ПРОГРАММИРОВАНИЕ В UNIGRAPHICS

UG/Open

«Любой и каждый может свести лошадь к водопою. Но если вы, научили ее плавать на спине - считайте, что добились определенных результатов!»

Критерий Хартли. «Законы Мерфи»

Каждый пользователь, освоив интерактивные средства моделирования, формирования сборочных узлов и оформления конструкторской документации, в процессе решения практических, порой весьма нетривиальных задач сталкивается с потребностью дополнить возможности системы новыми функциями для решения таких задач или автоматизации выполнения повторяющихся процедур. Очень часто требуется интегрировать Unigraphics с различными специализированными приложениями, предназначенными для проведения расчетов. При этом геометрия, сформированная в UG, может выступать в качестве исходной информации для выполнения анализа, или, наоборот, Unigraphics возьмет на себя функции отображения сгенерированной приложением геометрии в модельном пространстве.

МОДУЛИ UG/OPEN

Для решения подобных задач Unigraphics располагает достаточно развитыми возможностями модуля UG/Open API (Application Program Interface), реализованного на принципах открытой архитектуры и предоставляющего доступ к

объектам геометрической модели для программных приложений разработчиков или программ отдельных пользователей. UG/Open дает возможность программным способом создавать геометрические модели, получать информацию об объектах, формировать сборки, генерировать чертежную документацию и т.д. Практически все функциональные возможности Unigraphics, доступные пользователю при интерактивном взаимодействии с системой, реализуемы посредством функций API. Однако обратное утверждение неверно - существует ряд объектов (например, Custom Objects - Объекты пользователя), формирование которых возможно только программным способом.

Модуль UG/Open включает в себя следующие программные интерфейсы:

•**UG/Open API (User Function)** Реализует взаимодействие Unigraphics и программ пользователя, написанных на языке C. Заголовочные файлы (.h) соответствуют требованиям стандарта ANSI C и поддерживают разработку программ с использованием языка C++

В зависимости от способа построения программа пользователя может выполняться как внешнее (External) или как внутреннее (Internal) приложение. В первом случае программа запускается средствами операционной системы как независимое приложение или как процесс, порожденный Unigraphics. Так как внешнее приложение не имеет средств графического вывода, ему доступны функции вывода на печатающие устройства и формирования CGI-файла. Во втором случае программа может быть запущена только из текущей сессии Unigraphics. Она загружается в пространство процесса и может быть остановлена (выгружена) только соответствующими командами API, а все результаты работы программы отображаются в графическом окне Unigraphics.

• **UG/Open GRIP** Сокращение GRIP происходит от Graphics Interactive Programming (Язык интерактивного графического программирования). Это интерпретируемый язык, использующий инструкции, во многом сходные с такими некогда популярными языками, как BASIC или FORTRAN. Программы, реализованные на языке GRIP, имеют доступ к внутренним программам UG/Open API, В свою очередь, любая GRIP-программа может быть запущена из приложения UG/Open API.

• **UG/Open GRIP NC** Специализированное расширение языка GRIP для формирования управляющих программ движения инструмента станков с ЧПУ модуля UG/Manufacturing.

• **UG/Open MenuScript** Позволяет пользователям и разработчикам программного обеспечения посредством редактирования текстовых ASCII-файлов изменять меню Unigraphics и создавать собственные меню и панели инструментов, интегрированные с их собственными приложениями. Инструментарий MenuScript доступен также программным способом через функции UG/Open API, описанные в заголовочном файле uf_mb.h.

• **UG/Open UlStyler** Позволяет формировать диалоговые окна и панели Unigraphics.

Следует отметить, что система лицензирования Unigraphics разделяет права пользователя как на создание приложений UG/Open, так и на их запуск и исполнение (за соответствующими разъяснениями обратитесь к вашему поставщику Unigraphics).

Описание работы с модулями UG/Open GRIP NC, UG/Open MenuScript, UG/Open UlStyler выходит за рамки данной книги, а о написании программ с использованием языков C и GRIP мы поговорим подробно, С чего начать? В чем отличие UG/Open GRIP и UG/Open API и в каких ситуациях следует применять то или иное средство разработки собственных приложений?

Если нужно быстро, что называется, «на скорую руку» написать небольшое приложение, но вы не располагаете опытом программирования и самым большим вашим достижением была BASIC-программа из нескольких строк, выводящая на экран приветствие «Hello, World!», -начните с разработки GRIP-nporраммы. Для этого не потребуется привлекать какие-либо среды разработки типа Microsoft Visual C++: весь необходимый набор инструментов для компилирования и построения исполняемой GRIP-программы поставляется вместе с системой Unigraphics. При этом и сам исходный код приложения, и способ его создания будут абсолютно одинаковы как для платформы Windows NT/2000, так и для UNIX-станций Hewlett-Packard, IBM, Sun, SGI и др.

Однако следует помнить, что GRIP - интерпретируемый язык. Программы, написанные на нем, выполняются достаточно медленно и малопригодны для обработки больших объемов данных, для реализации итерационных расчетных процессов или для моделирования методом Монте-Карло, когда для получения достоверной картины имитируемого процесса требуется произвести огромное количество вычислительных операций. GRIP не располагает процедурами высвобождения используемой памяти, поэтому после запуска программы итерационные процессы постепенно снижают скорость работы.

В отличие от GRIP-программ приложения, реализованные на языках C или C++, подключаются к Unigraphics в виде динамически подгружаемых библиотек DLL (Dynamic Link Library), выполняются очень быстро и в процессе разработки предоставляют пользователю весь потенциал языка C для управления памятью и ресурсами. Конечно же, для создания приложений с использованием UG/Open API требуется опыт написания программ на языке C, навыки работы в среде разработки программ Microsoft Visual C++. Поэтому для создания серьезных приложений разумно поручить работу по реализации программы квалифицированному программисту, а постановку задачи - инженеру, хорошо понимающему суть решаемой проблемы. Достаточно же простые приложения сможет написать любой программист, когда-либо создававший программы, подобные той, что выводит на экран пресловутое «Hello, World!».

Для разработки приложений UG/Open++ на рабочих станциях под управлением Unix потребуется наличие соответствующих платформе компиляторов:

Платформа	Компилятор
IBM/AIX	Не поддерживается
Hewlett-Packard (HP)	ACC (A.01.12) или aCC (A.01.18)
Digital UNIX (DUX)	Не поддерживается
Silicon Graphics (SGI)	CC (7.2.1)
Sun	CC (4.2)
Windows/NT	Microsoft Visual C++ 6.0 (service pack 3)

UG/GRIP

Целью данного раздела не является детальное описание функций, используемых при разработке программного приложения к системе. В руководстве пользователя Unigraphics оно занимает не одну сотню страниц. По собственному опыту мы можем утверждать, что наиболее трудным в разработке самой первой, самой простой программы является понимание правил компилирования и построения исполняемого модуля, подключения приложения к Unigraphics и его запуска. Именно эти вопросы мы рассмотрим детально.

ПРОГРАММНАЯ ОБОЛОЧКА GRADE

Для создания программы, ее компиляции и формирования исполняемого модуля пользователю Unigraphics предлагается GRip Advanced Development Environment (GRADE) - интегрированная среда разработки. Искушенные программисты могут посетовать на небогатый, на первый взгляд, набор функций этой программной оболочки. Однако, независимо от типа рабочей станции (PC или Unix-машина) и операционной системы (Windows, IRIX, Solaris и т.д.), это окружение будет выглядеть абсолютно одинаково, а исходный код GRIP-программ, созданный на одной платформе, после перекомпиляции работоспособен на любой другой.

Следует отметить, что в системе Unigraphics отдельно предоставляется лицензия как на средства разработки (GRIP Development) GRIP-приложений, так и на запуск (GRIP Execute) исполняемых программ. Поэтому, прежде чем приступить к созданию собственной GRIP-программы, узнайте у вашего поставщика Unigraphics, располагаете ли вы лицензией на разработку и запуск GRIP-программ.

Запустить программную оболочку GRADE можно из меню рабочего стола Windows ***Start ~> Programs ~> Unigraphics ~> Unigraphics Tools ~> UG Open GRIP*** (см. рис. 7.1).

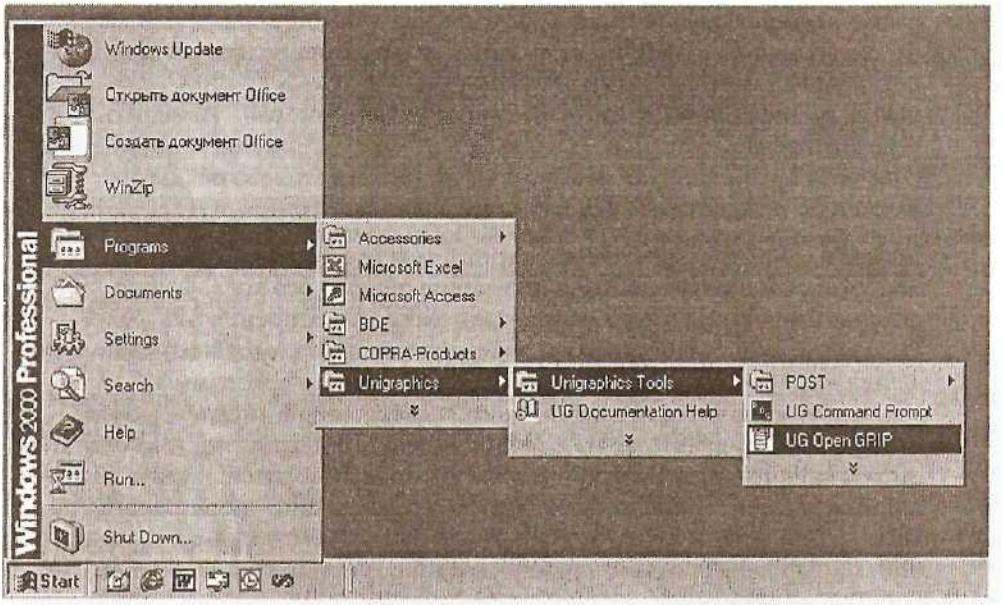


Рис. 7.1. Запуск программной оболочки GRAD

GRADE содержит 10 пунктов меню (см. рис. 7.2). Назначение некоторых из них мы рассмотрим подробно.

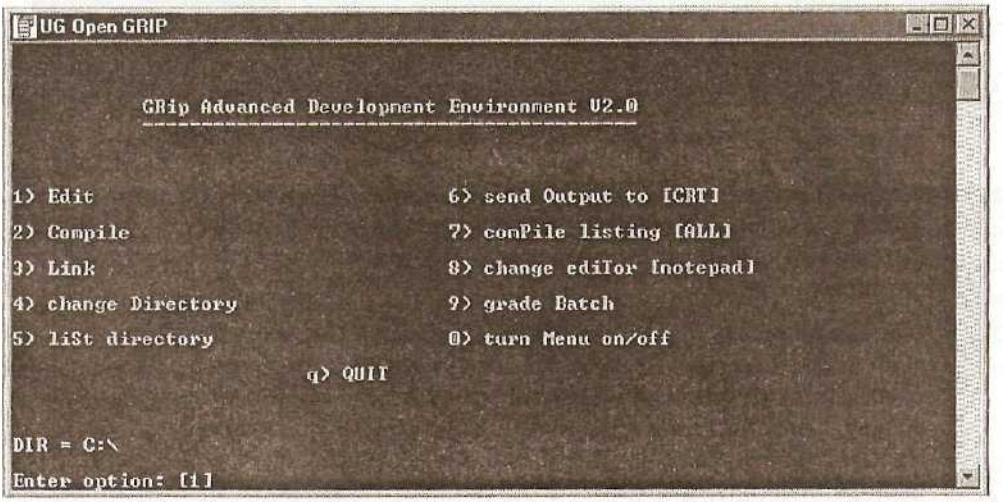


Рис. 7.2. Программная оболочка GRADE после запуска

• *Edit* Создание нового файла с исходным текстом GRIP-программы, вызов на редактирование существующего файла. В соответствии

с принятыми обозначениями для типов файлов Unigraphics файлы с исходным кодом GRIP-программ имеют расширение .grs, а для их редактирования в операционной системе Windows NT/2000 используется простейший текстовый редактор Notepad. Изменить тип текстового редактора можно с помощью 8-го пункта меню ***change editor.*** При этом путь к исполняемому файлу нового текстового редактора должен быть включен в системную переменную Windows PATH.

- ***Compile*** Компилирование исходного текста программы или подпрограмм на языке GRIP и получение объектного кода. Файлы, получаемые в результате компиляции, имеют расширение .gri. В процессе компиляции на экран выводится содержимое исходного файла и сообщения о возможных ошибках в тексте программы. Управлять количеством информации, выводимой на дисплей, можно с помощью 7-го пункта меню ***Compile listing,*** а 6-й пункт меню ***(Send output to)*** определяет устройство, на которое выводится информация - это может быть экран компьютера, принтер или текстовый файл. Если вы не располагаете лицензией на разработку GRIP-программ, то при попытке произвести компиляцию созданного приложения оболочка GRAD выведет следующее сообщение (см. рис. 7.3):

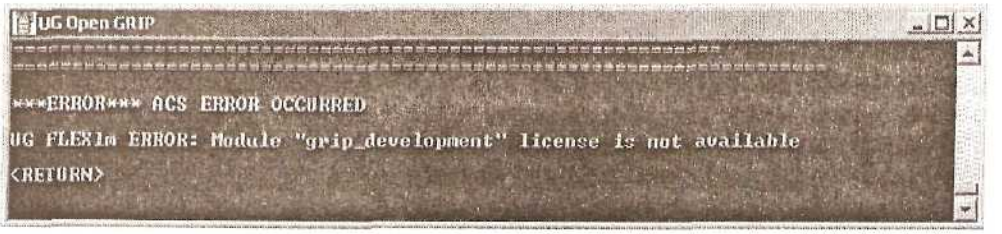


Рис. 7.3. Сообщение об ошибке: отсутствует лицензия на модуль

- ***Link*** Формирование исполняемого файла GRIP-программы, которая может состоять из одного модуля или включать несколько подпрограмм. Исполняемые файлы, готовые к запуску в сеансе Unigraphics, имеют расширение .grx.
- ***Change directory*** Этот пункт меню изменяет имя текущей папки, в которой размещены файлы с исходными текстами и в которой будут сохраняться результаты компилирования программ и исполняемые файлы. Соответственно, пункт меню List directory позволяет просмотреть содержимое текущей папки с применением фильтра по типу файлов (.grs, .gri, .grx).
- ***Turn menu on/off*** Для пользователей, достаточно «набивших руку» в использовании программной оболочки GRADE, может оказаться полезной отмена вывода перечня пунктов меню на экран с сохранением только строки ввода команд.

