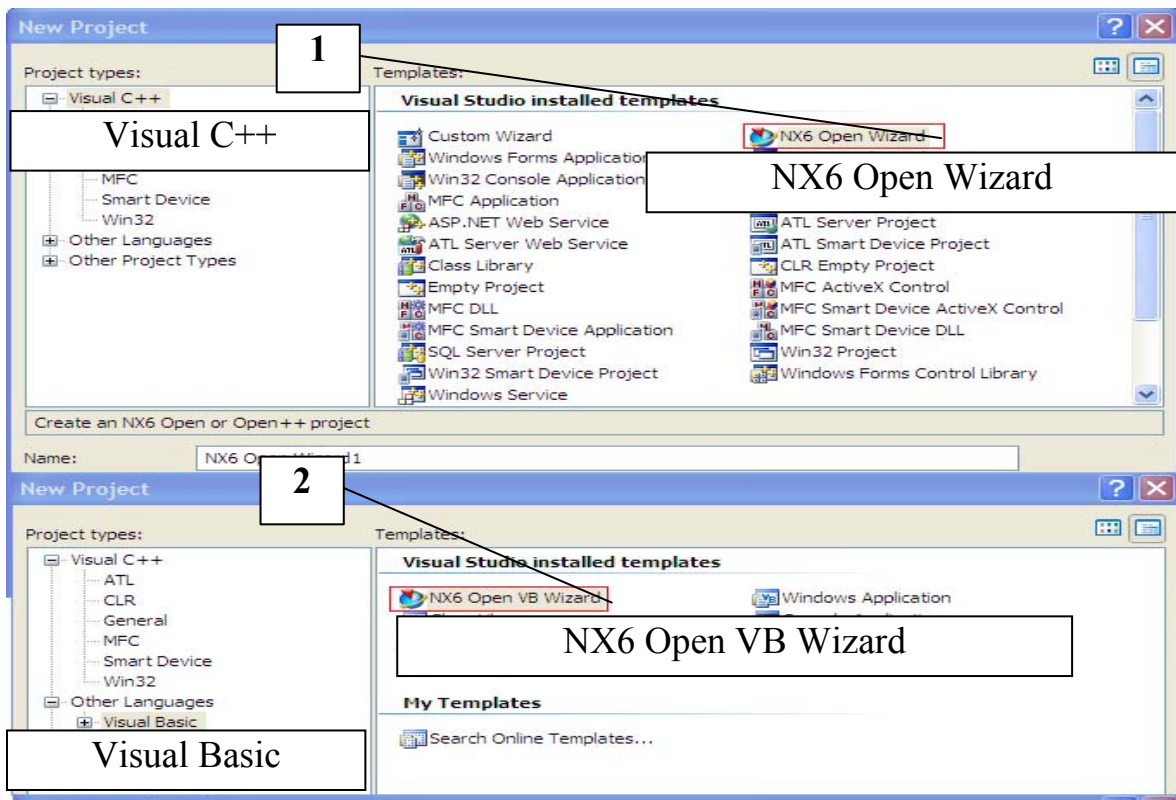


Рис. 1.1 Начало установки менеджера лицензий системы NX (а), завершение установки NX (б)



*Рис. 1.2 Появление шаблона проекта приложения NX 6.0 в окнах стандартных шаблонов MS Visual Studio*

Во время установки происходит автоматическая интеграция пакета NX в среду разработки MS VS (если версии пакетов соответствуют табл. 1.1) и если после завершения установки UG открыть Visual Studio (File -> New-> Project), можно увидеть в окне шаблонов проектов вновь добавленные специальные шаблоны для создания приложений NX. Новые шаблоны доступны вместе со стандартными - на вкладках для различных языков программирования: Visual C++ , Visual C#, Visual Basic (1,2 рис. 1.2).

### **1.1.2. Подготовка среды разработки по варианту – 2. (Сначала NX, затем MS VS)**

Если по каким-то причинам правильный вариант (вариант-1) установки пакетов провести не удалось, и пакеты установлены в обратном порядке, то автоматической интеграции NX в Visual Studio не происходит и шаблоны проектов NX в соответствующих окнах (рис.1.2) не появляются.

Чтобы вручную установить шаблоны проектов NX в Visual Studio необходимо скопировать содержимое папки X:\Program Files\UGS\NX xx.xx\UGOPEN\vs\_files в папку X:\Program Files\Microsoft Visual Studio xx (рис. 1.3), где X: – буква дисководы с установленным пакетом NX, а xx – версии пакетов NX и MS Visual Studio (с путями установки возможны варианты...).

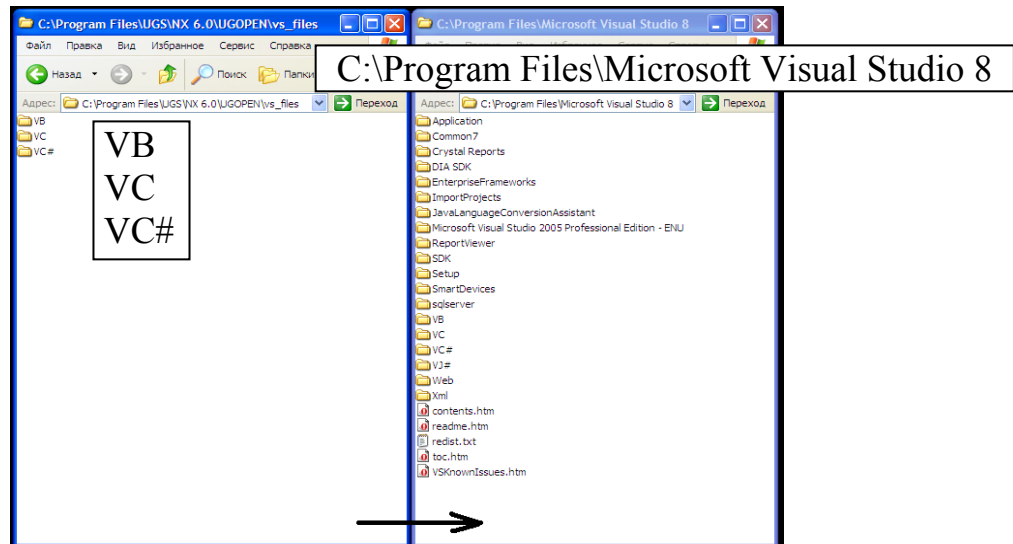


Рис. 1.3 Копирование папок каталога ... \UGOPEN\vs\_files для UG NX 6.0 в папку ... \ Program Files\Microsoft Visual Studio 8 для MS VS 2005

После копирования указанных папок в окне шаблонов проектов Visual Studio появятся шаблоны для создания приложений NX. Следует отметить, что таким же образом можно установить и шаблоны для Unigraphics 5.0 (рис. 1.4).

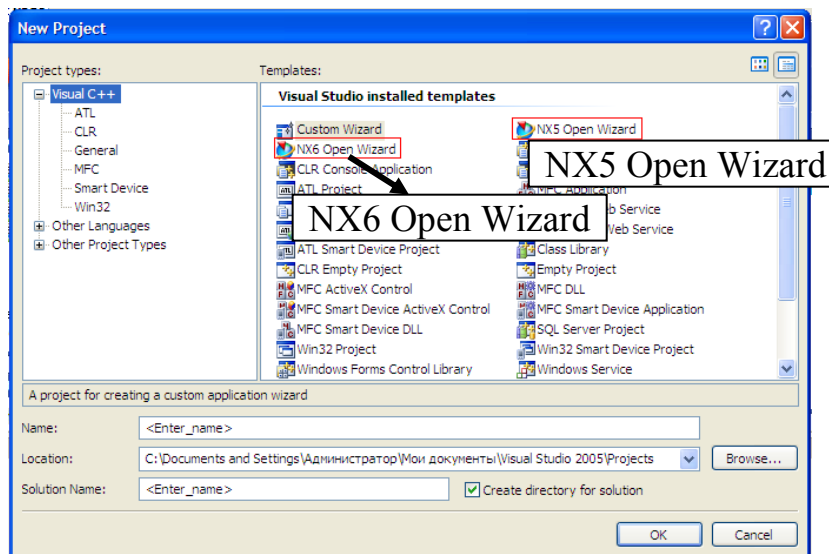


Рис. 1.4 Появление шаблонов NX 5.0 и NX 6.0 в Visual Studio 2005.

### 1.1.3. Подготовка среды разработки по варианту – 3. (Ручное создание проекта NX)

Возможна ситуация, когда необходимо создать проект NX в среде, не воспринимающей готовые шаблоны проектов, находящиеся в каталоге ... \UGOPEN\vs\_files. Например, Visual Studio 2008 шаблоны версий NX 4.xx, 5.xx, 6.xx «не понимает».

В таком случае требуется ручная настройка проекта разрабатываемого модуля NX.

Откроем Visual Studio 2008 и выберем «Создать новый проект» (File->New->Project) (рис. 1.5).

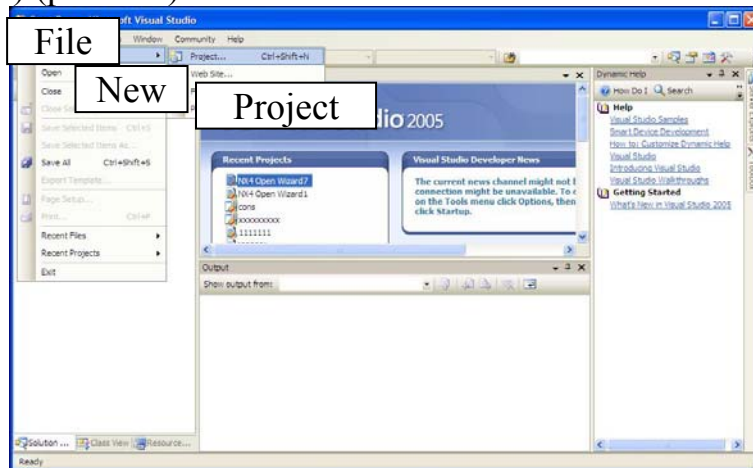


Рис. 1.5 Выбор нового проекта

Следует выбрать язык программирования (в нашем случае C++), для чего в левом окне Project types выберем Visual C++. Затем следует выбрать тип проекта - в нашем случае Win32 Console Application, и дать название проекту (в примере - UG) (рис.1.6).

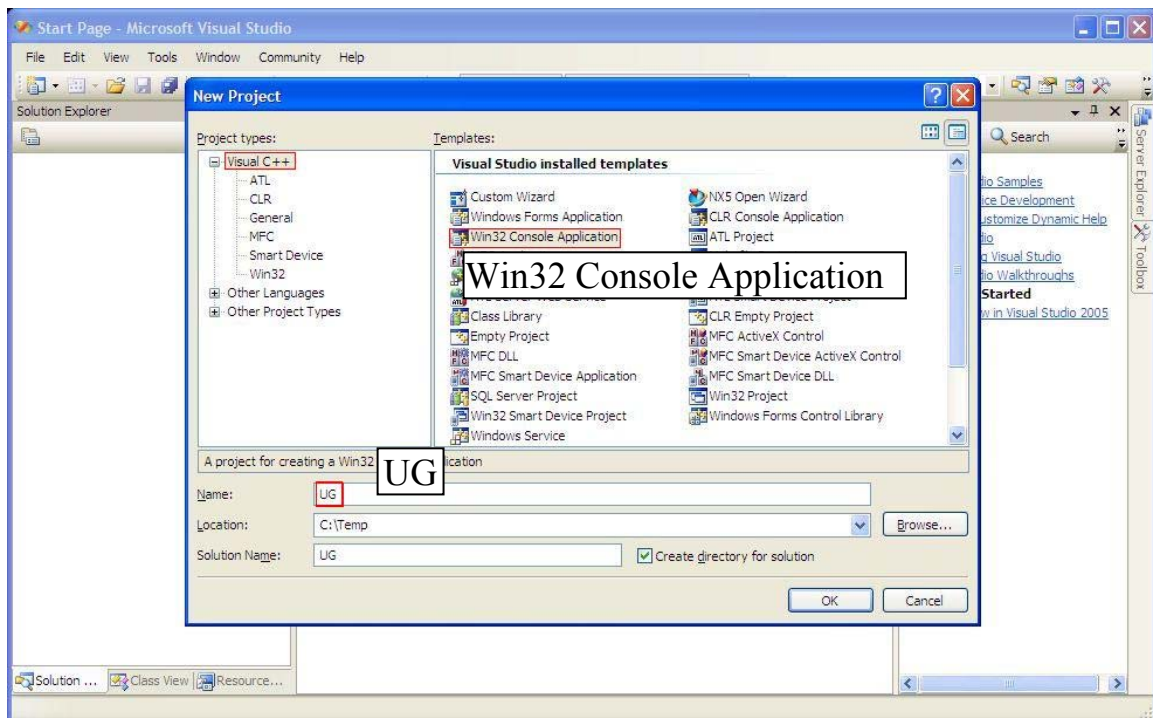


Рис. 1.6 Выбор языка и типа проекта.

В опциях создания проекта Application Settings (рис. 1.7) следует выбрать тип DLL (если планируется создание внутреннего модуля UG) или Console application (если планируется создание внешнего модуля UG). В дополнительных опциях следует поставить галочку «пустой проект» (Empty project). Нажать кнопку Finish. Создастся новый проект.

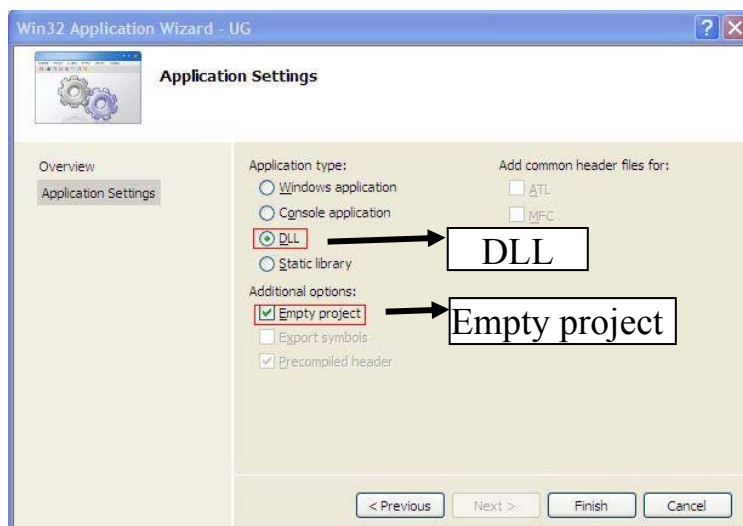


Рис. 1.7 Установка начальных свойств проекта приложения UG

Теперь следует настроить свойства проекта. Чтобы проект правильно скомпилировался в исполняемый файл NX, необходимо прописать место нахождения подключаемых библиотек и некоторые зависимости компонентов. Для настройки этих свойств необходимо в окне Solution Explorer нажать правой кнопкой мыши на имени проекта и выбрать пункт Свойства (Properties) (рис. 1.8).

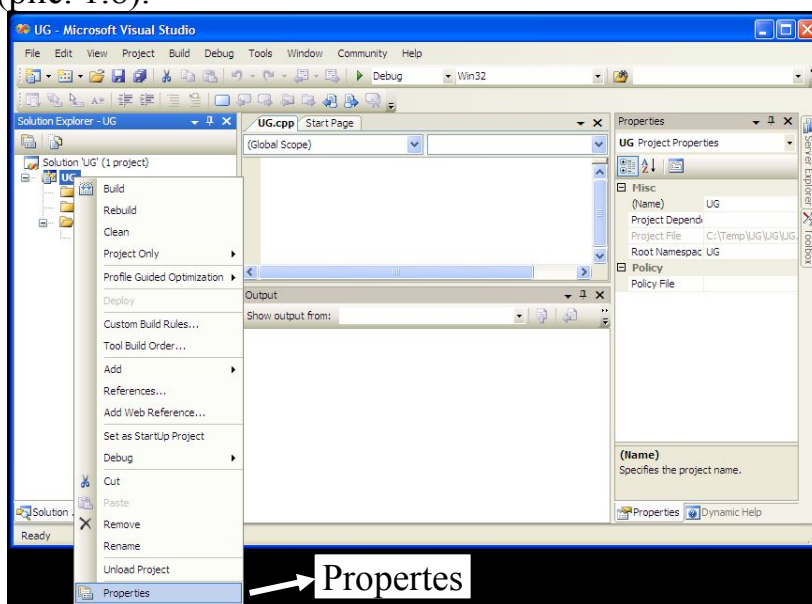


Рис. 1.8 Включение окон свойств проекта

Из вкладки Configuration Properties (Свойства конфигурации) выбрать во вкладке C/C++ опцию General (Общие) и к строке **Additional Include Directories** (Дополнительные каталоги включения) (рис. 1.9) с помощью кнопки 1 добавить местонахождение папок **ugoren** и **ugorenpp** которые находятся в каталоге NX (в примере обе папки находятся в каталоге C:\Program Files\UGS\NX 5.0). Данная строка подключает папки

**ugopen** и **ugopenpp** к проекту NX (для программирования только на языке C достаточно пути к одной папке **ugopen**).

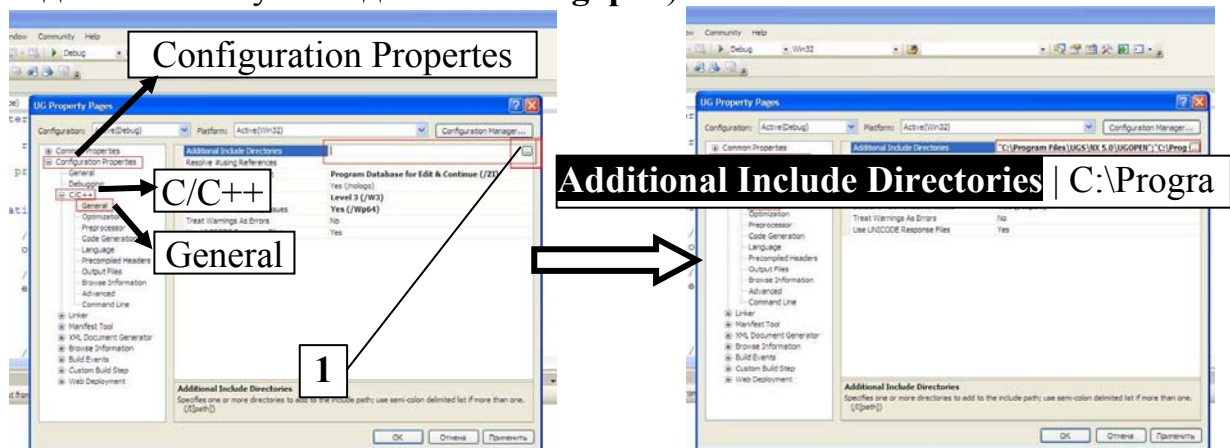


Рис. 1.9 Заполнение строки опции *Additional Include Directories*

Из вкладки **Configuration Properties** (Свойства конфигурации) выбрать вкладку **Linker** (Компоновщик) выбрать опцию **General** (Общие) и добавить к строке **Additional Library Directories** (Дополнительные каталоги библиотек) (рис. 1.10) путь к папке **ugopen**, которая находится в каталоге NX (на примере папка находится в каталоге C:\Program Files\UGS\NX 5.0). Это необходимо для подключения к проекту основной папки с библиотеками **ugopen**.

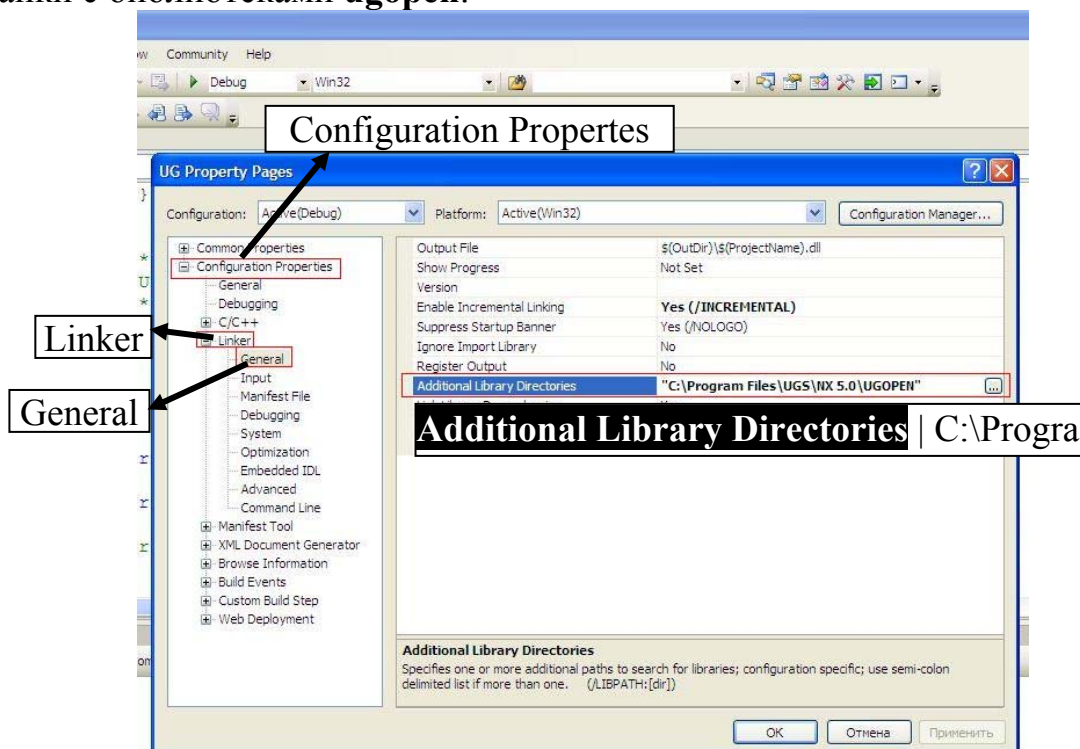


Рис. 1.10 Добавление опции к строке *Additional Library Directories*

Из вкладки **Configuration Properties** (Свойства конфигурации) выбрать вкладку **Linker** (Компоновщик) опцию **Input** (Ввод) (рис. 1.11) и да-



лее - прописать в строке **Additional Dependencies (Дополнительные зависимости)**, через пробел, следующее библиотеки: **libufun.lib libnxopencpp.lib libugopenint.lib libnxopenuicpp.lib libopenpp.lib libopenintpp.lib libvmathpp.lib**

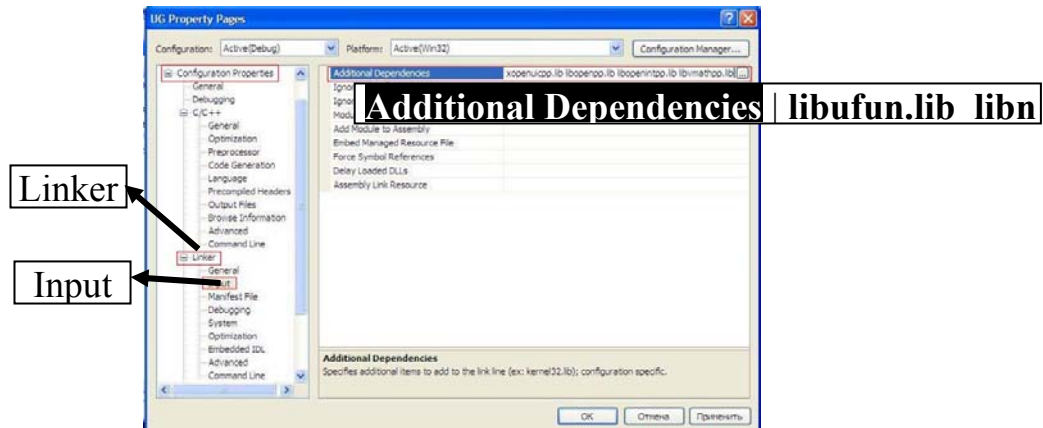


Рис. 1.11 Добавление опции к строке *Additional Dependencies (Дополнительные зависимости)*

Из вкладки **Configuration Properties(Свойства конфигурации)** выбрать вкладку **Linker (Компоновщик)** опцию **System (Система)** в строке **SubSystem (Подсистема)** выбрать **Console (/SUBSYSTEM:CONSOLE)** (рис. 1.12).

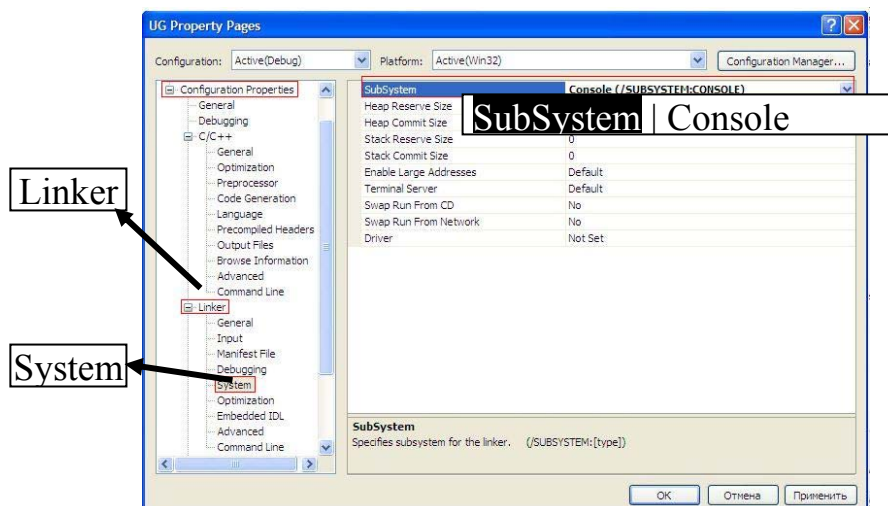


Рис. 1.12 Выбор опции в строке *SubSystem (Подсистема)*

В простейшем случае можно не выполнять все эти многочисленные настройки, а все необходимые для компиляции файлы (и описательные и библиотечные) включить прямо в проект создаваемого приложения (рис. 1.13).

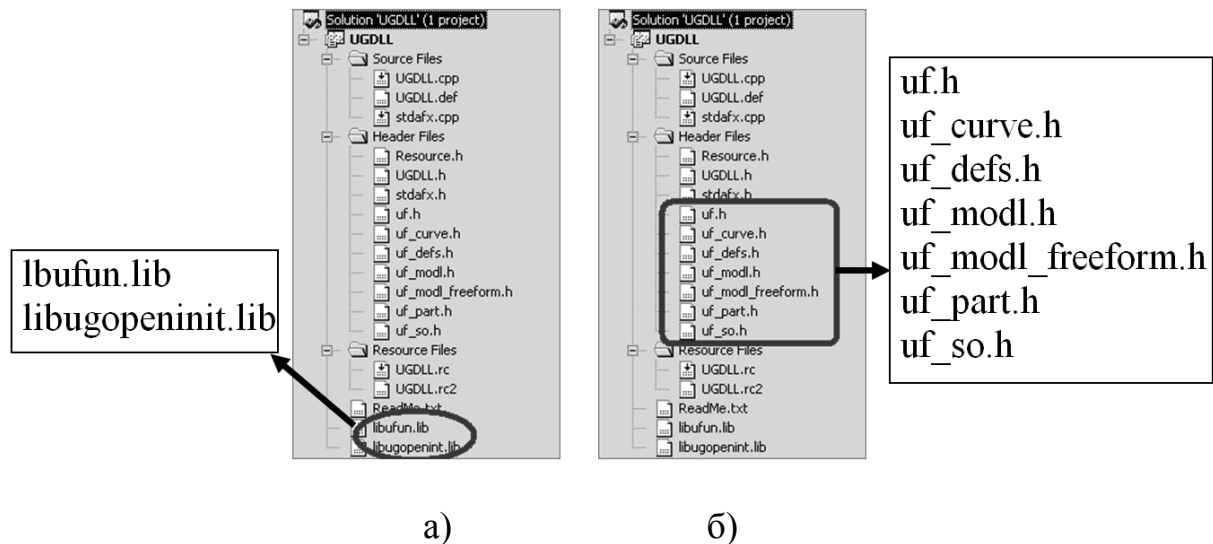


Рис. 1.13 Список библиотек (а) и включаемых файлов (б) для проекта

При этом, например, для программирования на чистом языке C, достаточно библиотечных файлов (рис. 1.13 а), указанных в таблице 1.1.