ИНСТРУКЦИИ ЯЗЫКА GRIP

Любая GRIP-программа должна содержать три обязательные части: объявления или определения, инструкции и завершение программы.

Объявления производятся с помощью четырех ключевых слов (ENTITY, STRING, NUMBER, DATA), которые определяют переменные и их исходные значения для использования в инструкциях программы. Эти объявления резервируют память под числовые значения, объекты, строковые переменные. Например, для определения строковой переменной длиной 30 символов необходимо использовать ключевое слово STRING:

```
STRING/STR(30)
```

Инструкции GRTP-программы записываются с помощью ключевых слов языка (например: LINE, CIRCLE) и чаще всего используются для создания и манипулирования графическими объектами, для вывода сообщений, управления файлами и т.п.

Следующий пример демонстрирует построение четырех линий в форме прямоугольника. Программа содержит объявления переменных (LN1, LN2, LN3, LN4), которым будут соответствовать создаваемые линии и инструкции для их построения. Эта программа создает один и тот же прямоугольник независимо от того, сколько раз она будет запущена, так как числовые значения координат вершин жестко определены в ее коде. Позже, при создании программы построения параллелограмма, мы введем дополнительное диалоговое окно для ввода числовых параметров геометрической фигуры.

```
ENTITY/LN1,LN2,LN3,LN4
LN1=LINE/0,0,0,2,0,0
LN2=LINE/2,0,0,2,2,0
LN3=LINE/2,2,0,0,2,0
LN4=LINE/0,2,0,0,0,0
```

Для корректного завершения каждая GRIP-программа должна* заканчиваться инструкцией HALT.

```
ENTITY/LN1,LN2,LN3,LN4
LN1=LINE/0,0,0,2,0,0
LN2-LINE/2,0,0,2,2,0
LN3=LINE/2,2,0,0,2,0
LN4=LINE/0,2,0,0,G,0
HALT
```

Любая последовательность символов в пределах строки после $$ игнорируется компилятором и может использоваться для комментирования и документирования программы.

По мере необходимости мы познакомимся и с другими инструкциями языка GRIP, которые потребуются для написания нашего приложения (для более детального знакомства с языком GRIP обратитесь к документации Unigraphics).

СОЗДАЕМ ПЕРВУЮ GRIP-ПРОГРАММУ

В качестве первого упражнения предлагаем вам написать небольшую программу на языке **GRIP, выполняющую** построение параллелограмма по введенным значениям длины основания, высоты, угла при вершине и угла поворота параллелограмма (см. рис. 7.4). Разумеется, эту геометрическую фигуру молено построить с помощью имеющихся интерактивных средств моделирования, но применение средств UG/Open GRIP для решения даже такой простой задачи дает заметный выигрыш во времени и сокращает число интерактивных действий.

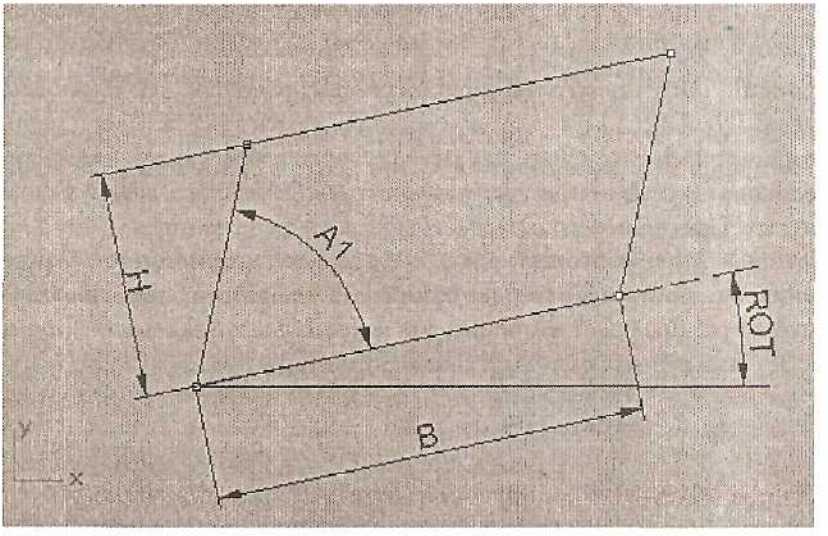


Рис. 7.4. Схема построения параллелограмма

Ниже приведен полный исходный текст программы PLOGRAM.GRS:

```
$$ Программа:   PLOGRAM.GRS
$$
$$ Описание: Программа построения параллелограмма
$$ в указанной точке по длине основания, высоте, углу
$$ при вершине
$$
$$ История разработки: 11-мая-2002
$$ Требуемые подпрограммы: (нет)
$$
$$ Компания: Consistent Software
$$ Адрес: Москва, Токмаков пер., 11
$$ Телефон: (095) 913-2222
```

```
$$ e-mail: jura@csoft.ru $$
$$ Операционная система: Windows 2000, Unigraphics v!8.0 $$

NUMBER/A1, B, H, RESP, ROT, X, Y, Z
ENTITY/L(9),P(5)
DATA/A1, 100.0, B, 120.0,H,50.0,ROT,20.0

BEG:GPOS/'Укажите точку вставки', X,Y, Z, RESP
JUMP/BEG:,ENDD:,,,,,RESP

PAR:PARAM/'Введите значения',              $ Ввод параметров
'Длина основания',B,'Высота',H,
'Угол при вершине', A1,'Угол поворота',ROT,RESP
      IF/A1*B*H,,,TST:
      MESSG/'Недопустимые параметры'
      JDMP/PAR:

TST:IF/90-ROT, ,NXT:,NXT:           $$ Test Rot Angle
MESSG/'Угол поворота слишком велик'
JUMP/PAR:

NXT:DRAW/OFF                        $$ Запрет отрисовки
      P(1)=POINT/X,Y,Z
      L(1)=LINE/P(1),ATANGL,A1+ROT
      P(2)=POINT/P{1),POLAR,B,ROT
      L(2)=LINE/P(1),P(2)
      L(3)=LINE/PARLEL,L(2) ,5,H
      P(3)=POINT/INTOF,L(1),L(3)    $$ Определение точки
                                    $$ пересечения двух линий
      L(4)=LINE/P(2),PARLEL,L(1)
      P(4)=POINT/IHTOF,L(3),L(4)
      DELETE/L(1..4)                $$ Удаление вспомогательных
                                    S $ линий
      L(5)=LINE/P(1),P(2)     $$ Отрисовка линий по 4-м точкам
      L(6)=LINE/P(2),P(4)
      L(7)=LINE/P(4),P(3)
      L(8)=LINE/P(3),P(1)
      DRAW/L{5..8)             $$ Отрисовка параллелограмма
      DELETE/P(1..4}          $$ Удаление вспомогательных точек
      JUMP/BEG:
ENDD:HALT
```

Рассмотрим более подробно алгоритм работы программы и операторы, выполняющие различные действия.

```
NUMBER/A1,B,H,RESP,ROT,X,У,Z
```

Следующие за ключевым словом **NUMBER** имена переменных соответствуют определенным числовым значениям, причем безразлично, является ли это число целым, вещественным, одинарной или двойной точности. В зависимости от присваиваемого значения переменная будет соответствовать тому или иному типу числовых данных. В нашем примере переменной A1 соответствует значение угла при вершине параллелограмма. B - это длина основания, H - высота параллелограмма, ROT - угол поворота параллелограмма, переменные X, Y, Z будут использованы в промежуточных вычислениях, а переменная RESP - целочисленная величина, которой будут присваиваться коды завершения операций.

```
ENTITY/L(9), P(5)
```

Ключевое слово **ENTITY** объявляет переменные, которые будут соответствовать геометрическим объектам модельного пространства Unigraphics. Так же, как и в случае с числовыми данными, объекту типа **ENTITY** может соответствовать точка, линия, кривая, поверхность, твердотельная модель. В нашей программе объявлены 9 объектов L (им будут соответствовать отрезки прямых линий) и 5 объектов P, которые определяют точки, участвующие во вспомогательных построениях.

Переменные типа NUMBER или ENTITY могут образовывать массивы, однако размерность их не может превышать 3.

Например, объявление NUMBER/NUM(3) определяет три числовых значения, которые образуют одномерный массив, и числовые значения могут быть присвоены элементам массива NUM(l), NUM(2) и NUM(3).

Строковые переменные объявляются оператором STRING.

Еще небольшое замечание. Все необъявленные переменные трактуются интерпретатором языка GRIP как «простые* (simple), и им могут быть присвоены только числовые значения. Попытка назначить соответствие необъявленной переменной какому-либо геометрическому объекту приведет к сообщению об ошибке в процессе компиляции. Пример программы (он не имеет отношения к нашей задаче построения параллелограмма) приведен ниже:

```
NUMBER/NUM(3)
SIZE=.25
NUM(l)=1.0625
NUM(2)=2.25
NUM(3)=1
HALT
```

Переменная SIZE в этом примере не объявляется, ей просто присваивается числовое значение.

DATA/A1,100.0,B,120.0,H,50.0,ROT,20.0

Первоначальные значения переменным, которые объявлены как **NUMBER** или **STRING,** могут быть присвоены оператором DATA. В нашем примере всем геометрическим размерам будущего прямоугольника назначаются произвольные величины, а чуть позже мы предложим пользователю ввести нужные значения в соответствующие поля диалогового окна. Если не произвести предварительную инициализацию переменных, диалоговое окно будет содержать их случайные значения. Кроме того, с помощью оператора DATA очень удобно производить начальные присвоения массивов.

Переменные объявлены, начальные значения присвоены - приступим к построению параллелограмма. Условимся, что все наши построения происходят в плоскости X-Y текущей системы координат, и прежде всего необходимо определить геометрическое положение вершины параллелограмма. Для этого молено было бы предложить диалоговое окно для ввода координат X и Y для вершины, но есть способ лучше. Язык GRIP предлагает воспользоваться вызовом стандартной процедуры Unigraphics для задания координат точки всеми доступными в интерактивном режиме способами (диалоговое окно Point Constructor) (см. рис. 7.5). Вызов осуществляется с помощью оператора GPOS, описание которого приведено ниже.

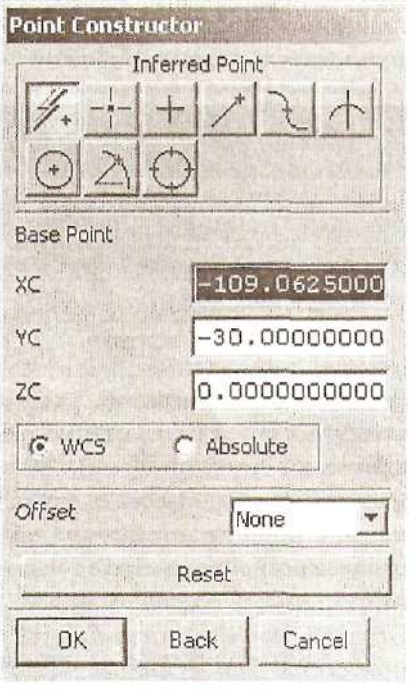


Рис. 7.5. Диалоговое окно Point Constructor

BEG:GPOS/'Укажите точку вставки', X, Y, Z, RESP

В нашем примере непосредственно оператору GPOS предшествует метка BEG, и, прежде чем продолжить описание оператора GPOS, скажем несколько слов об использовании меток в языке GRIP. Этот язык не обладает столь же мощными средствами структурирования программы, как, например, C или C++, и для организации условных и безусловных переходов внутри программы очень широко задействован механизм перехода к строке, обозначенной какой-либо меткой в зависимости от выполнения ряда условий. В качестве метки может выступать любая буквенно-цифровая комбинация из 6 символов, но в качестве первого символа обязательно должна стоять буква. У многих программистов считается дурным тоном использование инструкций безусловного перехода вида **goto** (в языке GRIP эта инструкция имеет вид JUMP/метка:), но в языке GRIP этот прием

достаточно часто применим. Он встретится и в нашей простейшей программе. Наверное, этого способа передачи управления в программе можно было бы избежать, но иногда для простоты реализации можно воспользоваться и им.

Итак, вернемся к строке, помеченной меткой BEG, и продолжим знакомство с оператором GPOS. За ключевым словом GPOS после символа «/» следует ряд обязательных параметров, первый из которых - заключенное в кавычки сообщение, выводимое в строке подсказки графического окна Unigraphics в момент активизации диалога Point Constructor. В нашем случае в качестве подсказки пользователь увидит сообщение «Укажите точку вставки» (см. рис. 7.6) и далее - подсказки уже от диалога Point Constructor для определения координат точки. Мы использовали строку на русском языке. При работе на платформе Windows NT/Windows 2000 с поддержкой русского языка в строке подсказки Unigraphics будет выведено сообщение на русском языке. Однако даже попытка скомпилировать на Unix-станциях программу, включающую кириллические символы, может закончиться неудачей (хотя многие разновидности операционной системы Unix поддерживают русский язык). Поэтому, при всем уважении к родному языку, советуем вам использовать для заголовков диалоговых окон, сообщений или подсказок строки на английском языке. Кроме того, руководство по языку GRIP настоятельно не рекомендует использовать символ «*» в заголовках, сообщениях и т.п.