Таблица 1.2

Библиотеки, включаемые в проект при разработке приложений UG/Open

Тип приложения	Включаемые библиотеки
Внутреннее приложение	libugopeninit.lib, libufun.lib
Внешнее приложение	libufun.lib

После установки всех параметров можно вставить эталонный код из файла примеров, прилагаемых к системе NX, в файл .cpp проекта и компилировать данный проект. Если все настройки были прописаны правильно, компилятор не должен выдавать сообщения об ошибке (рис. 1.14).

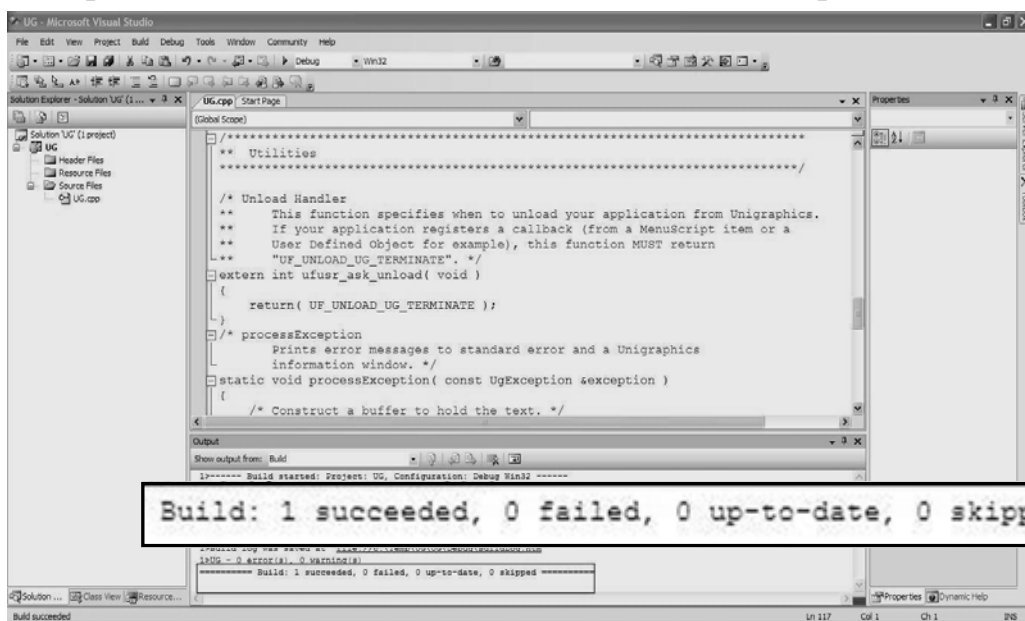


Рис. 1.14 Пример успешной компиляции созданного вручную DLL проекта модуля NX

## 1.2. Выполнение модулей DLL в среде NX.

После успешной компиляции любого приложения созданного с помощью шаблона либо вручную, в папке с приложением (или в папке Debug в зависимости от настроек) появится dll (или exe) файл.

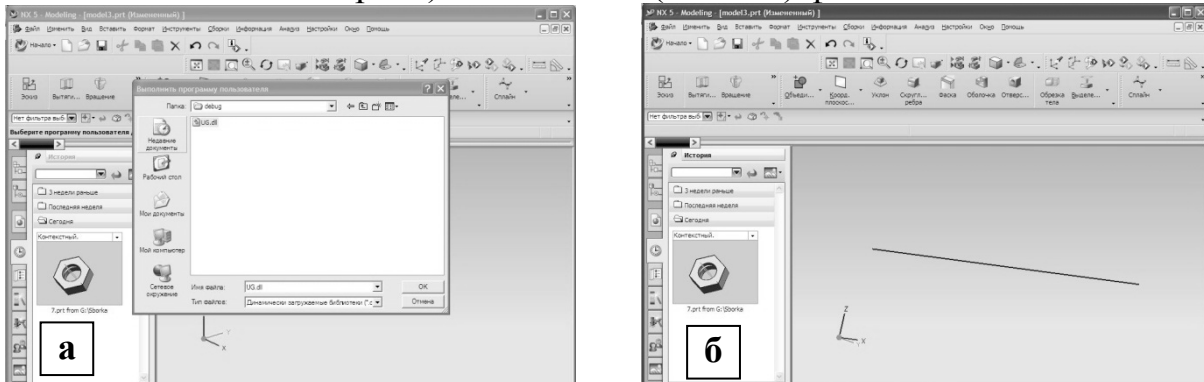


Рис. 1.15 Выбор DLL модуля для запуска в среде NX (а) и результат выполнения модуля – прочерчивание линии в рабочем пространстве (б)

DLL можно запустить в среде NX. Для этого нужно открыть NX (если надо - создать новую модель, или открыть имеющуюся) и нажать Ctrl+U откроется окно диалога выбора пользовательской программы, в котором следует указать созданный dll файл (рис.1.15 а). После чего сразу происходит загрузка и выполнение выбранного модуля (рис. 1.15 б), в примере – рисующего прямую линию на рабочей сцене (другие методы запуска внутренних приложений NX будут рассмотрены далее).

## 1.3. Шаблон внутренних прикладных модулей NX

Внутренние (Internal) прикладные модули системы NX, разрабатываемые на языке C, создаются в формате DLL библиотек и запускаются из работающей NX системы. Таким образом, внутренние прикладные модули выполняются в адресном пространстве процесса системы NX, что облегчает доступ из модуля к объектам NX, ускоряет операции их создания и редактирования.

Стандартной точкой входа в прикладную DLL NX является функция `ufusr()`. Рабочий шаблон этой функции представлен в листинге 1.1

### Листинг 1.1

---

```
/* Минимальный шаблон внутреннего модуля NX на базе Open C
```

```
входные параметры:
```

```
param – если эта функция будет вызвана из menuscrypt (см.далее),
то сюда передается название меню, в других случаях этот
параметр не используется;
```

```
parm_len – длина строки параметров
```

```
выходные параметры:
```

```
retcod – возвращаемый код прикладного модуля */
```

```
#include <uf.h> // Используемые файлы описаний
void ufusr(char *param, int *retcod, int parm_len)
{ UF_initialize(); //Инициализация библи. функций Open API
  ... // здесь должно быть тело прикладного модуля
  UF_terminate(); //Закрывание библиотек функций Open API
}
```

В шаблоне представлена одна, главная, точка входа в прикладной модуль – `ufusr()`. Всего же в NX предусмотрено 39 различных точек входа в процедуры исполняемого модуля, которые система NX вызывает в различные моменты своей работы. Информация о трех, наиболее часто используемых точках входа представлена в таблице 1.3. Наименование и назначение остальных точек входа представлены в таблице П6 приложения. Порядок использования первой точки таблицы 1.3 описан в листинге 1.1.

Таблица 1.3

## Основные точки входа во внутренних приложениях системы NX

Имя точки входа	Момент вызова	Назначение точки входа
<code>ufusr</code>	Запуск модуля	Запустить процесс
<code>ufusr_ask_unload</code>	Сразу после запуска модуля	Принять код завершения процесса
<code>ufusr_cleanup</code>	При завершении работы модуля	Выполнить операции по очистке и освобождению динамических переменных модуля

Вторая точка входа имеет сигнатуру:

```
extern int ufusr_ask_unload(void)
{ return (<код выхода из модуля>); }
```

и обычно содержит только код возврата, определяющий правила завершения работы прикладного модуля. Возможные коды возврата представлены в таблице 1.4.

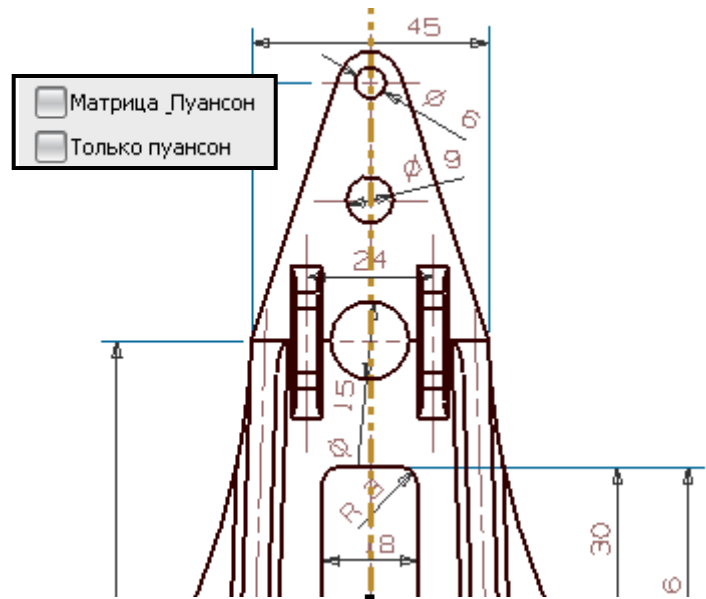
Таблица 1.4

## Коды возврата из внутренних приложений NX

Код возврата	Описание
<code>UF_UNLOAD_IMMEDIATELY</code>	Обеспечивает выгрузку DLL библиотеки модуля из памяти сразу после завершения работы модуля.
<code>UF_UNLOAD_SEL_DIALOG</code>	Обеспечивает возврат управления окну диалога, вызвавшему данный модуль.
<code>UF_UNLOAD_UG_TERMINATE</code>	Обеспечивает выгрузку DLL модуля вместе с завершением работы системы NX.

# 2

## Моделирование кривых функциями OPEN API NX



В CAD системах для моделирования контуров и ребер 3D тел, построения путей, траекторий перемещения сечений и т.п. операций в основном (кроме прямых дуг и окружностей)) используются следующие кривые [15].

**Spline** - сплайн. Кривая, четвертая производная которой равна нулю. Широко распространенный формат представления данных. Кривизна контролируется разбросом контрольных точек. Для построения используются различные типы, обычно, кубических кривых.

**Bezier curve** - кривая Безье. Гладкая кривая, состоящая из серий по четыре контрольные точки, которые в разной мере определяют ее направление. Совершенно необязательно условие прохождения кривой через все контрольные точки. Две точки определяют ее направление, а две другие являются конечными точками. Уравнение сплайна для кривой Безье в каноническом виде выглядит следующим образом.

$$\begin{cases} X(t) = a_x t^3 + b_x t^2 + c_x t + x_0 \\ Y(t) = a_y t^3 + b_y t^2 + c_y t + y_0 \end{cases}$$

где:  $t$  – параметр сплайна, переменная принимающая значения от 0 (начало кривой) до 1 (конец кривой);

$a_x, b_x, c_x, a_y, b_y, c_y$  - коэффициенты уравнения. Определяются из координат четырех точек, которые определяют направление кривой;  
 $x_0, y_0$  - координаты начальной точки сплайна.

$$\begin{cases} a_x = x_3 - x_0 - b_x - c_x \\ b_x = 3 \cdot (x_2 - x_1) - c_x \\ c_x = 3 \cdot (x_1 - x_0) \\ a_y = y_3 - y_0 - b_y - c_y \\ b_y = 3 \cdot (y_2 - y_1) - c_y \\ c_y = 3 \cdot (y_1 - y_0) \end{cases}$$

где:  $X_3, Y_3$  - координаты конечной точки сплайна;

$X_1, Y_1$  - координаты первой управляющей точки сплайна;

$X_2, Y_2$  - координаты второй управляющей точки сплайна.

**B-spline** (Bezier-spline). Один из основных способов, используемых в САД системах для математического представления гладких кривых. Кривая формируется по отношению к 3D-полилинии (т.е. ломаной линии). B-spline всегда начинается от первой контрольной точки и заканчивается в последней, всегда касается этой полилинии в этих точках, хотя в целом не проходит через другие контрольные точки (рис.2.1). Как уже говорилось, полиномиальная кривая, задается набором определяющих точек и представляет уравнение порядка на одну степень меньше количества учитываемых точек. Кривые B-spline записываются в памяти компьютера в виде математических формул, поэтому геометрия, полученная с помощью этих кривых, занимает малый объем памяти.



Рис. 2.1 Формирование B-сплайна по контрольным точкам

**C-spline** - сплайн, который образуется путем прохождения через все контрольные точки (рис. 2.2). Совпадение с контрольными точками более явное, чем у Bezier кривых, так как эти точки задаются непосредственно, а не через тангенциальные полилинии.



Рис. 2.2 Формирование C-сплайна по контрольным точкам

Рассмотрим порядок программного построения некоторых из перечисленных типов кривых в системе NX. Но начнем с самой простой линии – прямой.

## 2.1. Построение прямой

Для построения прямой достаточно иметь информацию о координатах двух точек пространства (рис. 2.3).

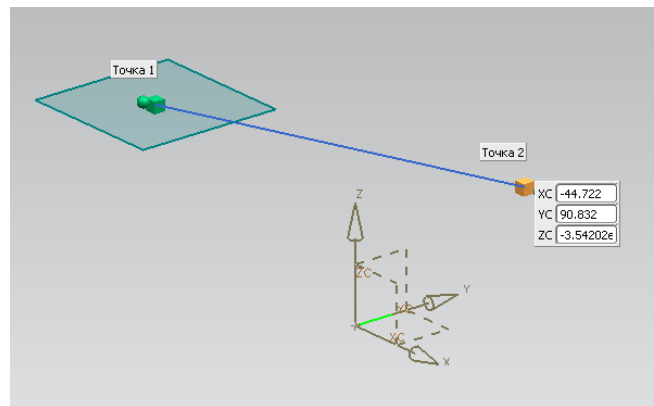


Рис. 2.3 Построение прямой

Функция для построения прямой линии в NX для языка C имеет вид:

```
int UF_CURVE_create_line(
    UF_CURVE_line_p_t line_coords, tag_t *line).
```

Первый параметр функции – структура, содержащая координаты точек концов прямой. Структура состоит из двух массивов по три элемента двойной точности (по одному массиву на каждую точку):

```
struct UF_CURVE_line_s {
    double start_point[3]; //координаты начальной точки
    double end_point [3]; //координаты конечной точки
};
```

Пример модуля для создания прямой линии с помощью вышеуказанной функции представлен в листинге 2.1.

Листинг 2.1

```

/*****
Пример модуля, строящего прямую линию в 3D пространстве сцены NX
*****/
#include <uf.h>
#include <uf_curve.h>

void ufusr(char *param, int *retcode, int paramLen)
{
    tag_t entid = 0; // Идентификатор объекта «линия»
    UF_CURVE_line_t line_coords; /* структура концевых точек
линии */

```

```

    if (UF_initialize()) return;

//заполнение координат конечных точек линии
    line_coords.start_point[0]=0.; // X1
    line_coords.start_point[1]=0.; // Y1
    line_coords.start_point[2]=0.; // Z1

    line_coords.end_point[0]=100.; // X2
    line_coords.end_point[1]=100.; // Y2
    line_coords.end_point[2]=100.; // Z2
// построение линии
    UF_CURVE_create_line (&line_coords, &entid);
UF_terminate();
}
//процедура выхода из приложения
int ufusr_ask_unload(void)
{ return (UF_UNLOAD_IMMEDIATELY); }

```

## 2.2. Построение конических кривых

Коническое сечение (или коника) в NX есть пересечение плоскости с круговым конусом. Существует три главных типа конических сечений: эллипс, парабола и гипербола (кроме того, существуют вырожденные сечения: точка, прямая и пара прямых. Окружность можно рассматривать как частный случай эллипса).

Конические сечения могут быть получены как пересечение плоскости с двусторонним конусом

$$\alpha^2 z^2 = x^2 + y^2 \text{ (в декартовой системе координат),}$$

где

$$\alpha = \operatorname{tg} \theta,$$

$\theta$  — угол между образующей конуса и его осью.

Если плоскость проходит через начало координат, то получается вырожденное сечение. В невырожденном случае, если секущая плоскость пересекает все образующие конуса в точках одной его полости, получаем эллипс, если секущая плоскость параллельна одной из касательных плоскостей конуса, получаем параболу и если секущая плоскость пересекает обе полости конуса, получаем гиперболу.

Уравнение кругового конуса квадратично, поэтому все конические сечения являются квадриками, также все квадрики плоскости являются коническими сечениями (хотя две параллельные прямые образуют вырожден-



денную quadriку, которая не может быть получена как сечение конуса, но всё же обычно считается «вырожденным коническим сечением»).

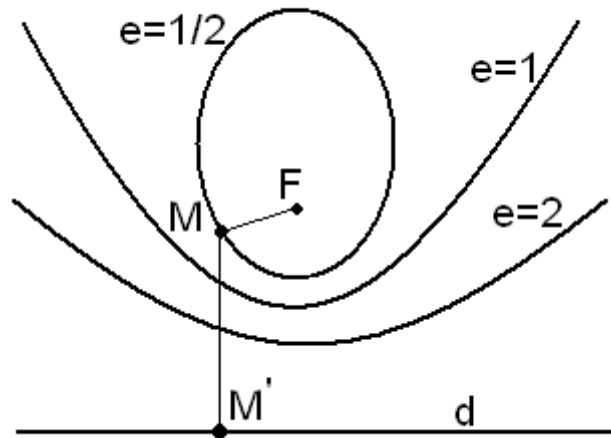


Рис. 2.4 Эллипс ( $e=1/2$ ), парабола ( $e=1$ ) и гипербола ( $e=2$ ) с фиксированными фокусом  $F$  и директрисой.

Все невырожденные конические сечения, кроме окружности, можно описать следующим способом: выберем на плоскости точку  $F$  и прямую  $d$  (рис. 2.4) и зададим вещественное число  $e > 0$ . Тогда геометрическое место точек (ГМТ), для которых расстояние от точки  $F$  до прямой  $d$  отличается в  $e$  раз, является коническим сечением. Точка  $F$  называется фокусом конического сечения, прямая  $d$  — директрисой, число  $e$  — эксцентриситетом.

$$|FM| = e \cdot |MM'|, \quad MM' \perp d$$

В зависимости от эксцентриситета, получится:

- при  $e > 1$  — гипербола.
- при  $e = 1$  — парабола;
- при  $e < 1$  — эллипс.

Для окружности полагают  $e = 0$  (хотя формально при  $e = 0$ , ГМТ получается только точка  $F$ ).

В NX все конические кривые (эллипс, парабола и гипербола) строятся одной функцией:

```
int UF_CURVE_create_conic(
//вход:
UF_CURVE_conic_p_t conic_data, //структура параметров кривой
//выход:
tag_t * conic //тег созданной кривой
);
```

Структура первого параметра функции состоит из восьми полей:

```
struct UF_CURVE_conic_s {
tag_t matrix_tag; //тег матрицы X поворота координат объекта
```

```

int conic_type;           // подтип кривой
double rotation_angle;   // угол поворота относит. матрицы X
double start_param;      // параметр начала кривой объекта
double end_param;        // параметр конца кривой объекта
double center [ 3 ];     // координаты центра объекта
double k1;                // первый параметр объекта
double k2;                // второй параметр объекта
} ;

```

Что касается тега матрицы поворота координат объекта (первый параметр), то его задание и использование полностью совпадает с приемами, описанными в п. 2.5.1 - «Построение точек и кривых».

Подтип кривой задается тремя определенными константами:

```

UF_conic_ellipse_subtype - эллипс;
UF_conic_parabola_subtype - парабола;
UF_conic_hyperbola_subtype - гипербола.

```

Угол поворота объекта относительно матрицы X объекта (rotation\_angle) определяется углом  $\alpha$  (рис. 2.5), где оси XYZ олицетворяют векторы матрицы X.

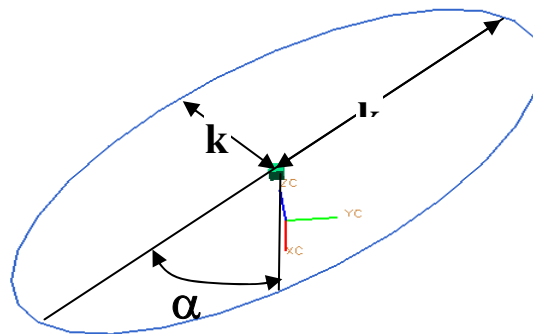


Рис.2.5 Схема конической кривой

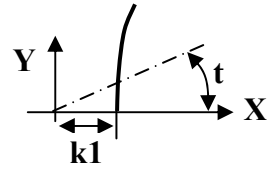
Коэффициенты  $k1$  и  $k2$  задают образующие параметры кривой. Для эллипса – это соответственно малая и большая полуось. Для других кривых назначения  $k1$  и  $k2$  показаны в таблице 2.1, при этом  $t$  – это параметр кривой (start\_param – начальное  $t$ , end\_param – конечное  $t$ ).

Таблица 2.1

Значения коэффициентов  $k1$  и  $k2$  для различных кривых

Обозначения	$k1$	$k2$	Формулы вычисления координат
Эллипс 	Малая полуось	Большая полуось	$X = k1 \cdot \cos(t)$ $Y = k2 \cdot \sin(t)$ $0 \leq t \leq 2\pi$

Продолжение таблицы 3.1

Обозначения	k1	k2	Формулы вычисления координат
Парабола 	Фокальный коэффициент	Не используется	$X = t^2 / k1$ $Y = t$ $t$ – любое вещественное число.
Гипербола 	Координата пересечения с осью X	Коэффициент кривизны	$X = k1 / \cos (t)$ $Y = k2 \cdot \sin (t) / \cos (t)$ $0 < t < \pi/2$

Пример кода модуля, строящего гиперболу в плоскости XY, представлен в листинге 2.2.

Листинг 2.2

```

/*****
Модуль построения гиперболы в 3D пространстве сцены NX
*****/
#include <uf.h>
#include <uf_object_types.h>
#include <uf_curve.h>
#include <uf_csys.h>

void ufusr(char *param, int *retcode, int paramLen)
{ tag_t conic_id, wcs_tag;
  UF_CURVE_conic_t orig_conic;
  UF_CURVE_conic_t read_conic;
  UF_CURVE_conic_t conv_conic;
  UF_CURVE_genconic_t gen_conic;
  logical conv_sense;

  if (!UF_initialize())
  {
//задание параметров гиперболы
  orig_conic.matrix_tag = NULL_TAG;
  orig_conic.conic_type = UF_conic_hyperbola_subtype;
  orig_conic.rotation_angle = 0.0 * DEGRA;
  orig_conic.start_param = 0.005;
  orig_conic.end_param = 1.5;

```

```

orig_conic.center[0] = 0.0;
orig_conic.center[1] = 0.0;
orig_conic.center[2] = 0.0;
orig_conic.k1 = 5;
orig_conic.k2 = 2.0;
//получение матрицы абсолютной системы координат
UF_CSYS_ask_wcs(&wcs_tag);
//связывание матрицы объекта (гиперболы) с полученной матрицей
UF_CSYS_ask_matrix_of_object(wcs_tag,
                             &orig_conic.matrix_tag)
;
// построение конической кривой (гиперболы)
UF_CURVE_create_conic(&orig_conic, &conic_id);
(А) → UF_terminate();
}
}

```

Результат работы модуля показан на рис. 2.6.

Кривая, созданная в этом примере, относится к объектам так называемого стандартного типа (standard form). Недостаток у них в том, что имеющаяся информация в параметрах информационной структуры не всегда четко поддерживает параметризацию объекта. Более качественно ее обеспечивают информационные структуры кривых основного типа (general form). Поэтому разработчикам прикладных модулей рекомендуется после создания стандартной кривой, откорректировать ее информационное наполнение «прогнав» информацию через структуры аналогичных объектов основного типа.

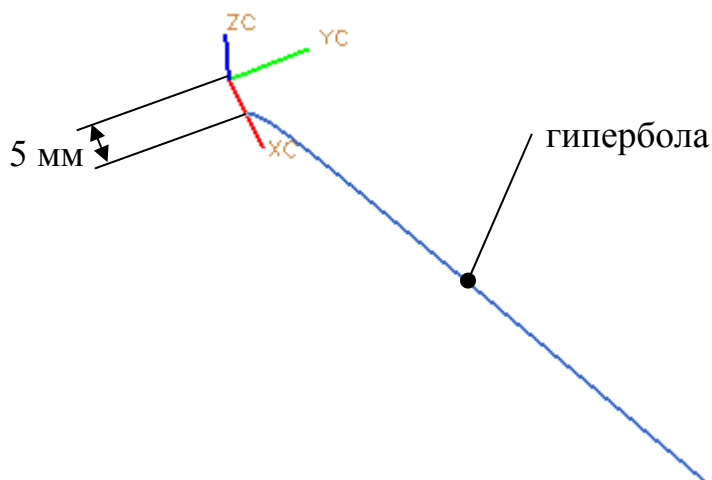


Рис. 2.6 Гипербола, построенная по листингу 2.2

Обеспечить эту операцию можно, вставив в область А листинга 2.2 следующие команды:

```
/* Прочитать информацию по заданной конической кривой */
UF_CURVE_ask_conic_data(conic_id, &read_conic);
/* Конвертировать информацию в главную форму */
UF_CURVE_convert_conic_to_gen(&read_conic,
                               &gen_conic);
/* Конвертировать информацию обратно в стандартную форму */
UF_CURVE_convert_conic_to_std(&gen_conic,
                              &conv_conic, &conv_sense);
```

### 2.3. Построение сплайнов

Математической моделью трехмерного сплайна является матричное произведение матрицы весовой функции положения ключевых точек на сплайне  $[F]$  и матрицы геометрической информации  $[G]$  в этих точках [5].

$$P_k(\tau) = [F][G].$$

Это произведение определяет вектор координат некоторой  $k$ -той точки сплайна, находящейся на относительном расстоянии  $\tau$  (по дуге сплайна) от его начала.

Компоненты матрицы весовой функции определяются через параметр  $t$  сплайна, где  $t$  – это длина дуги сплайна от начала сплайна до некоторой его расчетной точки. Для сплайна третьей степени, например, матрица весовой функции состоит из следующих элементов:

$$[F] = [F_1(\tau) \quad F_2(\tau) \quad F_3(\tau) \quad F_4(\tau)],$$

где

$$\tau = \begin{pmatrix} t \\ t_{k+1} \end{pmatrix},$$

$$F_{1k}(\tau) = 2\tau^3 - 3\tau^2 + 1,$$

$$F_{2k}(\tau) = -2\tau^3 + 3\tau^2,$$

$$F_{3k}(\tau) = \tau(\tau^2 - 2\tau + 1)t_{k+1},$$

$$F_{4k}(\tau) = \tau(\tau^2 - \tau)t_{k+1}.$$

Индексом  $k$  в этих формулах обозначен номер некоторой промежуточной ключевой точки на сплайне.

$$1 \leq k \leq n - 1$$

где  $n$  – количество ключевых точек, через которые строится сплайн.

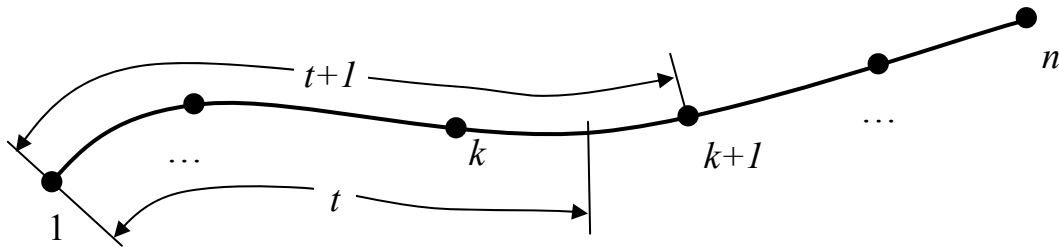


Рис. 2.7 Параметры сплайновой кривой

Понятие параметра  $t$  сплайна для некоторой промежуточной расчетной точки, лежащей между ключевыми точками  $k$  и  $k+1$ , проиллюстрировано рисунком 2.7.

Элементы матрицы  $[G]$  (для того же кубического сплайна) определяются из выражения

$$[G]^T = [P_k \quad P_{k+1} \quad P'_k \quad P'_{k+1}].$$

Здесь  $P_k$  и  $P_{k+1}$  – вектора координат ключевых точек сплайна,

$P'_k$  и  $P'_{k+1}$  – вектора первых производных (касательных) в этих точках.

## 2.4. Сплайны в Open API NX

Для построения сплайнов в Open API NX для языка C предусмотрено две функции:

UF\_CURVE\_create\_spline\_thru\_pts – построение сплайна по точкам;

UF\_CURVE\_create\_spline – построение произвольного сплайна.