Рис. 7.6. Подсказка об указании точки вставки

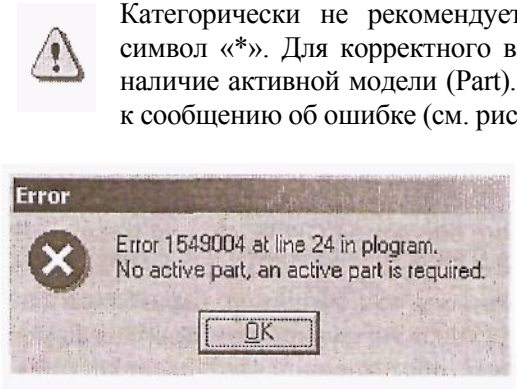


Рис. 7.7. Сообщение об отсутствии автивной модели Unigraphics

Категорически не рекомендуется использовать в тексте подсказки символ «*». Для корректного выполнения оператора GPOS требуется наличие активной модели (Part). Невыполнение этого условия приведет к сообщению об ошибке (см. рис. 7. 7).

Далее в операторе GPOS следуют ранее декларированные переменные X, Y, Z, которым будут присвоены числовые значения координат, определенных пользователем с помощью диалога Point Constructor. Обратите внимание: несмотря на то что мы строим плоскую геометрическую фигуру, координаты вершины определяются в трехмерном пространстве.

RESP	Значение
Кнопка Back	1
Кнопка Cancel	2
Кнопка OK	3
Не используется	4
Координаты определены	5

Последний параметр - целочисленная переменная RESP, которой, в зависимости от действий пользователя при завершении работы с диалогом Point Constructor, присваивается определенное значение (см. таблицу). Например, введя координаты точки, пользователь нажал кнопку OK - переменная RESP приобретает значение 3. Если же пользователь в качестве привязки выбрал, например, концевую точку существующей линии, диалог Point Constructor завершит работу с присвоением переменной RESP значения 5.

В зависимости от значения переменной RESP следующий оператор условного перехода в нашей программе направит выполнение программы по тому или иному пути. Рассмотрим его подробнее:

```
JUMP/BEG:,ENDD:, , , , ,RESP
```

В первую очередь обратите внимание на последний параметр этого оператора - целочисленную переменную RESP. Значение этой переменной определяет порядковый номер метки, которой будет передано дальнейшее управление программой. В нашей программе в результате выполнения предыдущего оператора GPOS переменная RESP может принимать целые значения от 1 до 5. Соответственно, в операторе JUMP, перед параметром RESP, следует указать 5 меток, которым и будет передаваться управление программой в зависимости от значения переменной RESP. Например, при нажатии кнопки Back в диалоге Point Constructor переменная RESP принимает значение 1 и оператор JUMP передаст управление программой строке с меткой BEG, которая еще раз предложит пользователю определить координаты вершины параллелограмма. Напротив, при нажатии кнопки Cancel (RESP = 2), завершение всей программы будет прервано, так как вторая метка в операторе JUMP отправляет программу на последний оператор HALT, который завершает работу любой GRIP-программы.

Если нам безразличны остальные значения переменной RESP, соответствующие этим значениям метки можно не указывать, заменив их парой запятых. При этом управление передается оператору, следующему за оператором JUMP. К тому лес результату приведет нулевое значение переменной RESP. В любом случае общее количество параметров должно соответствовать возможным принимаемым значениям последнего параметра оператора JUMP.

По правилам языка GRIP оператор условного перехода может содержать до 43 меток, которым может быть передано управление. Этого количества более чем достаточно, однако рекомендуем вам использовать не более пяти меток в одном операторе JUMP.

В качестве переменной RESP в операторе JUMP может выступать любое вычисляемое выражение, например X+Z. От результата вычисления берется целая часть, и управление передается метке с соответствующим порядковым номером в строке оператора JUMP. Однако пользоваться таким управлением следует осторожно.

Мы достаточно подробно рассмотрели оператор условного перехода. В языке GRIP он применяется несколько своеобразно. Переходим к следующему оператору в нашей программе:

```
PAR:PARAM/'Введите значения',                    $ Ввод параметров
'Длина основания',  B,  'Высота',  H,
'Угол при вершине',  A1,'Угол поворота',  ROT,  RESP
```

Сам оператор имеет ключевое слово PARAM, а назначение метки PAR мы поясним чуть позже. При выполнении этого оператора GRIP-программа предлагает пользователю ввести определенные параметры в диалоговом окне (см. рис, 7.8):

Рис. 7.8. Диалоговое окно для ввода параметров и строка подсказки

Первый из параметров оператора PARAM - это сообщение длиной до 40 символов, выводимое в строке состояния графического окна Unigraphics. Далее парами следуют названия предлагаемых полей ввода числовых значений и имена переменных, которым эти значения будут присвоены. GRIP позволяет создавать диалоговые окна, содержащие до 14 полей ввода, названия которых могут включать до 15 символов. Оператор PARAM может оказаться достаточно длинным, и для его записи в несколько строк можно воспользоваться символом "$".

Обратите внимание: начальные значения переменных в полях ввода соответствуют значениям, присвоенным оператором DATA. В рассматриваемом примере все числовые значения имеют вещественный тип, и в поле ввода они

отображаются с десятичной точкой и дробной частью. Оператор PARAM позволяет определить вводимый тип данных как целое с помощью специального ключевого слова INT. В этом случае в поле ввода число будет отображаться без десятичного разделителя. Пример записи оператора выглядит следующим образом:

```
PARAM/'Введите число повторений', 'Количество', INT, Б, RESP
```

Категорически не рекомендуется использовать в названиях полей ввода и в строке подсказки символ «*». Для корректного выполнения оператора PARAM требуется наличие активной модели (Part). Невыполнение этого условия приведет к сообщению об ошибке.

RESP	Значение
Back	1
Cancel	2
OK	3

Выводимое диалоговое окно имеет три кнопки; Back, Cancel и OK. В зависимости от того, какую кнопку нажмет пользователь, переменной RESP присваиваются значения (см. таблицу). Честно говоря, переменная RESP может принимать значение RESP = 4, но рассмотрение таких случаев выходит за рамки нашего примера (подробнее о работе оператора **PARAM** см. в документации по UG/Open). В данном случае нас не интересует, какую именно кнопку нажмет пользователь, изменит ли он значения геометрических параметров, - мы будем проверять именно корректность введенных данных. Естественно, что длина стороны параллелограмма, его высота и угол при вершине не могут иметь нулевые или отрицательные значения, поэтому используем оператор условного перехода другого типа, с классическим ключевым словом IF:

```
IF/A1*B*H,,,TST:
```

В общем виде оператор IF имеет следующую форму записи:

```
IF/[Выражение], label1, label2, iabel3
```

В зависимости от значения вычисляемого выражения управление программой будет передано оператору с соответствующей меткой. Если результат вычисления отрицательный - управление передается оператору с меткой label1, если результат равен нулю - выполняется оператор с меткой 1aЬe12, а в случае положительного значения результата вычисления выражения происходит переход к строке с меткой 1aЬe13. Если какая-либо из трех меток в записи оператора отсутствует, управление передается оператору, следующему за оператором **IF**.

В нашей программе в качестве вычисляемого выражения оператора IF представлено произведение переменных A1, B, H (угол при вершине, длина стороны и высота), и в случае ввода отрицательных или нулевых значений для какого-либо геометрического параметра результат вычисления выражения будет нулевым или отрицательным. А так как первая и вторая метки в нашем операторе IF

отсутствуют, происходит переход к следующему оператору, который выводит сообщение о неверно введенных значениях(см. рис. 7.9):

MESSG/'Недопустимые параметры'

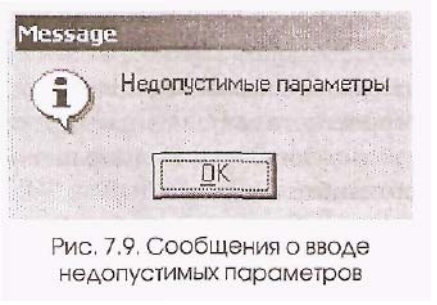


Рис. 7.9. Сообщения о вводе недопустимых параметров

Сообщения в GRIP выводятся в модальное окно. Это означает, что программа приостанавливает свое выполнение до закрытия окна. В нашем примере после закрытия окна следует безусловный переход к оператору, помеченному меткой PAR. Пользователю предлагается исправить неверно введенные параметры.

Если же введенные параметры - положительные величины, переходим к оператору с меткой TST. Здесь происходит проверка введенного значения угла поворота. Предлагаем читателю самостоятельно разобрать логику работы этого участка программы. Мы же перейдем сразу к оператору с меткой NXT:

NXT:**DRAW**/OFF

Действие оператора DRAW с параметром OFF очень простое. Начиная с этого момента запрещен вывод любой геометрической информации в графическое окно Unigraphics, чтобы скрыть промежуточные, вспомогательные построения.

А далее начинается самое интересное: построение геометрических примитивов - вспомогательных точек, линий!

P (1) =**POINT**/X,Y,Z

В разделе объявлений программы был объявлен массив из 5-ти точек. Приведенный выше оператор производит построение первой из них с координатами X, Y, Z, которые были определены с помощью оператора **GPOS.** Это самый простой способ построения точки с известными координатами. Средства языка GRIP позволяют программным образом реализовать практически все способы интерактивного построения точек, часть которых представлена в таблице:

Оператор	Описание
POINT/X,Y,Z	Построение точки по известным координатам X, Y, Z.
POINT/CENTER, circle	Построение точки в центре существующей окружности circle.
POINT/INTOF, obj1, obj2	Построение точки пересечения двух объектов. В качестве первого объекта obj1 должна выступать кривая, а второй объект obj2 может быть кривой, плоскостью, гранью твердого тела (подробнее о параметрах см. в документации UG/Open).

Оператор	Описание
POINT/point, VECT,line,dist	Построение точки на заданном расстоянии dist от существующей точки point в направлении вектора, параллельного существующей линии line.
POINT/circle, ATANGL,angle	Построение точки по заданному угловому положению angle на окружности circle.
POINT/point, DELTA, dx,dy,dz	Построение точки в виде смещения от существующей . точки на заданные расстояния по X, Y, Z.
POINT/ENDOF, "PMOD3",obj	Построение концевой точки линии, кривой.
POINT/point, POLAR, dist.ansrle	Построение точки в виде смещения от существующей точки в полярных координатах (но углу и расстоянию).

```
L(1)-LINE/P(1),ATANGL,A1+ROT
```

От первой построенной точки - вершины будущего параллелограмма - проводим линию под углом, величина которого равна сумме угла при вершине и угла поворота параллелограмма. Обратите внимание: длина этой линии нами не определяется; в данном случае объект L(l) -луч, имеющий начало в точке P(1) и уходящий в бесконечность (конечно же, в пределах модельного пространства Unigraphics!) (см. рис. 7.10). Это одна из причин, по которой полезно временно отключить вывод геометрических объектов на экран.

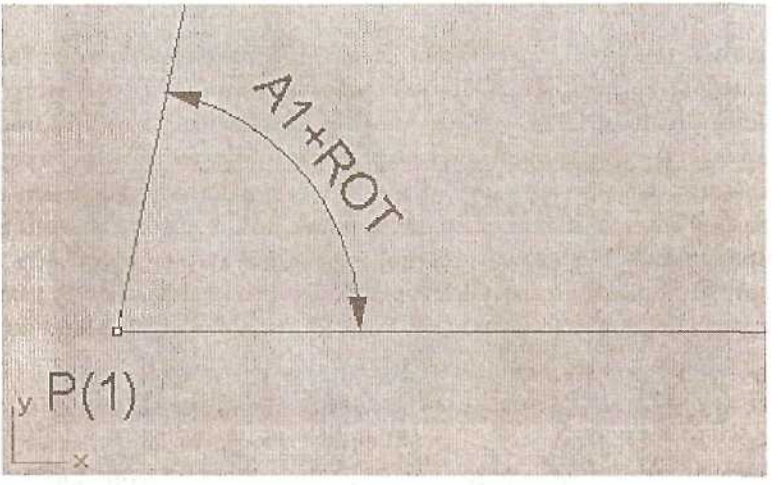


Рис. 7.10. Определение луча, проходящего через заданную точку, под определенным углом

```
P(2)=POINT/P(1),POLAR,B,ROT
```

Вторую точку P(2) построим, применяя смещение от вершины в полярных координатах на угол поворота параллелограмма и на расстояние, равное длине основания (см. рис. 7.11).

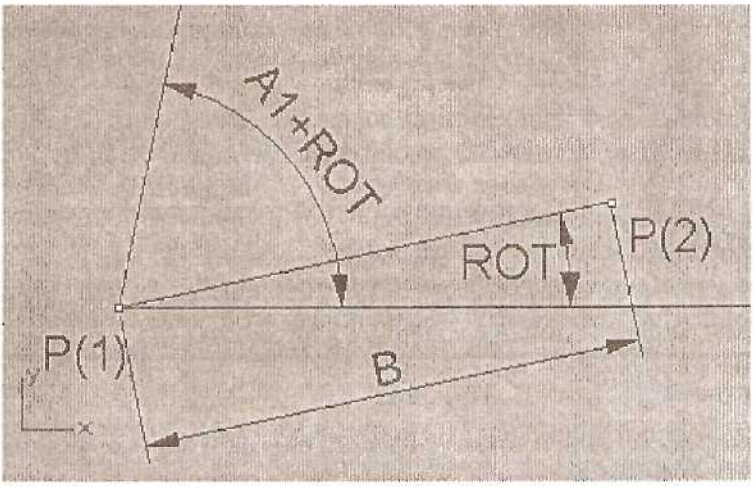


Рис. 7.11. Задание основания параллелограмма

```
L(2)=LINE/P(1), P(2)
```

Строим линию L(2), соединяющую точки P(1) и P(2).

```
L(3)=LINE/PARLEL,L(2) ,5,H
```

Проводим линию L(3) параллельно линии L(2) на расстоянии H (высота параллелограмма) (см. рис. 7.12). Обратите внимание на третий параметр оператора с ключевым словом LINE - целое число 5. Дело в том, что при построении линии, параллельной заданной на определенном расстоянии, необходимо указать, с какой стороны провести эту линию. Чтобы не приостанавливать выполнение программы и не спрашивать пользователя о том, с какой стороны провести линию (принципиально это возможно, но мы же хотим построить параллелограмм автоматически!), язык GRIP предлагает использование так называемых пространственных модификаторов и соответствующих им числовых кодов:

```
XSMALL - 1 - PMOD2/PMOD3        XLARGE - 4 - PMOD2/PMOD3
YSMALL - 2 - PMOD2/PMOD3        YLARGE - 5 - PMOD2/PMOD3
ZSMALL - 3 - PMOD3              2LARGE - 6 - PMOD3
```

В нашем примере мы использовали модификатор YLARGE (код 5), проще говоря, предлагаем провести параллельную линию в сторону больших координат Y относительно линии L(2),

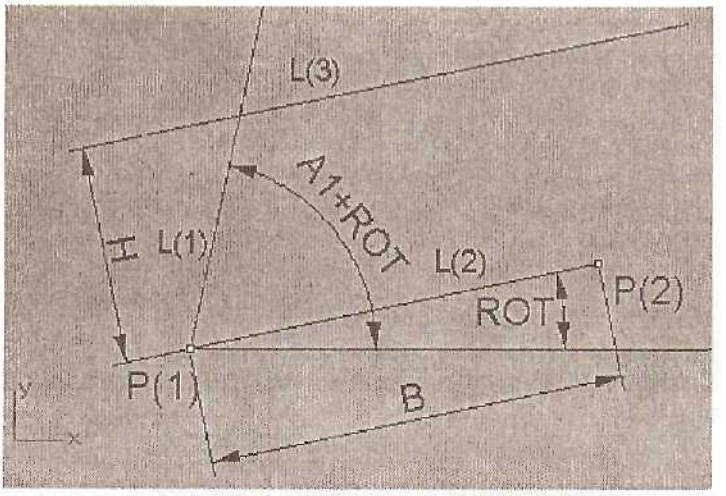


Рис. 7.12. Задание стороны, параллельной основанию

```
P(3)=POINT/INTOF,L(1) , L ( 3 )
L(4)=LINE/P(2),PARLEL,L(1)
```

Находим точку пересечения линий L(l) и L(3) - точку P(3) и проводим через точку P(2) линию L(4), параллельную линии L(1) (см. рис. 7.13).

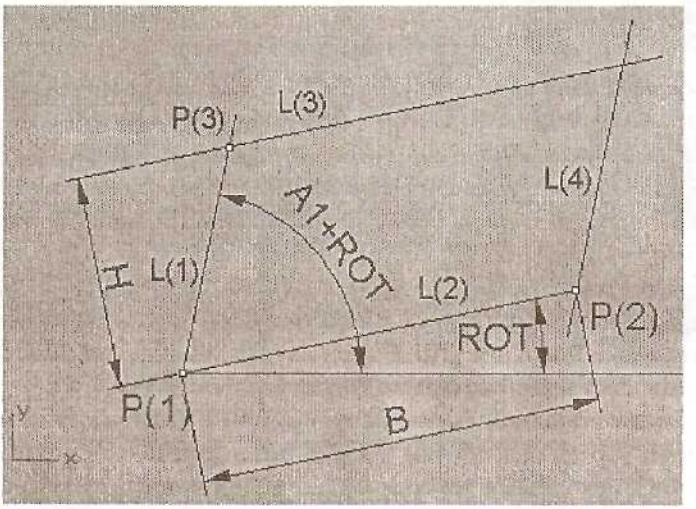


Рис. 7.13. L(4) – четвертая сторона параллелограмма

```
P(4)=POINT/INTOF,L(3),Lf4)
DELETE/L(1..4)
```

```
L(5)=LINE/P(1),P(2)
L(6)=LINE/P(2) ,P(4)
L(7)=LINE/P(4) ,P(3)
L(8)=LINE/P(3),P(1)
```

Далее находим точку пересечения линий L(3) и L(4) - точку P(4). Удаляем вспомогательные построения - линии L(l), L(2), L(3), L(4) - и строим «чистовые» линии (стороны нашего параллелограмма) L(5), L(6), L(7) и L(8). Обратите внимание на способ записи в операторе DELETE списка объектов: указаны первый и последний элементы списка, которые разделены двумя точками.

```
DRAW/L(5. .8)
DELETE/P(1. .4)
JUMP/BEG:
```

Мы на финишной прямой - наконец-то выводим изображения построенных линий, удаляем вспомогательные точки и любезно предлагаем пользователю построить еще один параллелограмм, осуществляя переход к оператору определения вершины GPOS.

Конечно же, пока мы только шаг за шагом «проиграли» алгоритм выполнения программы и познакомились с некоторыми операторами языка GRIP.

ПОСТРОЕНИЕ ИСПОЛНЯЕМОГО GRIP-ПРИЛОЖЕНИЯ

Теперь поговорим о построении исполняемого модуля программы. Для этого снова запустим программную оболочку GRADE и с помощью 4-го пункта меню ***(change Directory)*** прежде всего изменим путь к папке, в которой сохранен исходный текст нашей программы PLOGRAM.GRS (в именах папок и файлов лучше избегать русскоязычных названий или имен, содержащих пробелы). Убедитесь в том, что программа существует в этой папке. Для этого воспользуйтесь 5-м пунктом меню ***(list directory)***. Оболочка GRADE настроена по умолчанию таким образом, что она выводит на экран список всех файлов, имеющих расширение .gr*. Таким образом, в этот перечень попадают исходные тексты с расширением .grs, и скомпилированные объектные модули с расширением .gri, и исполняемые программы, имеющие расширение .grx.

Предположим, вы сохранили исходный текст программы в файле с названием plogram.grs на диске D в папке grip. В этом случае команда ***list directory*** выведет на экран список, состоящий из одного файла (см. рис. 7.14).

Возвращаемся к основному меню (клавиша ENTER) и приступаем к компиляции программы. Среда разработки GRADE позволяет компилировать сразу несколько файлов с исходными текстами. Наш проект состоит из одного файла, и после выбора 2-го пункта меню ***(Compile)*** мы указываем имя файла plogram.grs (расширение .grs в имени файла можно опустить).

Компилятор анализирует каждую строчку программы и формирует достаточно обширный листинг, содержащий список всех переменных, меток и возможных ошибок. Если вы набирали исходный код программы вручную, ошибки вполне

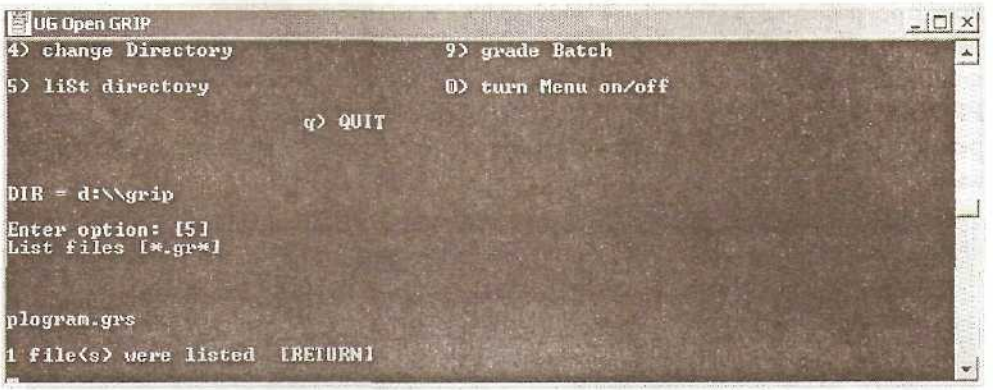


Рис. 7.14. Текущее сообщение в окне

вероятны. В этом случае компилятор метит в листинге строку с неверной записью символом «*». Если программа скомпилирована без ошибок, выводится сообщение «N GRIP PROGRAM COMPILED WITHOUT ERROR» (см. рис. 7.15). В результате в папке появляется файл со специальным объектным кодом, имеющий то же имя, что и файл с исходным текстом, и расширение .gri. С помощью 5-го пункта меню его можно увидеть в перечне файлов.

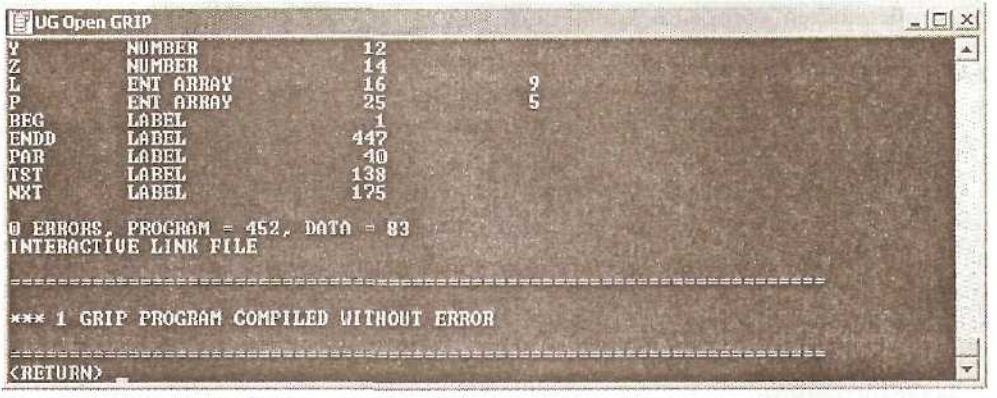


Рис. 7.15. Программа выполнена без ошибок

Третий, заключительный этап - построение исполняемого модуля из одного или нескольких объектных файлов. На сленге программистов этот процесс называется «линкированием», «линкованием» (от англ. Link). В нашем примере основная программа не включает подпрограмм, весь исходный текст содержится в одном файле, и в результате компиляции сформирован один файл с объектным кодом. Для линкования выбираем 3-й пункт меню ***(Link)*** оболочки GRADE и указываем имя файла plogram.gri. Выводимый па экран листинг содержит перечень

всех программных модулей, включаемых в исполняемое GRIP-приложение, сообщения об ошибках и другую полезную информацию. В случае успешного завершения процесса линковки, что подтверждается сообщением вида «N GRIP PROGRAM LINKED WITHOUT ERROR», создается исполняемый файл с расширением .grx. Надеемся, ваши труды увенчались успехом и исполняемый модуль с именем plogram.grx успешно сформирован (см. рис. 7.16).

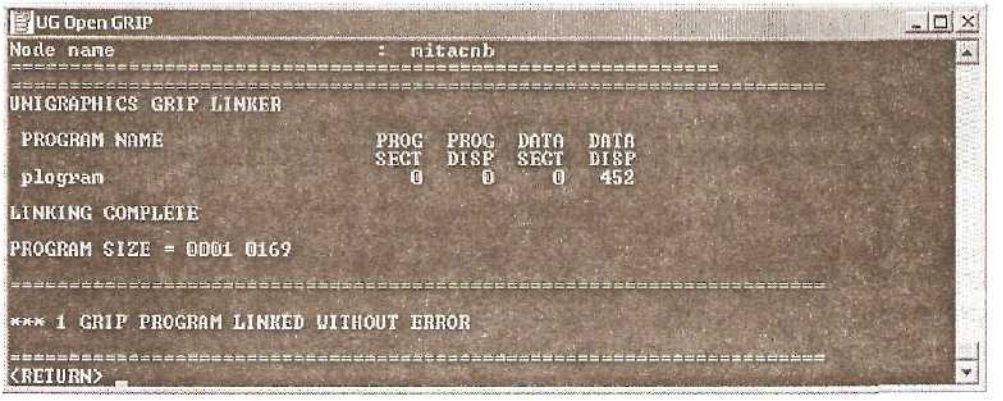


Рис. 7.16. Сообщение об успешном завершении процесса

Формирование исполняемого модуля из нескольких объектных модулей имеет ряд особенностей, которые подробно изложены в разделе документации UG/Open.

ЗАПУСК GRIP-ПРИЛОЖЕНИЯ

Как запустить созданную программу в сеансе Unigraphics? Для этого запустите Unigraphics, создайте новый или откройте существующий Part-файл и выберите в меню ***File*** раздел ***Execute UG/Open.*** Далее выберите запуск GRIP-программы (см. рис. 7.18) либо в обычном режиме (пункт ***Grip...,*** ему соответствует комбинация «горячих клавиш» *Ctrl+G),* либо в режиме отладки (пункт ***Debug Grip...,*** комбинация «горячих клавиш» *Ctrl+Shift+G).* При любом способе запуска будет выведено стандартное окно Windows для выбора исполняемого файла с раширением .grx. Выберите программу plogram.grx - и стройте параллелограмм за параллелограммом!