Полный синтаксис первой функции выглядит следующим образом:

```
int UF_CURVE_create_spline_thru_pts (
    int degree,          // степень сплайна.
    int periodicity,    /*периодичность сплайна: 0- не периодич-
ный;
                        1 – периодичный. */
    int num_points,     // количество заданных точек.
    UF_CURVE_pt_slope_crvatr_t point_data[], /*массив
координат точек и направлений касательных в них.*/
    double parameters[], //параметры входных точек.
    int save_def_data,  /* 1 – сохранять входные данные при
создании кривой, 2 – не сохранять. */
    tag_t *spline_tag  //тег создаваемого объекта.
)
```

Первое поле в структуре отвечает за степень сплайна. Обычно она изменяется от 3 и выше.

Второе поле – периодичность сплайна. Замкнутые сплайны называются периодичными (рис. 2.8): начальная и конечная точки кривой у периодичных сплайнов совпадают. Все остальные сплайны – не периодичные.

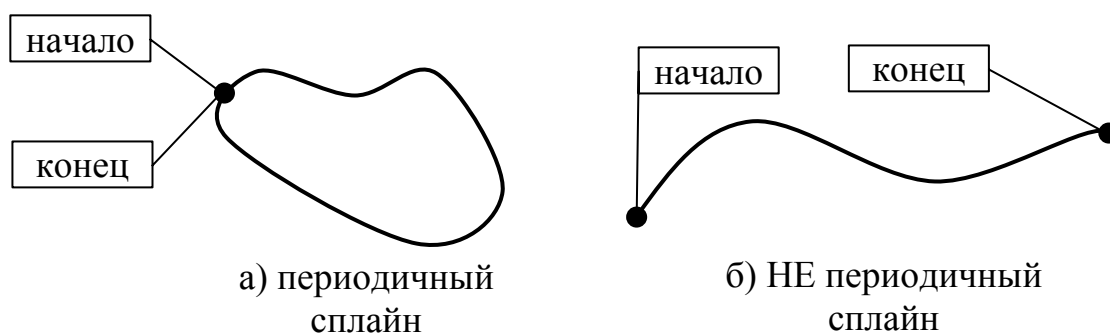


Рис.2.8 Виды сплайнов

Четвертое поле структуры – данные по точке, в свою очередь представляется структурой следующего вида:

```
struct UF_CURVE_pt_slope_crvatr_s
{
    double point[3];    //координаты точки.
    int slope_type;    //метод задания наклона кривой (см.ниже).
    double slope[3]; //вектор, задающий направление касательной.
    int crvatr_type;    //метод задания кривизны кривой.
    double crvatr[3];  //вектор, направления кривизны.
};
```

В первое поле этой структуры заносятся координаты точки (x, y, z).

Во второе поле - код метода задания касательной в точке. Варианты констант, определяющих метод задания касательной следующие:

UF\_CURVE\_SLOPE\_NONE – касательная не задается;  
 UF\_CURVE\_SLOPE\_AUTO – касательная определяется системой автоматически;

UF\_CURVE\_SLOPE\_VEC – касательная задается вектором (используется величина и направление вектора);

UF\_CURVE\_SLOPE\_DIR – вектором задается только направление касательной.

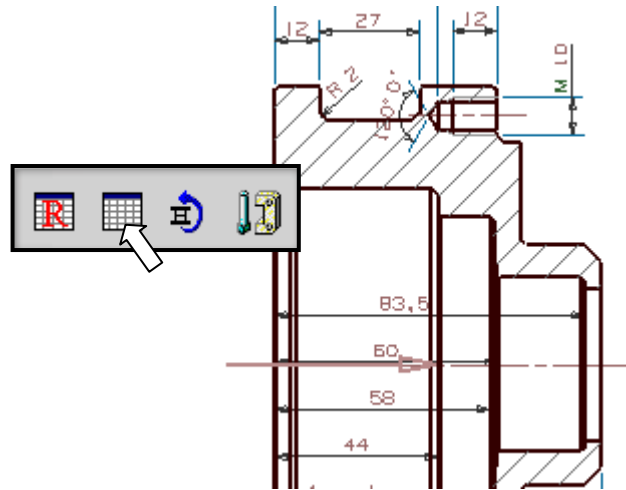
В третье поле заносится сам вектор касательной (его орты  $I, J, K$ );

Следующие два поля повторяют предыдущие два, но для вектора, задающего кривизну кривой в точке (величина, обратная радиусу кривой в точке). Константы для описания методов задания кривизны следующие:

UF\_CURVE\_CRVATR\_NONE – кривизна не назначена;  
 UF\_CURVE\_CRVATR\_AUTO\_DIR – автоматическое определение направления кривизны, при этом тип наклона кривой должен стоять UF\_CURVE\_SLOPE\_AUTO;

# 3

## МОДЕЛИРОВАНИЕ ОБЪЕКТОВ И ДЕЙСТВИЙ НАД НИМИ ФУНКЦИЯМИ OPEN API NX



### 3.1. Цилиндр

Чтобы в NX смоделировать какой-либо типовой объект, например – цилиндр (параллелепипед, шар, конус) достаточно одной функции. Для цилиндра это функция `UF_MODL_create_cyl1(...)`. В функции шесть параметров.

На входе в функцию:

- первый параметр – `UF_FEATURE_SIGN sign`, определяет тип булевой операции, которая должна произойти с создаваемым телом. Операция определяется predetermined константами:

- `UF_NULLSIGN` – если создается новое твердое тело;

- `UF_POSITIVE` – если создаваемое тело объединяется с уже существующим;

- `UF_NEGATIVE` – если создаваемое тело вычитается из существующего;

- `UF_UNSIGNED` – если создаваемое тело пересекается с существующим.

- второй параметр – `double origin[3]`, задает координаты точки привязки центра окружности основания цилиндра;

- третий параметр – `char* height`, указатель на текстовую строку, где прописана высота цилиндра;

- четвертый параметр – `char* diam`, указатель на текстовую строку с диаметром выстраиваемого цилиндра;

- пятый параметр – `double direction[3]`, задает вектор направления главной оси цилиндра;

На выходе функции всего один параметр – `tag_t* cyl_obj_id`, тег вновь созданного цилиндра.

Для опробования функции создадим небольшой программный модуль, который должен создать цилиндр в соответствии с рисунком 4.1





```

UF_terminate();
} }
/* а это типовая функция завершения приложения
   впоследствии мы её не будем приводить для экономии места */
int ufusr_ask_unload(void)
{ return (UF_UNLOAD_IMMEDIATELY); }

```

Запуск программы строит именно такой цилиндр, как показан на рис. 3.1.

### 3.2. Тело вращения

Создадим программный модуль, формирующий тело вращения из сечения, представленного на рис. 3.2

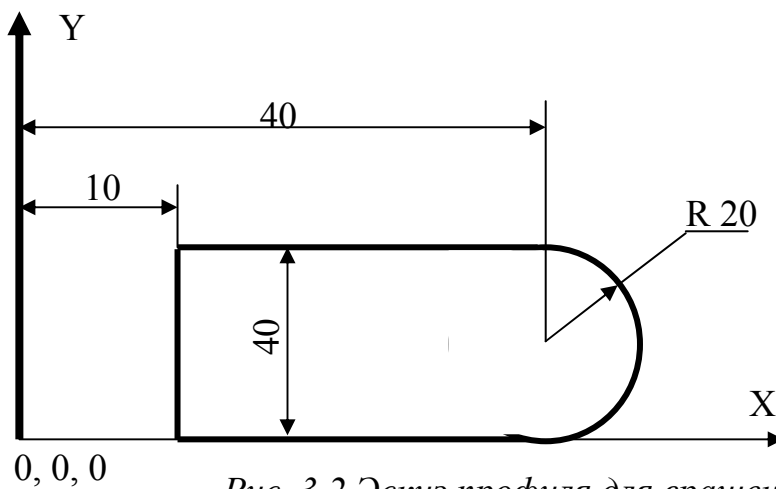


Рис. 3.2 Эскиз профиля для вращения

Вектор вращения будет совпадать с осью Y.

Для программирования операции используем функцию:

```

int UF_MODL_create_revolved(
    uf_list_p_t obj_id_list, //перечень объектов вращения
    char **limit,           //пределы вращения
    double point[3],        //точка начала вектора вращения
    double direction[3],    //направление вектора вращения
    UF_FEATURE_SIGN sign, /*логические операции при создании
                           тела вращения */
    uf_list_p_t *feature_list //перечень созданных объектов
)

```

Такие операции как вытягивание, вращение и т.п. на программном уровне проводятся с набором кривых, предварительно занесенных в специальный массив типа `uf_list_p_t` – перечень объектов.

Перечень объектов предварительно создается функцией:

```

UF_MODL_create_list(uf_list_p_t list),

```

а затем заполняется функцией:

```
int UF_MODL_put_list_item(
uf_list_p_t list , //указатель на заполняемый перечень
tag_t obj_id      //тег объекта, заносимого в перечень
);
```

Пределы вращения в функции `UF_MODL_create_revolved` – это массив из двух строк, в которых в текстовом виде заносятся значения начального и конечного угла вращения сечения, причем значения – в градусах.

Точка начала вектора вращения и его направление – два стандартных массива по три элемента. А вот объединение `UF_FEATURE_SIGN`, это новый элемент программирования. Объединение содержит список констант, определяющих логическую операцию с окружающими тело объектами в момент его создания. Константы следующие:

- `UF_NULLSIGN` – создается новое самостоятельное тело;
- `UF_POSITIVE` – тело объединяется с пересекаемыми телами;
- `UF_NEGATIVE` – тело вычитается из пересекаемых тел;
- `UF_UNSIGNED` – создается тело пересечения создаваемого тела с имеющимися.

Программный код модуля, строящего тело вращения, показанное на рис. 3.3, представлен в листинге 3.2

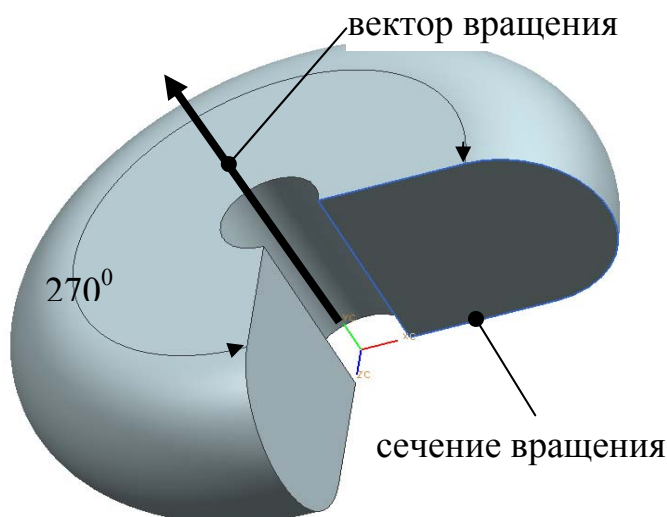


Рис. 3.3 Тело, полученное в результате выполнения функции вращения

Листинг 3.2

```
#include <uf.h>
#include <uf_curve.h>
#include <uf_csys.h>
#include <uf_modl.h>
```

```

void ufusr(char *param, int *retcode, int param_len)
{
/* Определяется массив линий с координатами их концов, используемых в
эскизе вращаемого сечения */
UF_CURVE_line_t line[3]={
    {{10.0,0.0,0.0},{10.0,40.0,0.0}},
    {{10.0,0.0,0.0},{40.0,0.0,0.0}},
    {{10.0,40.0,0.0},{40.0,40.0,0.0}}
};

    UF_CURVE_arc_t arc; //структура данных дуги
    tag_t objarray[4], //теги для трех линий и дуги
        wcs_tag;      //тег для системы координат
//предельные углы вращения сечения
char *limit[2] = {"0.0", "270.0"};
//точка начала вектора вращения
double point[3] = {0.0, 0.0, 0.0};
//орты вектора вращения
double direction[3] = {0.0, 1.0, 0.0};
//признак создания самостоятельного тела
UF_FEATURE_SIGN sign = UF_NULLSIGN;
//заготовки указателей на перечни объектов
uf_list_p_t loop_list, features;
int i;
if (!UF_initialize())
{
//задание параметров дуги в сечении
    arc.start_angle = -90.0 * DEGRA;
    arc.end_angle = 90.0 * DEGRA;
    arc.arc_center[0] = 40.0;
    arc.arc_center[1] = 20.0;
    arc.arc_center[2] = 0.0;
    arc.radius = 20.0;
// построение трех линий сечения
    for (i=0; i<3; i++)
        UF_CURVE_create_line(&line[i], &objarray[i]);
// получение матрицы абсолютных координат
    UF_CSYS_ask_wcs(&wcs_tag);
// связывание матрицы поворота дуги с абсолютными координатами
    UF_CSYS_ask_matrix_of_object(wcs_tag,
                                &arc.matrix_tag);
// построение дуги сечения
    UF_CURVE_create_arc(&arc, &objarray[i]);
//создание пустого перечня объектов

```

```

    UF_MODL_create_list(&loop_list);
// заполнение перечня тремя линиями и одной дугой
    for(i = 0; i < 4; i++)
        UF_MODL_put_list_item(loop_list, objarray[i]);
// создание тела вращения
    UF_MODL_create_revolved(loop_list, limit, point,
                            direction, sign, &features);

    UF_terminate();
}

```

### 3.3. Удаление объектов

Если требуется удалить какой либо из объектов сцены, можно использовать функцию `UF_OBJ_delete_object(...)`, где в качестве единственного входного параметра используется тег тела, подлежащего удалению. Однако, следует помнить, что не допускается использование этой функции при переборе объектов с помощью `UF_OBJ_cycle...` и им подобных. Удаление объекта в момент перебора с помощью указанных функций, вызовет нарушение работы программы. Рекомендуется сначала собрать информацию об удаляемых тегах в массив, с помощью тех же функций перебора, а уже затем удалить эти объекты.

Сценарий работы примера будет следующий: перед запуском модуля пользователь должен создать на сцене несколько одиночных тел. При запуске модуля система должно появиться диалоговое окно, позволяющее пользователю указать тело, подлежащее удалению. Это действие выполняется мышью, указанное тело должно подсветиться, после чего пользователь подтверждает удаление, нажатием кнопки на диалоге и тело исчезает со сцены.

Чтобы реализовать такой сценарий нам потребуется функция для создания стандартного системного диалога выбора тела в NX. Здесь мы используем функцию `UF_UI_select_single(...)`. (Более подробно о функциях системного интерфейса см. в главе 5).

На примере этой функции разберем методологию поиска информации о функциях Open API в установленной системе NX.

Обычно задают поиск имени интересующей функции среди \*.h файлов в каталоге UGOPEN системы NX. наша функция найдется в файле `uf_ui.h`. Вот пример стандартного описания функции в \*.h файлах:

```

extern UGOPENINTEXTPORT int UF_UI_select_single(
    char * message, /* <I> Cue line message to display. */
UF_UI_selection_options_p_t opts, /* <I> Selection options.
                                See the Data structures section of this chapter.
                                */

```

```

int * response, /* <O> response:      1 = Back
                                       2 = Cancel
                                       4 = Object selected by name
                                       5 = Object selected */
tag_p_t object, /* <O> Object identifier of selected object */
double cursor[3], /* <O> Cursor position. This is undefined if
                   response is 4 (object selected by name).
                   */
tag_p_t view /* <O>View of selection. This is NULL_TAG
              if response is 4 (object selected by name). */
);

```

Из описания этой функции становится понятным, что в этой функции шесть параметров и предназначена она для выбора на сцене UGS только одного объекта. Те параметры функции, которые являются входными обозначены символом <I> , а те, которые являются выходными – символом <O> . Таким образом, у рассматриваемой функции два входных и четыре выходных параметра.

Первый входной параметр – указатель на текстовую строку с сообщением, которое появляется в строке статуса системы при активизации диалога. Здесь обычно пишут фразу-указание пользователю, что он должен сделать при появлении диалогового окна.

Второй входной параметр указатель на структуру UF\_UI\_selection\_options\_t, определяющую шаблон выбора объекта на сцене системы. То, что это указатель, а не сама структура задается префиксом \_p\_ в имени переменной (это общее соглашение при обозначении переменных в NX). Чтобы разобраться с этой структурой снова даем поиск по файлам \*.h в каталоге UGOPEN и находим её в файле uf\_ui\_types.h (рекомендуется поиск структур проводить по имени без префиксов, то есть искать UF\_UI\_selection\_options\_):

```

struct UF_UI_selection_option_s
{
    int num_mask_triples;
    UF_UI_mask_p_t mask_triples; /* <len: num_mask_triples> */
    int scope; /* scopes are listed in uf_ui.h */
    int other_options; /* initially ignored (set to 0) */
    void *reserved; /* initially ignore (set to NULL) */
};

```

Видим, что в структуре пять полей. Первое поле не описано, но по комментариям ко второму полю становится понятным, что первое поле содержит количество структур масок, задаваемых при выборе объекта на

сцене. Отсюда ясно, что хотя функция выбирает всего один объект, но типов объектов при выборе может быть задано несколько.

Второе поле в структуре, следовательно – указатель на массив масок, каждая из которых сама представляет структуру типа `UF_UI_mask_t`.

Для объяснения третьего поля нас отправляют к файлу `uf_ui.h`, в котором мы находим объяснение, что `scope` – это константа, определяющая область выбора объекта:

```
Selection scope:  
UF_UI_SEL_SCOPE_NO_CHANGE  
UF_UI_SEL_SCOPE_ANY_IN_ASSEMBLY  
UF_UI_SEL_SCOPE_WORK_PART  
UF_UI_SEL_SCOPE_WORK_PART_AND_OCC
```

Из названий констант видно, что первая определяет выбор объекта без его последующего изменения, вторая определяет выбор объекта на сборочной модели, третья на рабочей сцене, а четвертая на рабочей сцене или подборке.

Четвертое и пятое поле структуры `UF_UI_selection_option` не задаются.

Чтож, осталось выяснить, что из себя представляет структура маски - `UF_UI_mask_t`. Находим ее опять же в файле `uf_ui_types.h`:

```
struct UF_UI_mask_s  
{  
    int object_type; /* This can be one of the object types that are  
                    listed in uf_object_types.h */  
    int object_subtype; /* This can either be UF_all_subtype, or a  
                        proper subtype of the object type selected.  
                        This is ignored for UF_solid_type */  
    int solid_type; /* This is only meaningful when the object_type is  
                   UF_solid_type or UF_feature_type.  
                   In that case, this should be set to  
                   one of the solid type constants.  
                   This is ignored for UF_all_subtype  
                   */  
};
```

Видим, что первое поле структуры должно содержать код типа выбираемого объекта, а коды эти описаны в файле `uf_object_types`.

Второе поле структуры – это код подтипа объекта (из того же описательного файла), а третье поле применяется только для твёрдого тела или его элементов и игнорируется для всех подтипов объектов.

Теперь вернемся обратно к параметрам разбираемой функции.

У нас остались выходные параметры. Третий параметр (`int * response`) – при закрытии диалога получает код нажатой на нем кнопки. Четвертый параметр (`tag_p_t object`) – получает тег выбранного объекта. Пятый параметр (`double cursor[3]`) – получает координаты курсора в момент закрытия диалога, а шестой (`tag_p_t view`) – тег области просмотра, в которой отображен выбранный объект (этот тег используется в ряде функций из библиотеки работы с дисплеем NX (см. описание в `uf_view.h`)).

В завершение разбора функции одиночного выбора следует отметить, что и маску и сценарий выбора объектов в ней можно динамически менять, в ходе выполнения программы, с помощью функций

```
UF_UI_set_sel_mask(...) и
        UF_UI_select_with_single_dialog(...).
```

Что ж, вооружившись этой информацией попробуем составить нужную нам программу. То, что у нас получилось и соответствует описанному выше сценарию, представлено в листинге 3.2.

Листинг 3.3

---

```
//Программа удаления выбранного тела со сцены
#include <uf.h>
#include <uf_ui.h>
#include <uf_obj.h>

void ufusr(char *param, int *retcode, int param_len)
{ //определяем две переменные тегов первую – для удаляемого тела
// вторую – для области просмотра объекта
  tag_t body,view;
//определяем структуру для настройки диалога
  UF_UI_selection_options_t opts;
//определяем структуру для маски выбора и сразу заносим в ее поля, что
//объектом выбора является твердое тело и элементы (FEATURE) твердого
//тела
  UF_UI_mask_t mask = {UF_solid_type,0,
                      UF_UI_SEL_FEATURE_BODY};
//определяем переменную для приема кода нажатой кнопки на диалоге
//и переменную для приема кода ошибки выполнения функции
  int response=2, error=0;
//и массивчик для получения текущих координат мыши
  double cursor[3];
//далее заполняем поля управляющей структуры диалога
  //сначала – определим, что у нас будет только одна маска
  opts.num_mask_triples = 1;
  //адрес этой маски занесем в требуемое поле структуры
```



```
opts.mask_triples = &mask;
//и определим, что выбор будет происходить на текущей рабочей сцене
opts.scope = UF_UI_SEL_SCOPE_WORK_PART;
if (!UF_initialize())
{
  //откроем диалог выбора объекта
  error = UF_UI_select_single("Выделите твердое тело",
                             &opts,
                             &response,
                             &body, cursor, &view);
//если не было ошибки при работе функции выбора
//и пользователь не нажимал ни кнопку НАЗАД ни ОТМЕНА
if (!error && response != 1 && response != 2)
  //то производим удаление выбранного объекта
  UF_OBJ_delete_object(body);

//в любом случае завершаем работу библиотеки Open API
  UF_terminate();
}
}
```

Для проверки работы приложения предварительно постройте на сцене несколько твердых тел. Нами, например было построено два шара (рис. 3.4).

При запуске приложения появляется диалог выбора объектов (1 рис. 3.4). Мы мышью выбираем один шар и он удаляется со сцены. программа завершает свою работу.

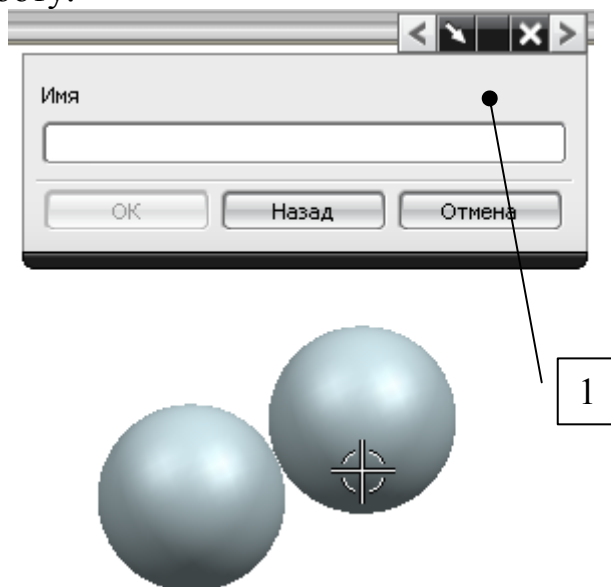


Рис. 3.4 Экранная форма работы программы листинга 3.3

В заключение можно отметить, что для удаления группы объектов используется функция `UF_OBJ_delete_array_of_objects(...)`, которую чита-

телю предлагается разобрать самостоятельно (по приведенной выше методике) и построить приложение, которое удаляет сразу несколько тел, указанных пользователем на сцене.

### 3.4. Копирование объектов

Разберем задачу о копировании объектов в NX программным путем. Для начала организуем копирование простого объекта из раздела элементы проектирования, например – шара (рис. 3.5) (блока, цилиндра или конуса).

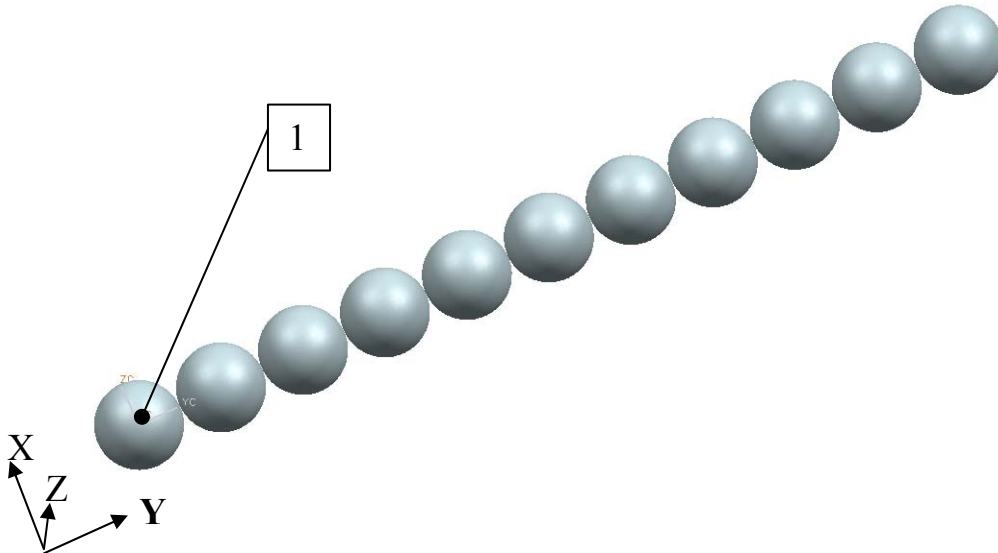


Рис. 3.5 Схема копирования шара вдоль оси Y

Исходный элемент находится в произвольной точке системы координат (на рисунке – в ее начала) и необходимо получить 10 копий этого элемента на равных расстояниях друг от друга вдоль оси X.

Для решения задачи воспользуемся двумя функциями:

`UF_MODL_copy_paste_features(...)` – для получения копии заданного элемента, и `UF_MODL_move_feature(...)` – для перемещения копии в заданную точку сцены.

У первой функции восемь параметров, из них семь первых – входные и один, последний, выходной.

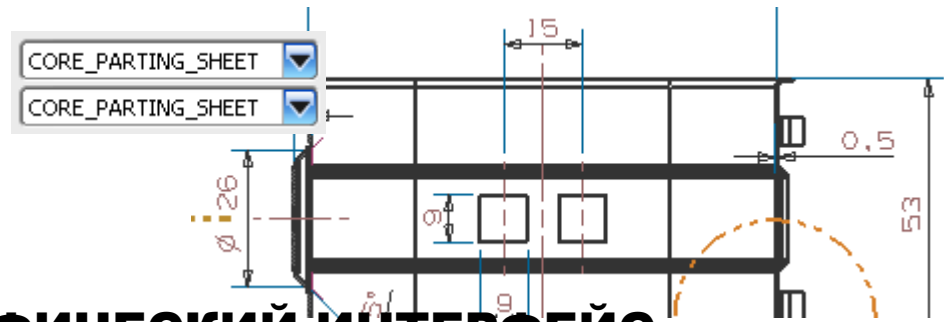
Первый параметр – указатель на массив тегов объектов, подлежащих копированию. Это удобно, когда копируется объект, состоящий из нескольких типовых элементов. В нашем случае копируется один элемент и указатель будет адресовать только его.

Второй параметр – количество элементов в предыдущем массиве (у нас – один).

Третий параметр – указатель на массив тегов «родителей» копируемых элементов. У нашего шара их нет, поэтому будем в этом параметре ставить ноль. Конечно, более грамотно было бы программным путем убедиться, что у копируемого элемента нет родителей, а если они есть, то

# 4

## ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПРИЛОЖЕНИЙ ДЛЯ NX



### Функции графического интерфейса

Важным элементом любого приложения является его интерфейс с пользователем. В NX Open API организация интерфейса с пользователем может быть выполнена четырьмя путями:

1. использование специализированных функций для организации интерфейса;
2. использование визуального конструктора User Styler Interfeys для построения окон диалога;
3. использование визуального конструктора Block Styler для построения окон диалога;
4. использование системных библиотек Windows API (или аналогичных библиотек «третьих фирм»).

#### 4.1. Организация интерфейса с помощью специализированных функций

В NX Open API специализированные функции для построения пользовательского интерфейса (для языка C) перечислены в файле описаний `uf_ui.h`.

##### 4.1.1. Вывод текстовых сообщений на информационных линейках

На главном рабочем окне NX ниже инструментальных линеек расположены информационные линейки системы, куда выводятся оперативные сообщения. Таких линеек две: линейка статуса (1 рис. 4.1) и линейка приглашения (2 рис. 4.1).