Рис. 7.17. Сообщение об отсутствии лицензии на выполнение GRIP-программ

Запуск GRIP-приложения невозможен, если вы не располагаете лицензией на выполнение GRIP-программ (Grip Execute). При отсутствии лицензии на экран будет выведено примерно такое сообщение (см. рис. 7.17):

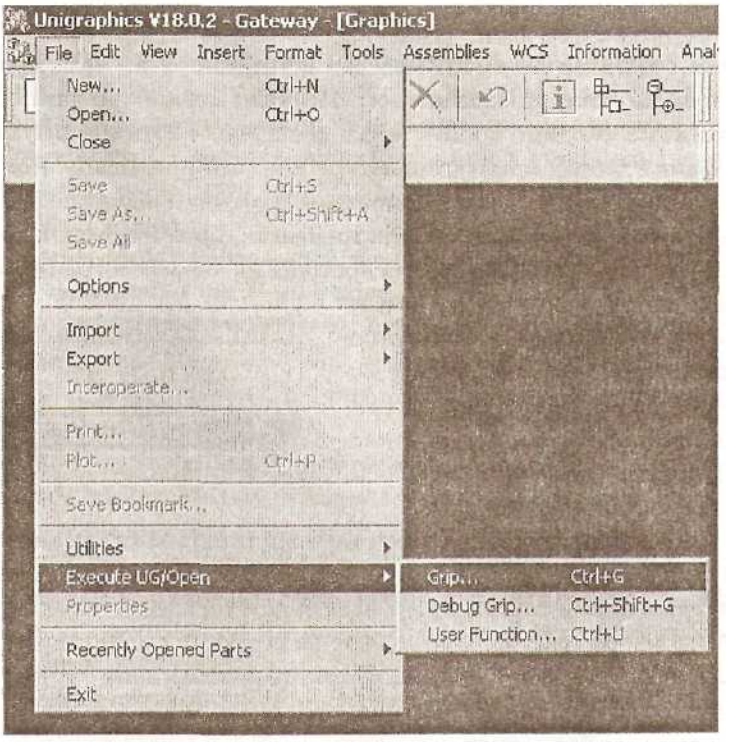


Рис. 7.18. Запуск GRIP-программы

Рис. 7.19. Отладка GRIP-программ

При создании сложных, состоящих из нескольких модулей, **GRIP**-приложений может оказаться полезным запуск приложения в режиме отладки - **Debug Grip** (см. рис. 7.19). После выбора исполняемой программы пользователю будет предложено окно с командами отладки. В этом случае возможны пошаговый режим выполнения, получение информации о текущем состоянии переменных, включение точек приостановки программы (breakpoints), аварийное прерывание программы и другие полезные функции. Те программисты, которым потребуется режим отладки созданных ими **GRIP**-программ, без труда сумеют разобраться в его возможностях.

СОЗДАЕМ ПАНЕЛЬ ИНСТРУМЕНТОВ ДЛЯ GRIP-ПРОГРАММ

Надеемся, созданная вами **GRIP**-программа успешно заработала, прошли первые восторги, и им на смену пришел вопрос: «Неужели каждый раз для построения параллелограмма необходимо пройти путь в меню ***File ~> Execute UG/Open ~> Grip,*** найти grx-файл в дебрях моего компьютера!? За что боролись?!». Комбинация горячих клавиш *Ctrl+G* мало спасает положение, поэтому предлагаем рассмотреть вопрос создания собственной панели инструментов, одна из кнопок которой будет активизировать GRIP-приложение для построения параллелограмма. Создаваемая нами панель инструментов будет выглядеть наподобие имеющихся в Unigraphics стандартных панелей (см. рис. 7.20) и может включать любое количество кнопок, каждая из которых активизирует пункт меню Unigraphics, программу UG/Open, GRIP-программу или макрос.

Рис. 7.20. Пример панели инструментов

Любая панель инструментов определяется в Unigraphics как внешний текстовый файл с расширением (.tbr), который содержит раздел описания заголовка панели инструментов и собственно раздел описания кнопок панели.

Создадим в любом текстовом редакторе файл grip.tbr следующего содержания:

```
!================================================
!   Панель инструментов для запуска GRIP-программ
!   Название файла:  grip.tbr
!

TITLE   GRIP-программы

VERSION  170.

BUTTON UG GRIP PLOGRAM LABEL  Построение параллелограмма
BITMAP D: \grip\bitir.aps\plogram.bmp ACTION
D:\grip\plogram.grx

SEPARATOR

BUTTON UG FILE RUN GRIP
LABEL  Запуск Grip программы
BITMAP D:\grip\bitmaps\grip.bmp

BUTTON UG FILE RUN GRIP DEBUG
LABEL  Запуск Debug Grip
BITMAP D:\grip\bitmaps\debug_grip.bmp

!===========     Конец файла    ==============
```

Панель инструментов, создаваемая файлом grip.tbr, имеет заголовок «GRIP-nporраммы» и содержит две группы кнопок, разделенные сепаратором. Первая группа включает всего одну кнопку, которой в качестве выполняемого действия определим запуск созданной нами программы построения параллелограмма plogram.grx. Во вторую группу входят две кнопки, одна из которых запускает выбранную пользователем GRIP-программу, а вторая производит те же действия в режиме отладки.

Каждая кнопка имеет уникальный идентификатор BUTTON, определение всплывающей подсказки LABEL (она появляется в момент наведения указателя мыши на кнопку панели инструментов), определение BITMAP для изображения кнопки и описание выполняемого этой кнопкой действия ACTION. Для первой кнопки в качестве выполняемого действия определим запуск скомпилированного GRIP-приложения plogram.grx

Поговорим несколько подробнее об изображении BITMAP для кнопок меню. Казалось бы, все просто: после определения BITMAP указан путь к bmp-файлу, который и будет выводиться в качестве изображения кнопки. Но Unigraphics позволяет отображать как стандартные панели инструментов, так и созданные пользователем панели с кнопками размером 16x16, 24x24, 32x32 и 48x48 пикселей, причем как с цветными, так и монохромными. Для каждой кнопки должны быть представлены 4 растровых файла - монохромный 16x16, цветной 16x16, монохромный 24x24 и цветной 24x24. Изображения кнопок 32x32 и 48x48 формируются в системе Unigraphics автоматически.

По причине вышеизложенного путь в определении BITMAP указывает путь к группе растровых bmp-файлов. В нашем примере определение plogram.bmp указывает на 4 файла:

plogram_lm.bmp	24x24, монохромный (lm - Large, Monochrome)
plogram_lc.bmp	24x24, цветной (Ic - Large, Color)
plogram__sm.bmp	16x16, монохромный (sm - Small, Monochrome)
plogram_sc.bmp	16x16, цветной (sc- Small, Color)

Создать bmp-файлы можно с помощью растрового редактора Paint (см. рис. 7.21) из стандартного набора инструментов Windows или любого другого (Adobe PhotoShop, CorelDraw и т.п.). Ниже приведен пример создания изображения цветной кнопки 24x24 plogram_lc.bmp с использованием редактора Paint. Не судите строго художественные способности авторов. По замыслу эта фигура должна изображать параллелограмм. Пространство 24x24 (тем более 16x16) пикселя несколько сдерживает полет фантазии, и хорошо нарисованную пиктограмму можно рассматривать как произведение художественной миниатюры!

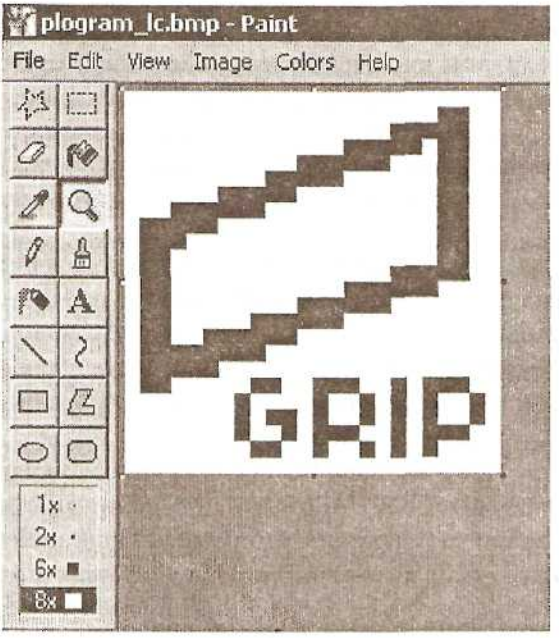


Рис. 7.21. Использование графического редактора Paint для рисования пиктограмм

Достаточно нарисовать цветные изображения кнопок 16x16 и 24x24, а затем средствами графического редактора создать их монохромные копии.

Итак, для трех кнопок меню необходимо создать 12 bmp-файлов, имена которых формируются по описанным выше правилам. В результате содержимое папки bitmaps выглядит так (см. рис. 7.22).

Несколько слов об уникальном для каждой кнопки идентификаторе BUTTON. Если для первой кнопки, запускающей наше собственное приложение, его молено определить абсолютно произвольно, то для кнопок, которые дублируют стандартные

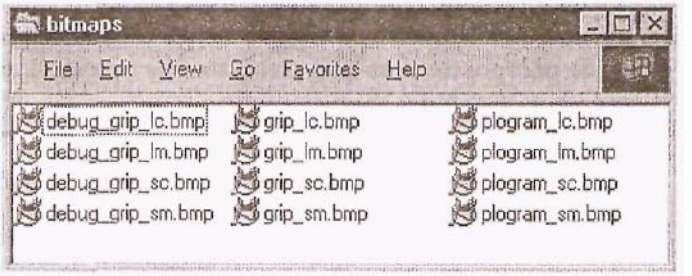


Рис. 7.22. Необходимый набор пиктограмм

пункты меню ***(File ~> Execute UG/Open ~> Grip...*** или ***Debug Grip...)***, идентификатор BUTTON должен соответствовать вполне определенному значению. Как узнать это значение? Для этого в меню Unigraphics выберите пункт ***Information ~> Custom Menuhar ~> Report Tool...***, укажите, что желаете получить полную информацию (пункт Complete Menubar) по основному меню Unigraphics (UG_GATEWAY_ MAIN_MENUBAR) и нажмите кнопку «ОК» (см. рис. 7.23). Полный список всех разделов меню с идентификаторами BUTTON (Item) будет выведен в информационное окно (см. рис. 7.24). Список будет достаточно длинным, и для того, чтобы найти требуемый пункт меню, придется воспользоваться средствами поиска. В нашем примере интерес представляют строки с пунктами меню «Grip...» и «Debug Grip...».

Из полученного списка выбираем нужные пункты меню (Item) и их идентификаторы (например, UG_FILE_RUN_GRIP) вставляем как соответствующие

Рис. 7.23. Получение информации

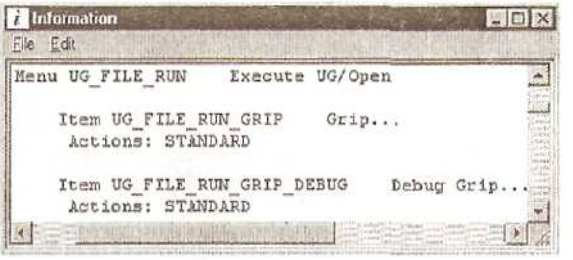


Рис. 7.24. Информация об идентификаторах меню

идентификаторы BUTTON в файл панели инструментов grip.tbr. Обратите внимание, что при создании кнопки панели инструментов, которая дублирует пункт основного меню и вызывает стандартную процедуру Unigraphics, выполняемое действие ACTION не указывается.

Окончательно панель инструментов для запуска GRIP-программ будет выглядеть следующим образом (см. рис. 7.25):

Рис. 7.25. Вид панели инструментов для запуска GRIP-программ

Мы сформировали файл нашей собственной панели инструментов. Конечно же, вы можете добавить в него и другие кнопки, создать вложенные, многоуровневые меню (см. в документации UG/CAD раздел Gateway/Toolbars) и назначить им соответствующее действие.

ЗАГРУЗКА ПАНЕЛИ ИНСТРУМЕНТОВ

Загрузка и активизация созданных пользователем панелей инструментов может производиться двумя способами.

1. В меню Unigraphics выберите пункт ***View*** ~> ***Toolbars*** ~> ***Customize...*** (см. рис. 7.26).

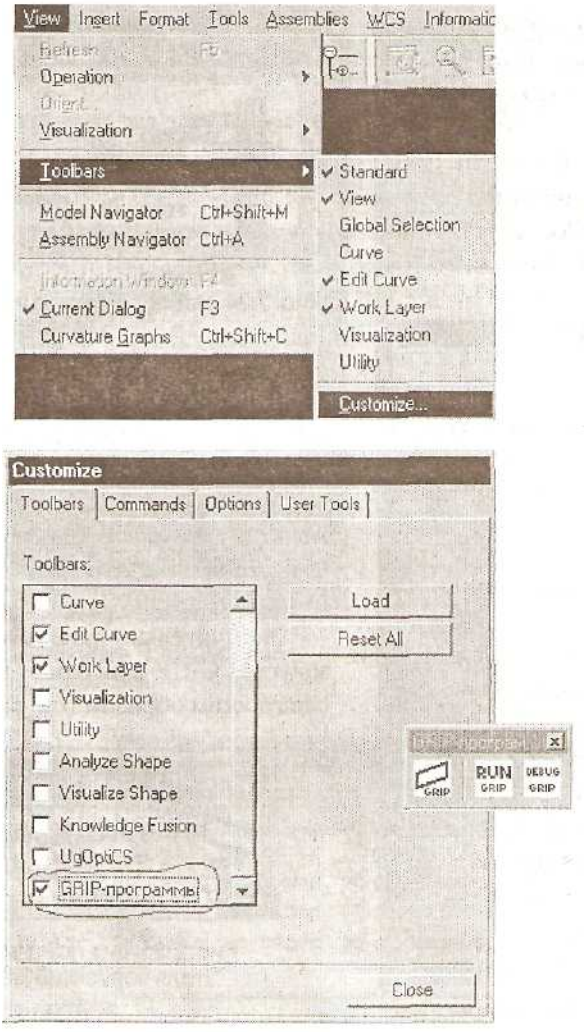


Рис. 7.26. Размещение панели инструментов

В предложенном диалоговом окне нажмите кнопку Load и выберите загружаемый файл панели инструментов (grip.tbr). В списке всех имеющихся в системе активных и скрытых панелей инструментов появится название новой подгружаемой панели (в нашем примере - «GRIP-программы»). При первой загрузке панель автоматически появляется на экране, в дальнейшем (в пределах сеанса работы) она скрывается или отображается вновь с помощью флажка, расположенного слева от ее названия. Размещение панели инструментов в пределах окна Unigraphics подчиняется законам Windows, ее положение запоминается в системе, однако при следующем запуске Unigraphics процесс загрузки созданной пользователем панели инструментов придется повторить.