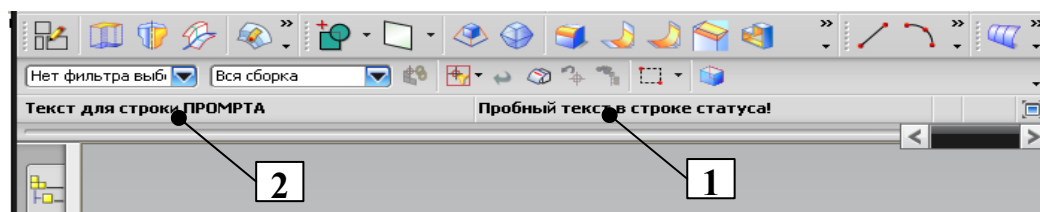


Рис. 4.1 Области вывода текстовых сообщений в NX

Вывод текста в первую линейку осуществляет функция `UF_UI_set_status(...)`, во вторую - `UF_UI_set_prompt(...)`. Например, команда:

```
UF_UI_set_status("Пробный текст в строке статуса!")
```

выводит текст, как показано на рис.4.1 поз.1.

#### 4.1.2. Получение информации о координатах точек

Координаты точек можно снять с 3D сцены NX с помощью функции `UF_UI_select_point_collection(...)`. Пример команды в коде модуля, вызывающей эту функцию следующий:

```
UF_UI_select_point_collection("Получение точек",
    FALSE, //запрещено возвращать совпадающие точки
    &points, //указатель на массив структур с данными по точкам
    &count, //количество полученных точек
    &response // код кнопки, нажатой на диалоге );
```

Вызов функции активирует на экране диалог (рис. 4.2а).

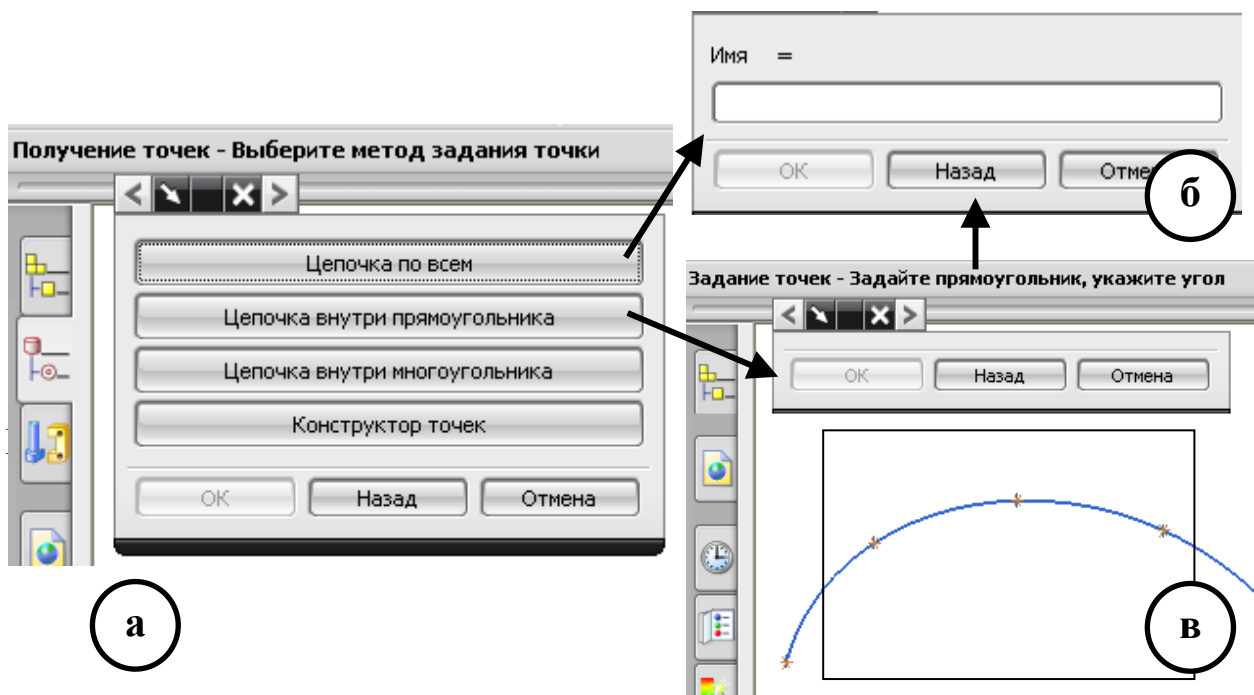


Рис. 4.2 Формы диалога координат точек

Кнопка «Цепочка по всем» запрашивает первую и последнюю точку в цепочке выбираемых точек (рис. 4.2б). Это работает, если при создании точки действительно были объединены в цепочку: например – принадлежат одной кривой.

Кнопка «Цепочка внутри прямоугольника» сначала открывает диалог (рис. 2в) для создания прямоугольника, внутри которого будут выбираться точки, а потом переходит к этапу (рис.2а), где задаются начальная и конечная точка в цепочке.

Кнопка «Цепочка внутри многоугольника» аналогична предыдущему режиму, только на этапе (рис.2в) на экране создается не прямоугольник, а многоугольник, охватывающий выбираемые точки.

Кнопка «Конструктор точек» запускает стандартный диалог задания (выбора) точек на 3D сцене (рис.3).

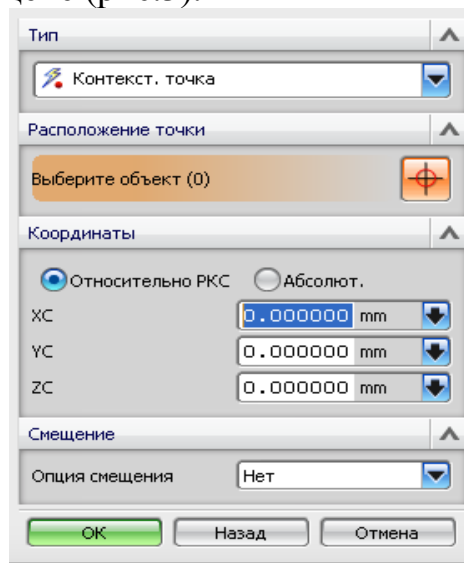


Рис. 3 Вид стандартного диалога задания (выбора) точек

С помощью него можно не только указать на существующие точки, но и сразу построить новые, если это необходимо.

#### 4.1.3. Диалог выбора объектов заданного типа

Существует несколько диалогов для выбора на рабочей сцене NX требуемых пользователем объектов.

Для выбора на 3D сцене объектов заданного (одного) типа используется функция `UF_UI_select_single(...)`. Пример использования этой функции приведен в листинге 3.2. При вызове функции с помощью специальной структуры (маски) задается тип объектов, подлежащих выбору. Это может быть твердое тело (`body`), элементы тела (`feature`), грани (`face`) и т.д. Функция возвращает массив тегов выбранных компонентов. При запуске функции возникает стандартное окно выбора (рис. 2б), на котором имеется три кнопки управления и текстовое поле для выбора объекта по имени. При этом кнопка ОК недоступна, пока не произведен корректный выбор.

Множественный выбор объектов (разного типа) может быть произведен диалогом, запускаемым функцией `UF_UI_select_feature(...)`. У функции четыре параметра, два на входе в функцию:

- 1 – текст, выводимый в строке пояснений NX;

2 – фильтр выбираемых объектов. Для фильтра зарезервированы константы:

UF\_UI\_FEAT\_SEL\_TYPE\_BROWSEABLE – все доступные элементы (features) тела;

UF\_UI\_FEAT\_SEL\_TYPE\_NO\_BOOLEAN\_UDF – все, кроме логических операций и udf элементов (осей, векторов, плоскостей) тела;

и три на выходе:

3 – количество полученных элементов;

4 – массив указателей на теги полученных элементов;

5 – код нажатой кнопки на форме диалога, один из следующих:

UF\_UI\_BACK – была нажата кнопка НАЗАД (BACK);

UF\_UI\_CANCEL – была нажата кнопка ОТМЕНА (CANCEL);

UF\_UI\_OK – была нажата кнопка ОК.

Пример вызова функции выглядит следующим образом. Предварительно следует описать применяемые в функции переменные:

```
int response; /*переменная для получения кода нажатой на диалоге кнопки */
```

```
int count; /*переменная для получения количества выбранных элементов */
```

```
tag_t *feature_tags; //массив тегов выбранных элементов.
```

Далее, в том месте кода модуля, где следует вызов функции множественного выбора, можно предложить такой код:

```
UF_UI_select_feature("Множественный выбор",
UF_UI_FEAT_SEL_TYPE_BROWSEABLE, &count,
&feature_tags, &response );
```

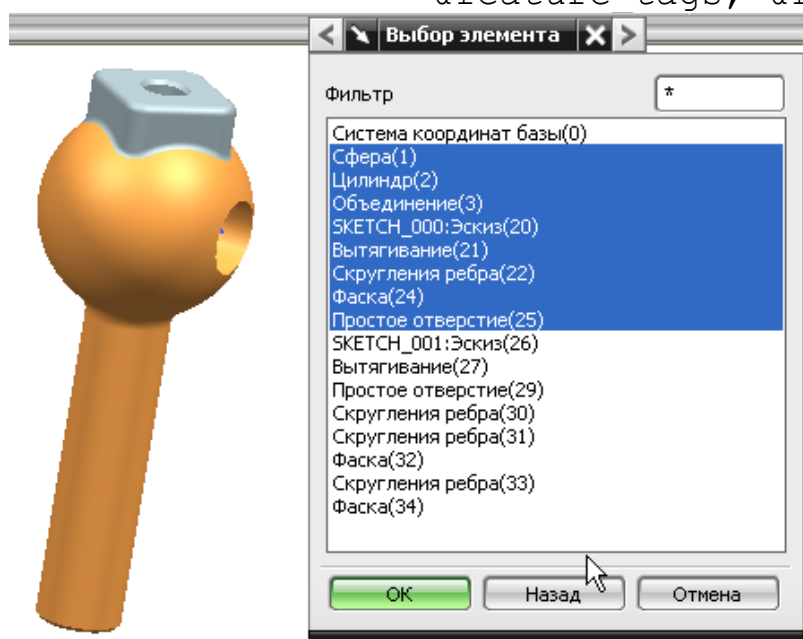


Рис. 4.4 Форма диалога выбора элементов тела

Выполнение этого кода приводит к появлению на экране формы диалога выбора (рис. 4.4). В списке диалога сразу высвечиваются все доступные на сцене для выбора элементы имеющихся тел. Пользователь должен выделить в этом списке те элементы, теги которых он хочет получить в программу и нажать ОК.

По результатам выполнения примера, представленного на рис. 4.4, переменная `count` получит значение 8, переменная `response` значение `UF_UI_OK`, а переменная `feature_tags` получит указатель на массив из восьми тегов выбранных на диалоге элементов.

Еще один диалог для выбора – это функция `UF_UI_select_parameters(...)`. Этот диалог позволяет выбрать теги параметров заданного элемента (`feature`) тела. Количество и назначение параметров в этой функции аналогично предыдущей, только вместо фильтра объектов задается тег «разбираемого на параметры» элемента.

Пример использования функции можно реализовать как продолжение предыдущего примера. Предположим, что на рис. 4.4 выбран только один элемент – «Цилиндр». И, как продолжение предыдущего кода примера, обработаем тег выбранного цилиндра, разобрав его на параметры:

```
if (count > 0)
    UF_UI_select_parameters(
        "Диалог выбора параметров элемента",
        feature_tags[0], &count, &exp_tags, &response);
```

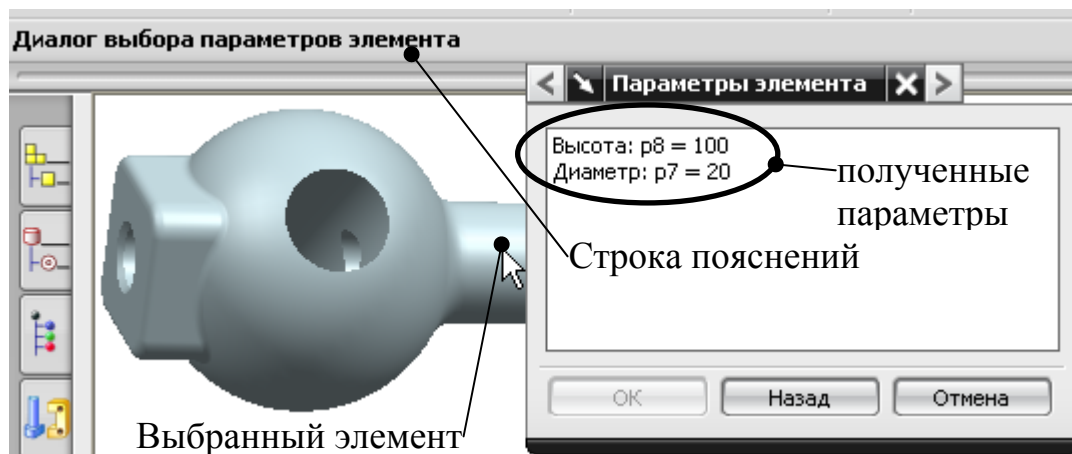


Рис. 4.5 Диалог выбора параметров элемента

При выполнении этого кода появляется окно диалога (рис. 4.5). В списке диалога перечислены параметры указанного цилиндра. Пользователь должен выбрать в списке те параметры, которые он хочет получить в программу и нажать кнопку ОК.

В примере функция выполняется при условии, что предыдущая функция по выбору элементов, вернула хотя бы один элемент.

В заключение параграфа отметим, что при использовании любой стандартной функции (или формы) выбора объектов в NX мы не ограни-

чены выбором только в окне геометрии – можно использовать для выбора объектов и навигатор построения и навигатор сборки.

#### 4.1.4. Выбор эскизов, имеющих в текущей сцене

Функция, вызывающая диалог выбора эскизов, имеет вид: `UF_UI_select_sketch(...)`. У функции четыре параметра. На входе она получает строку с текстом, выводимую, как обычно, на окне NX в строке (prompt) пояснений, и NULL, в качестве второго параметра (разработчики пока не нашли ему применения и зарезервировали на будущее). На выходе функция возвращает третий параметр – тег выбранного эскиза (scetch) и четвертый параметр – код нажатой на диалоге кнопки.