2. Для того чтобы пользовательская панель инструментов автоматически загружалась при каждом старте Unigraphics в файл custom_dirs.dat, находящийся в каталоге $UGII_BASE_DIR$\ugii\menus, включите путь к каталогу, в котором располагается папка sturtup, в которой, в свою очередь, сохранен файл с пользовательской панелью инструментов (grip.tbr). Обратите внимание: *необходимо указать путь к папке startup,* а не к файлу grip.tbr. Соответственно, если в папке startup находится несколько файлов (.tbr), все панели инструментов, определенные в этих файлах, будут загружены. Ниже приведен фрагмент файла custom_dirs.dat,

```
#   Customers should feel free to edit this file.
#
##########################################################
#
# Customer modifications can follow on here
#
D:\grip
######################### Конец файла ##################
```

Строго говоря, создание собственных панелей инструментов не относится к теме программирования на языке GRIP и вообще не входит в раздел UG/Open, но для рассказа о правилах построения пользовательских панелей инструментов этот раздел книги подходит как нельзя лучше. Позже мы добавим в эту панель кнопку для запуска приложения User Function.

Если вам удалось повторить примеры, рассмотренные в книге, или заработала ваша первая, самостоятельно разработанная GRIP-программа - тогда знакомство с основами этого простого, но достаточно гибкого и мощного языка для создания собственных графических приложений можно считать состоявшимся.

UG/Open User Function

ОБЩИЕ ПОЛОЖЕНИЯ

UG/Open API (User Function) - это набор библиотек и подпрограмм, позволяющих внешнему приложению получить доступ к объектам модели Unigraphics. UG/Open API позволяет программным способом моделировать детали и сборки, выпускать чертежи, формировать модели по расчетам приложения. Практически все интерактивные функции Unigraphics доступны посредством UG/Open API, однако имеется целый класс объектов, формирование которых возможно только программным методом.

В зависимости от способа построения программное приложение, созданное пользователем, может функционировать как внешнее или внутреннее.

Внешнее (или независимое) приложение запускается как самостоятельный (stand alone) процесс средствами операционной системы или как процесс, порождаемый Unigraphics. Так как внешнее приложение не имеет средств отображения графической информации, единственно доступные в этом случае средства вывода - формирование CGM (Computer Graphic Metafile), файла, который затем можно просматривать средствами Unigraphics, или непосредственный вывод информации на внешнее устройство (принтер, плоттер). Главная отличительная особенность внешних приложений заключается в том, что для их запуска не требуется запуск собственно Unigraphics, но в то же время внешнее приложение имеет возможность открывать существующие модели Unigraphics и оперировать с содержащейся в них информацией.

Внутреннее (или интегрированное) приложение может быть запущено только из активной сессии Unigraphics и загружается в основное пространство процесса Unigraphics. Будучи однажды загруженным в процесс Unigraphics, приложение резидентно остается в нем либо до прекращения сессии Unigraphics, либо до принудительной выгрузки из системы. Внутренние приложения, как правило, имеют меньший размер и большую скорость выполнения по сравнению с внешними приложениями. Кроме того, результат работы внутреннего приложения отображается в графическом окне Unigraphics.

UG/Open API имеет интерфейс языка C (стандарт ANSI); заголовочные .h-файлы поддерживают язык C++.

СОЗДАЕМ ПРОЕКТ UG/OPEN

Для создания пользовательского приложения UG/Open необходимо наличие установленной на компьютере среды разработки приложений Microsoft Visual Studio C++ 6.0. В общем случае внутреннее приложение UG/Open представляет собой построенную по определенным правилам динамическую библиотеку DLL, подгружаемую к процессу Unigraphics, а внешнее приложение UG/Open - обычное 32-разрядное EXE-приложение Windows.

Для того чтобы облегчить пользователю процесс создания собственного приложения, в состав поставляемого программного обеспечения включен шаблон проекта Microsoft Visual Studio, применимый для построения как внешнего, так и внутреннего приложения UG/Open. Этот AWX-файл находится в папке $UGII_BASE_DIR$/UGOPEN (см. рис. 7.27) и в зависимости от установленной версии Unigraphics может иметь название, например - UgOpen_vl8.awx (для 18-й версии Unigraphics). Для использования этого шаблона в среде Microsoft Visual Studio необходимо скопировать файл *UgOpen_vl8. awx* в соответствующую папку шаблонов; на нашем компьютере она имеет расположение *C:\Program Files\ Microsoft VisualStudio\Common\MSDev98\ Template.*

Все необходимые приготовления сделаны, и можно приступить к созданию первого приложения UG/Open User Function. В качестве примера позвольте предложить вам реализацию построения весьма замысловатой кривой, заданной параметрическим способом (x = x(t), y = y(t)). Формулы, определяющие эту

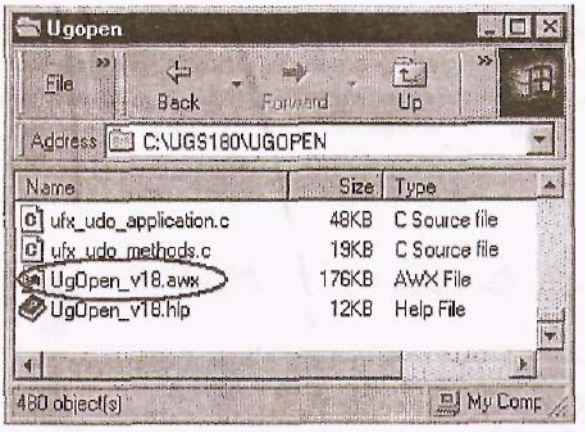


Рис. 7.27. Шаблон для Visual Studio C++

кривую, позаимствованы из рубрики «Занимательный Компьютер» журнала *Scientific American* (№ 3, 1985). «Безумие» - именно такое название дали авторы журнала этой кривой.

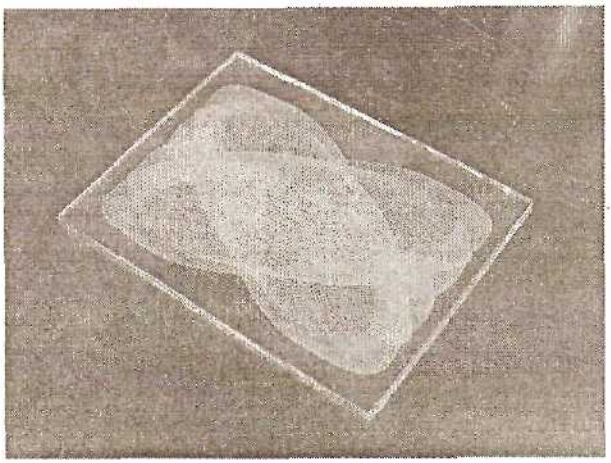
несмотря на то что кривая задается достаточно простыми формулами в полярной форме:

$$x = \sin(0.99t) - 0.7\cos(3.01t);$$

$$y = \cos(1.01t) + 0.1\sin(15.03t),$$

где переменная t - угол по отношению к положительному направлению оси X, радиус-вектор совершает 100 полных оборотов (что соответствует значению t=36000), прежде чем кривая замкнется, а полученная «осцилограмма» напоминает сеть, наброшенную на некий трехмерный объект.

Воспроизвести такую кривую интерактивными методами просто нереально. Даже использование методов построения кривых по аналитическим законам не всегда поможет в подобных ситуациях - слишком велика общая длина кривой. Поэтому предлагаем построить ее из отрезков прямых линий и не просто построить изображение кривой на экране, а получить CLS-файл управляющей программы для фрезерно-гравировального станка с числовым программным управлением с целью последующей гравировки этого замысловатого узора на металле, стекле и т.п. Мы уже проделали такой эксперимент, и результат гравировки алмазным резцом по стеклянной пластине приведен на снимке.



Для создания приложения UG/Open запускаем Microsoft Visual Studio 6.0 и создаем новый проект: ***File ~> New.*** В предлагаемом диалоговом окне выбираем закладку Projects и в качестве шаблона проекта указываем UG/OpenAppWizard V18 (см. рис. 7.28). Именно этот шаблон определяется файлом UgOpen_vl8.awx. Определите папку (Location:), в которой будет размещен проект, и дайте название создаваемому приложению (можете следовать приводимому примеру и назовите проект Mesh).

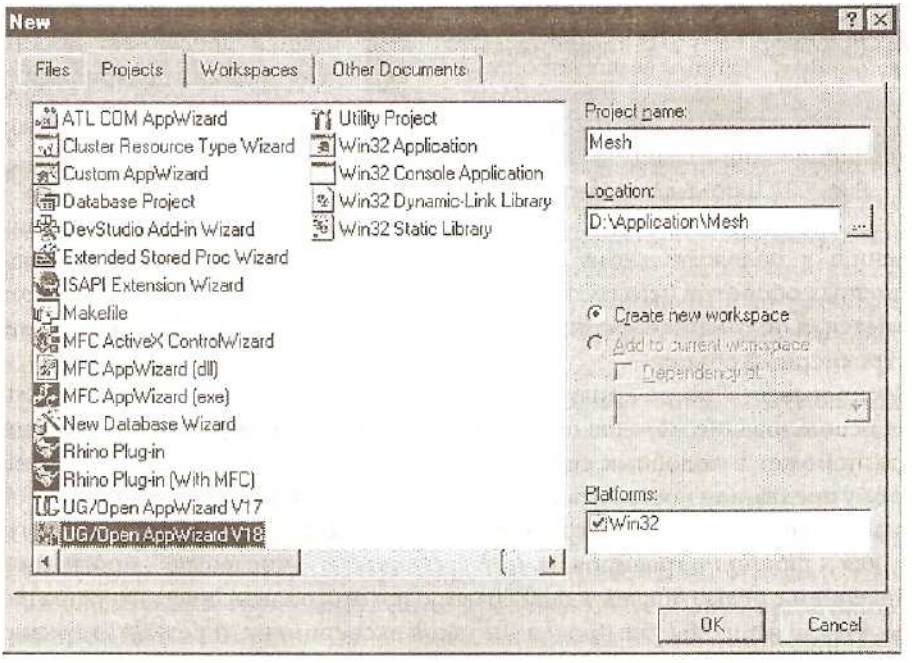


Рис. 7.28. Выбор шаблона UG/Open

Далее в действие вступает мастер (Wizard) формирования основы будущего приложения, и прежде всего вам будет предложено определить тип приложения (внешнее или внутреннее) и язык программирования (см. рис. 7.29), с помощью инструкций которого приложение будет создаваться.

Для нашего примера определяем тип создаваемого приложения как internal (внутреннее). Приложение активизируется подключением динамической DLL-библиотеки к процессу Unigraphics.

В качестве языка программирования будем использовать классический язык С (для реализации нашего примера его возможностей вполне достаточно).

Далее определяем способ активизации и выгрузки создаваемого приложения (см. рис. 7.30). Так как нет необходимости после завершения работы приложения оставлять его резидентно в памяти, назначаем его автоматическую выгрузку (unload) сразу после его исполнения. Для активизации приложения выберем

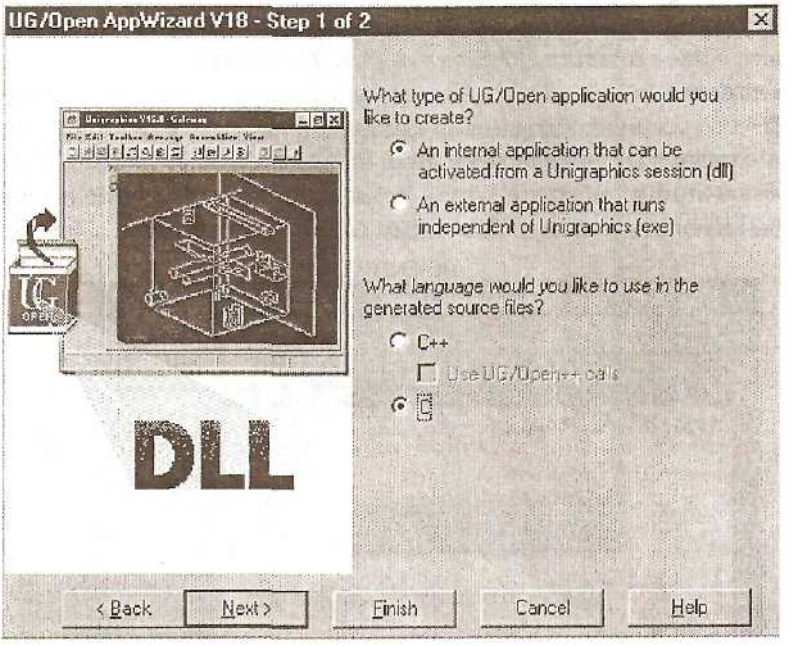


Рис. 7.29. Выбор языка программирования

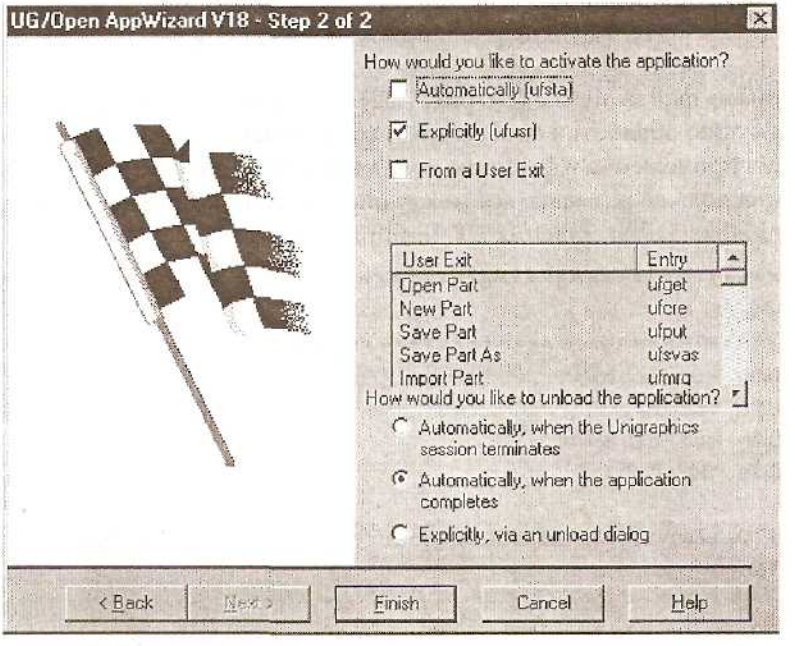


Рис. 7.30. Определение способа активизации и выгрузки приложения

явный способ запуска (Explicitly). В этом случае запуск производится через меню Unigraphics: ***File ~> Execute UG/Open ~> User Function...***

После нажатия кнопки «Finish» Visual Studio автоматически формирует рабочий проект Mesh со всеми необходимыми файлами и настройками компиляции и линковки для создания DLL-библиотеки Mesh.dll. В качестве активной конфигурации в меню Visual Studio выберите Win32 Release (наше приложение не столь сложное, чтобы включать отладку): ***Bild ~> Set Active Configuration...*** (см. рис- 7.31).