Перед вызовом функции, в области описания переменных, надо зака-зать переменную для получения тега выбранного эскиза:

```
tag_t sketch_tag;
```

и переменную для получения кода нажатой кнопки диалога:

```
int response;
```

Потом, в коде модуля, можно вызывать функцию выбора эскизов:

```
UF_UI_select_sketch("Диалог выбора эскизов", NULL,  
                    &sketch_tag, &response);
```

Такой вызов приведет к появлению диалога (рис. 4.6).



Рис. 4.6 Диалог выбора эскизов

На нем имеется список обнаруженных на сцене эскизов (scetchs), и пользователь может выбрать нужный, для передачи его тега в программу, после нажатия клавиши ОК.



Выбранный эскиз можно проанализировать на наличие в нем размеров и выбрать теги нужных размеров с помощью диалога `UF_UI_select_sketch_dimensions(...)`. Функция диалога имеет параметры, созвучные с параметрами предыдущих функций и читатель, используя накопленный опыт, сможет самостоятельно разобрать их назначение по описанию функции, приведенном разработчиками в файле `uf_ui.h`.

Пример вызова функции может быть представлен, как продолжение предыдущего примера, следующей командой:

```
if (response==UF_UI_OK)
    UF_UI_select_sketch_dimensions(
        "Диалог по выбору размеров эскиза",
        sketch_tag, //тег эскиза из предыдущего примера
        &count,     //количество выбранных размеров в эскизе
        &exp_tags, //тэги выбранных размеров
        &response  // код нажатой клавиши
    );
```

В этом примере функция вызывается, если в предыдущем примере диалога выбора эскизов была нажата кнопка ОК. Если это так, и в выбранном эскизе есть хоть один проставленный размер, то появляется диалог рис. 4.7.

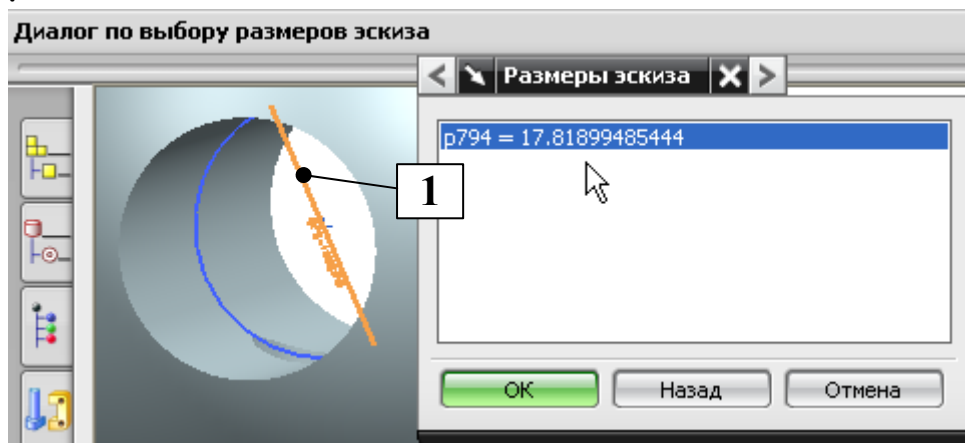


Рис. 4.7 Диалог по выбору размеров эскиза

В списке диалога перечислены все, имеющиеся на эскизе размеры. При выборе какой-либо строки в списке, соответствующие размеры эскиза подсвечиваются на экране монитора (1 рис. 4.7).

#### 4.1.5. Диалог построения плоскости.

Новая плоскость строится с помощью диалога, вызываемого функцией `UF_UI_specify_plane(...)`. В качестве входных параметров функции выступает три величины:

1. текстовая строка для вывода в области строки пояснений системы;

2. целочисленный код, определяющий метод построения плоскости при открытии диалога. Надо сказать, что после закрытия диалога в этой же переменной возвращается действительный код метода, которым воспользовался оператор при построении плоскости;
3. целочисленный код, определяющий отображать (0) или не отображать (1) построенную плоскость на экране.

Что касается выходных параметров, то их в функции четыре штуки:

1. целочисленный код нажатой на диалоге кнопки;
2. матрица поворота объекта (в данном случае – созданной плоскости), выдаваемая в виде девятиэлементного массива двойной точности;
3. координаты опорной точки созданной плоскости (трехэлементный массив двойной точности);
4. указатель на тег созданной плоскости.

Пример использования функции построения плоскости представлен в листинге 4.1. Код модуля приведен полностью, поскольку этот модуль, в дальнейшем, будет использован для исследования правил формирования матрицы поворота объекта.

#### Листинг 4.1

```
//построение плоскости с выводом полученной матрицы поворота
#include <stdio.h>
#include <uf.h>
#include <uf_ui.h>

void ufusr(char *param, int *retcode, int param_len)
{ int mode=5; /*брат по умолчанию плоскости мировой
              системы координат */
  int response; //ячейка кода нажатой на диалоге кнопки
  int i, j; //счетчики циклов
  double plane_matrix[9]; //матрица поворота объекта
  double plane_origin[3]; //опорная точка плоскости
  tag_t plane_tag; //тег построенной плоскости
  char mes[132]; //строка вывода текстовой информации

  if (!UF_initialize())
  { //стандартный диалог построения плоскости
    UF_UI_specify_plane(
      "Диалог построения плоскости",
      &mode,
      0,
      &response,
      plane_matrix,
```

```

        plane_origin,
        &plane_tag );
// Отобразить информационное окно системы
    UF_UI_open_listing_window();
UF_UI_write_listing_window("МАТРИЦА ПОВОРОТА =\n");
// Вывести в окно матрицу поворота
    for(i=0, j=0; i<9; i++,j++){
        if(j>2){j=0;
UF_UI_write_listing_window("\n");}
        sprintf(mes,"%9.6f  ", plane_matrix[i]);
        UF_UI_write_listing_window(mes);
    }

// Вывести в окно координаты опорной точки плоскости
UF_UI_write_listing_window("\nОПОРНАЯ ТОЧКА =\n");
    sprintf(mes,"%9.6f  %9.6f  %9.6f\n",
    plane_origin[0], plane_origin[1], plane_origin[2]);
    UF_UI_write_listing_window(mes);
    UF_terminate();
}
}

//процедура выхода из модуля
int ufusr_ask_unload(void)
{ return (UF_UNLOAD_IMMEDIATELY); }

```

В приведенном примере характеристики построенной плоскости: элементы матрицы поворота и координаты опорной точки плоскости выводятся, для наглядности и возможности анализа результатов, в информационное окно. (рис. 4.8)

При запуске модуля сначала появляется диалог 1 рис. 4.8, затем, в зависимости от выбранного метода построения плоскости, промежуточный диалог (2 рис. 4.8), после чего в информационном окне выводятся характеристики построенной плоскости (3 рис. 4.8), и сама плоскость отображается на экране (4).

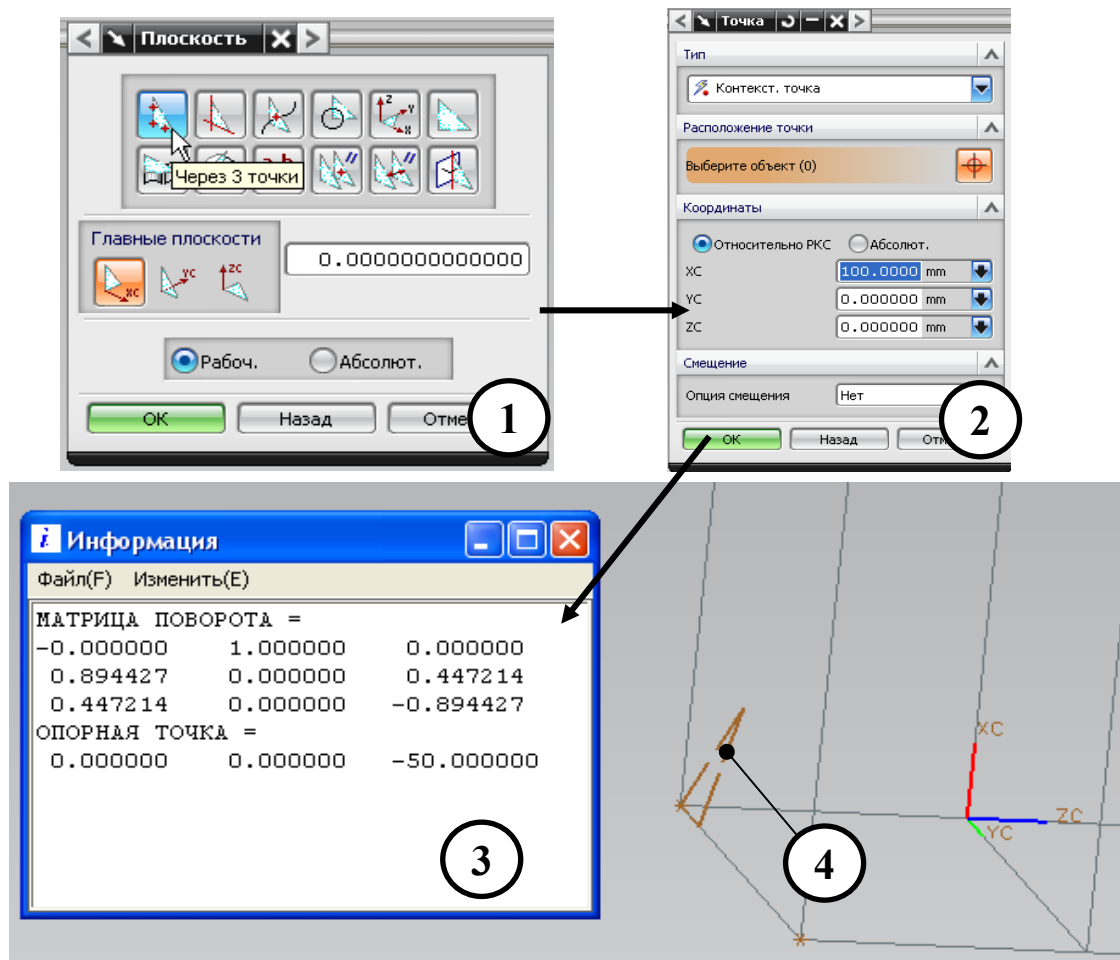


Рис. 4.8 Формы работы программы Листинг 4.1

#### 4.1.6. Диалог построения вектора

Для построения нового вектора предусмотрено диалоговое окно, вызываемое функцией `UF_UI_specify_vector(...)`. Параметры функции схожи с предыдущей и разобрать их теперь читателю не составит труда самостоятельно по файлу `uf_ui.h` системы NX. Пример использования функции может быть проиллюстрирован следующими командами:

в области данных описывает требуемые функцией переменные:

```
int response;           //код нажатой на диалоге кнопки
double direction[3];   //вектор направления
double origin[3];      //координаты опорной точки вектора
int mode =UF_UI_TWO_POINTS; /* код метода построения вектора
                               в области кода модуля вызов функции выглядит так: */
UF_UI_specify_vector("Диалог построения вектора",
                    &mode,
                    UF_UI_DISP_TEMP_VECTOR, //код режима отображения вектора
                    direction,
                    origin,
```

```
&response );
```

При запуске функции на экране появляется диалог (1 рис.4.9), где можно выбрать метод построения вектора и построить его на модели (2 рис. 4.9).

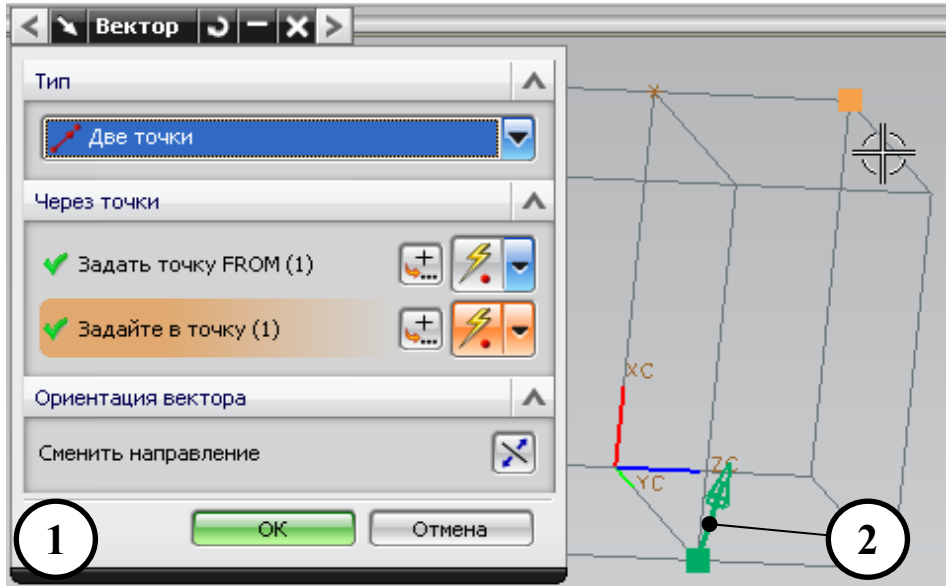


Рис. 4.9 Вид диалога построения вектора

После нажатия клавиши ОК в программу возвращаются параметры построенного вектора: направление и опорная точка (`direction` и `origin`) и код нажатой на диалоге клавиши (`response`).

#### 4.1.7. Диалог построения точки (конструктор точек)

Диалог построения точки вызывается функцией:

`UF_UI_point_construct(...)`. Параметры у функции следующие:  
входные параметры:

`char *cue` - текст, выводимый в информационной строке;

входной и выходной параметр:

`UF_UI_POINT_base_method_t *base_method` – указатель на элемент перечисления, задающий метод выбора точки по умолчанию. Обозначения допустимых методов выбора описаны в файле `uf_ui_types.h`;

`tag_t *point_tag` – тег созданной точки. По умолчанию она невидима на экране. Чтобы сделать ее видимой используется функция `UF_SO_set_visibility_option(...)` (описана в файле `uf_so.h`). Если же диалог был прерван без выбора точки, то сюда возвращается `NULL_TAG`;

`double base_pt[3]` – абсолютные координаты созданной точки;

`int *response` – код нажатой кнопки на диалоге:

`UF_UI_OK` – нажата кнопка ОК;

`UF_UI_CANCEL` – нажата кнопка ОТМЕНА.