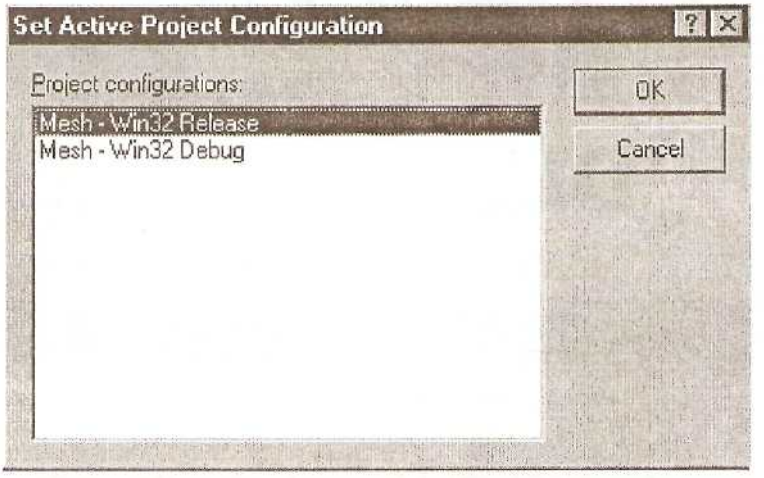


Рис. 7.31. Выбор активной конфигурации проекта

«Заготовка» файла Mesh.c (см. рис. 7.32) включает основную функцию приложения, функцию обработки сообщений о возможных ошибках инициализации и завершения приложения и функцию немедленной его выгрузки. Исходный текст приложения должен размещаться между операторами инициализации приложения UF_initialize() и завершения UF terminate().

```
/************************************************************
**    Mesh.c
************************************************************/
#include <stdio.h>
#include <uf.h>
#include <uf_ui.h>
#include "Mesh.h"

extern DllExport void ufusr(char *parm,int *returnCode,
int rlen )
{
    int errorCode = UF_initialize(); //Старт приложения
    if ( 0 == errorCode )
```

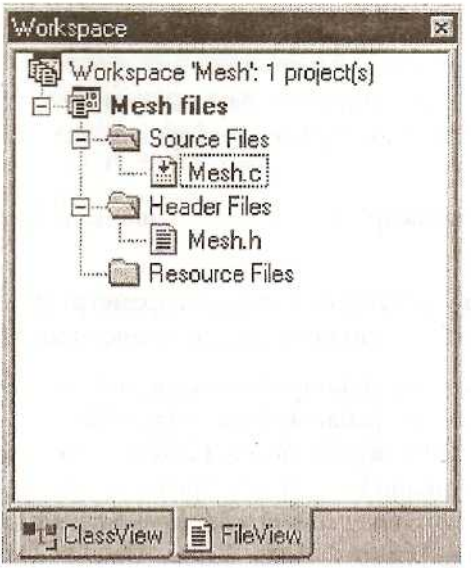


Рис. 7.32. Меню с "заготовкой" файла

```
        {
                //
                /* Исходный код приложения */
                //

                errorCode = UF__terminate();  //  Завершение приложения
        }
PrintErrorMessage( errorCode );
 }
extern int ufusr_ask_unload( void )
{
        return( UF_UNLOAD_IMMEDIATELY );
}

static void PrintErrorMessage( int errorCode ) {
        if ( 0 != errorCode )
        {
                char message[133];
                UF_get_fail_message( errorCode, message );
                UF_UI_set_status( message );
                fprintf( stderr, "%s\n", message );
        }
}
```

Приведем алгоритм работы приложения по построению параметрической кривой:

1. Запрос параметров кривой (габариты прямоугольной области, в которую будет вписана кривая, глубина гравировки, максимальное значение параметра t);
2. Вывод на экран габаритного прямоугольника и изменение масштабу, изображения (Fit);
3. Расчет в цикле с заданным шагом по параметру t всех точек кривой, соединение точек отрезками прямых, вывод информации в CLS-файл.

Для упрощения приложения предположим, что оно запускается после создания пользователем нового файла модели Unigraphics.

Прежде чем приступить к реализации изложенного алгоритма, необходимо отметить, что все API-функции Unigraphics сгруппированы по разделам и все определения, соглашения или имена переменных, относящиеся к этому разделу, сведены в определенный заголовочный .h-файл. Документация по UG/Open также построена с соблюдением этого принципа, что облегчает поиск требуемой функции или подпрограммы среди сотен API-функций, предоставляемых пользователю. **Вот** примеры основных разделов и соответствующих им заголовочных файлов.

Заголовочный файл	**Раздел**
uf_assem.h	Работа со сборочными узлами и составляющими их компонентами
uf_curve.h	Построение кривых и точек
uf_disp.h	Управление параметрами отображения информации на экране
uf_ui.h	Управление элементами пользовательского интерфейса (User Interface)
uf_modl.h	Программный интерфейс к наиболее общим операциям по работе с твердотельными моделями

Это далеко не полный перечень разделов, объединяющих определенные API-функции; приведены только те из них, которые потребуются для реализации нашего примера. Естественно, если вы обращаетесь к какой-либо API-функции Unigraphics, соответствующий заголовочный файл необходимо включить в исходный текст программы.

Разберем более подробно те функции, которые мы используем в нашем приложении. Для более полного изучения UG/Open API адресуем читателя к

документации Unigraphics: руководство пользователя содержит не только описание функций и способов обращения к ним, но и большое количество примеров с приведением исходных текстов программ.

Прежде всего создадим новый файл модели детали (part), в котором будет создаваться кривая. Функция UF_PART_new, создающая новый файл и делающая этот файл рабочим, выглядит следующим образом:

```
int UF_PART_new (
char * part_name ,     // Имя нового файла
int  units ,           // Определение единиц измерения (дюймы
                      // или миллиметры) // 1 = Миллиметры
                      // 2 = Дюймы
 tag_t * part          // Идентификатор вновь созданного файла
                       // или  NULL_TAG
```

Определение функции UFJPART_new находится в файле uf_part.h, его необходимо добавить в список заголовочных файлов.

Для ввода параметров создаваемой кривой воспользуемся функцией ис1608().

```
/*************************************************************
extern UGOPENINTEXPORT int ucl608(

char *menu,      <input>
                 Сообщение, выводимое в строке подсказки
                 (длина сообщения - до 80 символов).

char items[][16], <input>
                 Массив, включающий выводимые на экран
                 строки меню.

int ip3,         <input>
                 Количество пунктов меню.

int *ia4,        <input/output>
           Начальные значения целочисленных (int)
           переменных.

double *ra5,     <input/output>
                 Начальные значения вещественных (double)
                 переменных.

int *ip6         <input>
```

```
                    Целочисленный массив, определяющий тип
                    вводимых значений. Если ip[n] = 1,
                    вводимое значение будет
                    интерпретироваться как double, если же
                    ip[n] = 0, вводимое значение будет
                    интерпретироваться как целое (int).

};

/*************************************************************/
```

В зависимости от произведенных пользователем действий функция ис1608 возвращает следующие целочисленные значения:

1 = Нажата кнопка «Back» .

2 = Нажата кнопка «Cancel».

3= Нажата кнопка «ОК.», но пользователь не изменил начальных значений переменных.

4 = Нажата кнопка «ОК», при этом пользователь изменил как минимум одно из начальных значений переменных.

Определение функции ис1608 находится в файле uf_ui.h. Его необходимо добавить в список заголовочных файлов.

При запуске создаваемого приложения вышеописанное диалоговое окно будет выглядеть примерно так (см. рис. 7.33):

Для прекращения выполнения приложения при нажатии пользователем кнопок «Back» или «Cancel» введем в программу проверку возвращаемого функцией ис1608 значения res:

```
if (res == 1 || res == 2)
// Нажата кнопка "Back" или
//  "Cancel".
{
UF_terminate()•
}
```

Рис. 7.33. Пример диалогового окна

В этом случае дальнейшие построения кривой возможны только при нажатии кнопки "ОК". Параметры кривой могут быть изменены пользователем или приняты в виде предлагаемых начальных значений.

Для построения отрезка линии, соединяющей точки кривой с последовательными значениями параметра t, введем определение геометрического объекта Unigraphics:

```
tag_t    line;
```

Таким образом объявляется большинство объектов Unigraphics - точки, кривые, поверхности, твердотельные примитивы; tag_t - некий уникальный целочисленный идентификатор объекта модельного пространства, позволяющий программным способом производить создание или удаление объекта, придавать ему определенные свойства или статус (цвет, слой, погашен/высвечен и т.п.).

Следует отметить, что при создании нескольких объектов Unigraphics с одним и тем же идентификатором tag_t (такое тоже возможно) значение идентификатора присваивается только последнему из создаваемых объектов. В нашем примере все отрезки линий (а их количество может достигать десятков тысяч) создаются в программе с одним и тем же идентификатором line - в этом случае программно идентифицировать возможно только последний созданный объект. Для идентификации каждого сегмента линии необходимо определить массив объектов tag_t.

Для создания отрезка линии с заданными координатами начальной и конечной точки необходимо объявить структуру типа UF_CURVE_Iine_L (ее определение включает файл uf_curve.h; его необходимо добавить в список заголовочных файлов):

```
UF_CURVE_line_t  line_coords;
```

При таком способе определения точек линии координаты стартовой точки записываются в виде:

```
X =  line_coords.start_point  [0];
Y =  line_coords.start_point  [1];
Z =  line_coords . start__point  [2] ;
```

А координаты конечной точки:

```
X =  line_coords . end__point  [0j;
Y =  line_coords . end_point  [1],-
Z  =  line_coords.end_point  [2];
```

Для обращения к тригонометрическим функциям sin() и cos() добавим в список заголовочных файлов math.h и приступим собственно к построению ломаной, аппроксимирующей кривую.

Так как все отрезки линий создаются по стартовой и конечной точкам, координаты первой точки ломаной (t=0) рассчитаем вне цикла, а координаты всех последующих в цикле с изменением параметра t от 0 до максимального значения с

заданным приращением. Z-координаты для всех точек положим равными 0; значение глубины фрезерования будет учтено в формируемой управляющей программе для станка с ЧПУ.

Приведенная система уравнений формирует кривую, вписанную в прямоугольник со сторонами 3.4 и 2.2 (эти значения определяются множителями перед синусами и косинусами), поэтому для «растяжки» кривой до заданных в меню габаритов будем умножать получаемые координаты точек па соответствующие коэффициенты.

Итак:

```
t = 0;
line_coords.start_point[0]=
(W/3.4)*(sin(0.99*t)-0.7*cos(3.01*t) ) ;
line_coords.start_point [1] =
(H/2.2)*(cos{1.01*t)+0.1*sin(15.03*t));
line_coords.start_point[2]=0.0;
for (t=delta_t; t<=t_max; t=t+deita_t)
{
line_coords.end_point[0]=
(W/3.4)*(sin(0.99*(t*PI/18 0.))-0.7*cos(3.01*(t*PI/180.)));
line_coords.end_point[1]-
(H/2.2)*(cos(1.01*(t*PI/180.))+0.1*sin(15.03*(t*PI/180.)));
line_coords.end_point[2]=0.0;
UF_CURVE_create_line(&line_coords, &line);
line_coords.start_point[0]=line_coords.end_point[0];
line_coords.start_point[1]=line_coords.end_point[1];
line_coords.start_point[2]=line_coords.end_point[2];
}
```

Определение функции UF_CURVE_create_line находится в файле uf_curve.h; его необходимо добавить в список заголовочных файлов.

Небольшое замечание: при вычислении тригонометрических функций для перевода градусов в радианы использовалась константа PI=3.14159..., определенная в файле **uf_defs.h.**

Полный текст файла Mech.c выглядит так:

```
/*************************************************************
**      Mesh.c
*************************************************************/
#include <stdio.h>
#include <uf.h>
#include <uf_ui.h>
#include <uf_curve.h>
#include <uf_part.h>
#include <math.h>
```

```
 #include   "Mesh.h"

 extern DllExport void ufusr( char *parm, int *returnCode,
 int rlen )
 {
 int res;
     char menu[5][16]-{"Ширина","Высота","Z гравировки",
     "1 max", "Delta t"};
     double W, H, Z_eng, t_max, delta_t;
     int  menu_items - 5;
     int   default_int[5] = {1,1,1,1,1};
     double default_double[5] = {200.0, 100.0, 0.1, 3600.0,
        1.0};
     int   var_type[5] -- {1, 1, 1, 1, 1}; // 1 == double;
        0 == int;
     double t;
     tag_t line;
     UF_CURVE_line_t line_coords;

 int errorCode = UF_initialize () ;
 if ( 0 == errorCode )
 {
res = UF_PART_new ("Mesh.prt", 1, &part); // Создание
                                          // новой части
res = ucl608("Введите параметры", menu, menu_items,
 default_int, default_double, var_type);
if (res == 1 j | res -~ 2) //Нажата кнопка "Back"или"Cancel"
{
UF_terminate();
}

W         = default_double[0];
H         = default_double[l];
Z_eng     = default_double[2];
t_max     = default_double[3];
delta_t   = default_double[4];

t = 0;
line_coords.start_point[0]=
(W/3.4)*(sin(0.99*t)-0.7*cos(3.01*t) ) ;
line_coords.start_point[1]=
(H/2.2)*(cos(1.01*t)+0.1*sin(15.03*t) ) ;
line_coords.start_point[2]=0.0;
```

```
for (t=delta_t; t<=t_jmax; t=t+delta_t)
{
line_coords.end_point[O] =
(W/3.4)*(sin(0.99*U*Pl/180.>)-0.7*cos(3.01*(t*PI/180.)));
line_coords.end_point [1] =
(H/2.2)*(cos(1.01*(t*PI/180.))+0.1*sin(15.03*(t*PI/180.)));
line_coords.end_point[2]=0.0;

UF_CURVE_create_line (&line_coords, &line);

line_coords.start_point[0]=line_coords.end_point[0];
line_coords.start_point[1]-line_coords.end_point[1];
line_coords.start_point[2]=line_coords.end_point[2];
}
errorCode = UF_terminate();
}
   PrintErrorMessage( errorCode );
   }
   extern  int  ufusr_ask_unload ( void )
   {
   return( UF_UNLOAD_IMMEDIATELY );
   }

   static void PrintErrorMessage int errorCode )
   {
    if ( 0 != errorCode )
    {
        char message[133];
        UF_get_fail_message ( errorCode, message );
        UF_UI_set_status( message );
        fprintf( stderr, "%s\n", message );
    }
   }
```

Для формирования динамической библиотеки Mesh.dll воспользуемся меню Microsoft Visual C++ ***(Build ~> Build Mesh.dll,*** «горячая» клавиша - *FT)* (см. рис. 7.34). Результат этих действий - файл Mesh.dll в папке ...\Release.

ЗАПУСК ПРИЛОЖЕНИЯ USER FUNCTION

Для первого запуска приложения User Function запустите Unigraphics и, используя основное меню ***File ~> Execute UG/Open ~> User Function...*** (см. рис. 7.35) или комбинацию клавиш *Ctrl+ U,* выберите созданную вами динамическую библиотеку Mesh.dll, которая в данном случае представляет собой внутреннее (internal) приложение UG/Open.

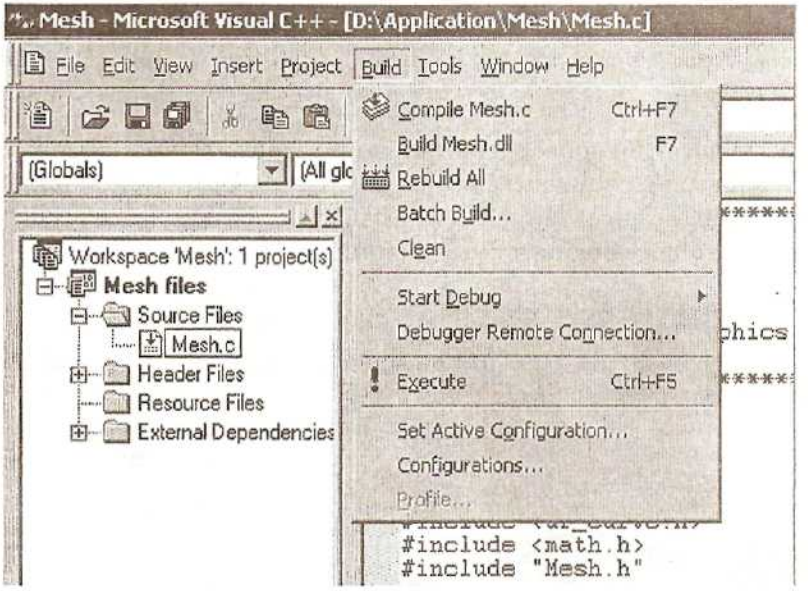


Рис. 7.34. Обращение к функции Build Mesh.dll

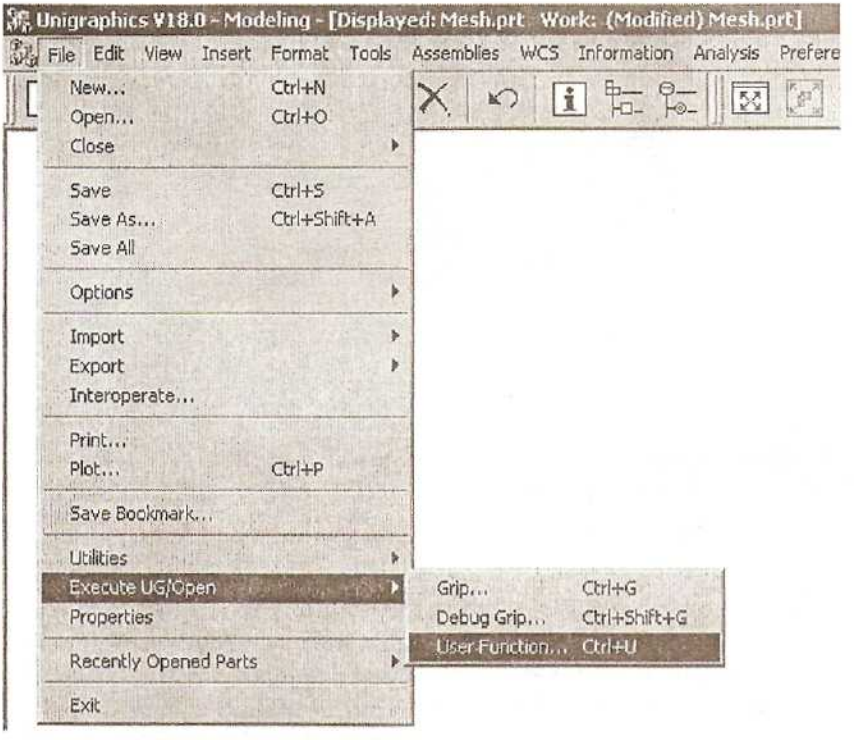


Рис. 7.35. Запуск приложения User Function

Запуск разработанных пользователем приложений User Function возможен только в том случае, если вы располагаете лицензией на запуск UG/Open.

Ниже (см. рис. 7.36) приведен пример выполнения приложения Mesh.dll с различной комбинацией исходных параметров. В первом случае параметр t изменяется от 0 до 3600 градусов (т.е. радиус-вектор совершает 10 полных оборотов), а кривая вписывается в прямоугольник с размерами 200x100.

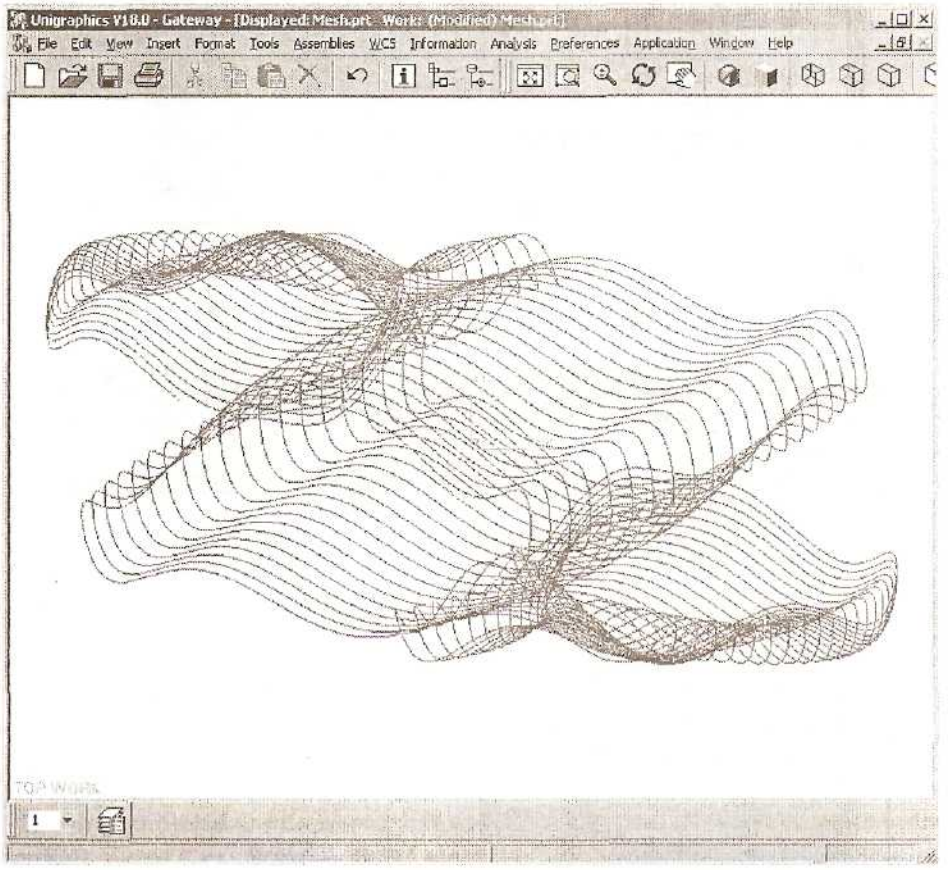


Рис. 7.36. Кривая при значении параметра t = 3600

Второй рисунок (см. рис. 7.37) показывает эту любопытную кривую во всей красе: параметр t меняется от 0 до 36000 (или 100 полных оборотов радиус-вектора)! Открою вам секрет: это то значение параметра, при котором кривая впервые замыкается. Дальнейшее увеличение параметра t не имеет смысла: координаты точек будут повторяться. Однако это справедливо только для тех коэффициентов,

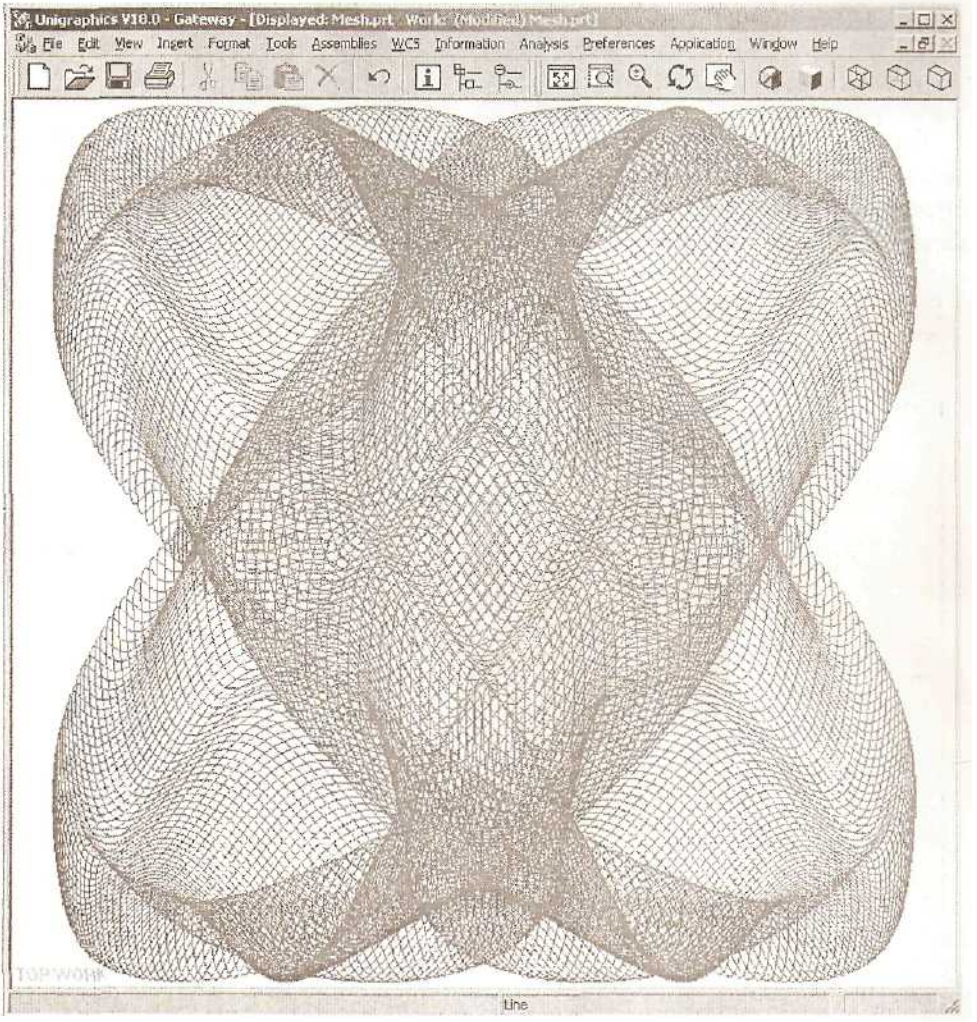


Рис. 7.37. Окончательный вид кривой

которые приведены в примере (0.99,1.01, 3.01, 15.03). Попробуйте изменить их-и вы получите совершенно другую, непредсказуемую форму кривой.

Более того, в качестве самостоятельного упражнения предлагаем добавить в меню выбора параметров кривой эти коэффициенты, чтобы иметь возможность изменять их, не прибегая к перекомпиляции DLL-библиотеки. Можно добавить и дополнительные возможности: автоматическое масштабирование изображения после завершения построения, выбор цвета по определенному закону для каждого создаваемого отрезка. Еще одно направление - определение закона изменения Z-координаты для каждой точки, и тогда замысловатая кривая престанет быть плоской.