УЧЕБНЫЙ КУРС
по геометрическому ядру C3D

РАБОТА 1

Знакомство с ядром C3D

https://c3dlabs.com
Москва
2018

Содержание

1. Введение

Геометрическое ядро C3D (C3D Labs, дочерняя компания АСКОН) – это программная библиотека, предназначенная для построения математических моделей геометрической формы твердотельных объектов и математического обслуживания этих моделей (редактирование, наложение связей на элементы модели, построение триангуляции для визуализации, расчет инерционных характеристик, определение столкновений элементов модели, построение плоских проекций модели, обмен данными с другими системами). Эти модели в компьютерной памяти представляются в виде объектов, элементами которых являются вершины, ребра, грани, оболочки. Операции отображения математических моделей и взаимодействие с ними в интерактивном режиме в состав геометрического ядра не входят. Как правило, эти операции реализуются в конкретной прикладной программе (например, в САПР КОМПАС-3D производства АСКОН). Однако, для изучения ядра C3D необходимо иметь возможность отображения и проверки структуры геометрических моделей. Поэтому в качестве первого вопроса при знакомстве с ядром C3D будет рассмотрена сборка тестового приложения и дополнение этого приложения новыми функциями для построения новых геометрических моделей.

В данной работе рассматриваются следующие основные темы:

- установка дистрибутива,
- сборка тестового приложения,
- построение и отображение простых геометрических моделей в составе тестового приложения.

Текст данной практической работы подразумевает самостоятельное выполнение описываемых действий. Эти действия рассматриваются в разделах работы, а также повторяются в конце каждого раздела в пункте **«Задания»**. Предполагается, что эти задания будут самостоятельно выполнены перед переходом к следующему разделу работы.

В качестве базовой среды программирования используется Microsoft Visual Studio 2017 в конфигурации x64/Unicode в среде 64-х разрядной ОС Microsoft Windows 10 (выбор версии среды и ОС не принципиален, может применяться и другая версия из числа поддерживаемых ядром C3D).

2. Состав ядра

Дистрибутив ядра C3D для ОС MS Windows поставляется в виде одного архивного файла C3D_Kernel_NNNNNN.zip, в имени которого “NNNNNN” обозначает номер сборки (версии). Распакуйте архивный файл ядра в отдельный каталог. Имя и расположение этого каталога не имеют принципиального значения, и далее этот базовый каталог может обозначаться как <BaseDir> (или совсем не будет упоминаться).

В базовом каталоге ядра имеются следующие элементы:

- **Kernel** – каталог архивов с файлами геометрического ядра.
- **Documentation** – каталог с документацией по ядру в форматах PDF и CHM[1].
- **Example.zip** – архив с файлами проекта для компиляции тестового приложения ядра в среде программирования MS Visual Studio.

Сделаем несколько замечаний относительно этих элементов дистрибутива ядра.

В каталоге **Kernel** располагаются архивы, в каждом из которых содержатся скомпилированные файлы геометрического ядра c3d.dll, c3d.lib и заголовочные файлы ядра на языке

[1] CHM (Compiled HTML Help File) – формат справочных файлов компании Microsoft. В ОС Windows обычно просматриваются с помощью системной утилиты hh.exe.

программирования C++. Ядро поставляется в нескольких вариантах, рассчитанных на разные версии сред программирования и операционной системы. Некоторая избыточность комплектации связана с тем, что скомпилированное представление программ на Си++ зависит и от аппаратных особенностей процессора, и от особенностей целевой операционной системы, и от среды программирования. Такой подход позволяет повысить эффективность вычислений за счет использования новых возможностей языка Си++ и возможностей новых версий компиляторов.

Для каждой версии Visual Studio C++ имеется несколько архивов, например, для Visual Studio 2017:

Win64_Visual_Studio_2017.zip
Win64_Visual_Studio_2017_Unicode.zip

Варианты скомпилированного ядра отличаются разрядностью (определяет объем памяти для хранения целых чисел и указателей) и представлением символьных данных (устаревшая однобайтная кодировка или двухбайтная Unicode). Эти отличия закодированы в именах архивов с файлами ядра по следующему шаблону:

<Разрядность программы>_Visual_Studio_<Версия среды>_<Представление символьных данных>

В каждом архиве файлы ядра распределены по трем каталогам: **Debug**, **Release** и **Include**. В каталогах **Debug** и **Release** хранятся 2 скомпилированных файла ядра: c3d.dll и c3d.lib. Это динамическая и статическая библиотеки. Динамическая библиотека – собственно геометрическое ядро, поэтому файл c3d.dll должен распространяться вместе с приложением. Статическая библиотека импорта c3d.lib необходима только для разработки приложения, для организации связи между приложением с динамической библиотекой в процессе компиляции и сборки. Каталог **Include** содержит заголовочные файлы, описывающие типы данных, классы и функции, реализованные в ядре. Количество заголовочных файлов довольно велико – несколько сотен. По общепринятым правилам программирования на Си++, в ядре преобладает подход размещения в одном заголовочном файле описания только одного класса.

Каталог **Documentation** содержит справочную документацию в 2-х форматах: в виде *.chm-файлов и в формате PDF. Содержательно документация тесно связана с текущей версией геометрического ядра, она построена на основе автоматического извлечения комментариев к исходным текстам ядра и поэтому содержит наиболее актуальную информацию о нем. При открытии справки в CHM-формате в ОС Windows 7 может появиться предупреждение системы безопасности операционной системы, и содержимое справки показано не будет. В таком случае, разблокируйте файл справки (вызовите его окно свойств и нажмите кнопку **Разблокировать**). Актуальная версия документации по последней версии ядра также доступна в Интернет на сайте разработчика по адресу:

http://c3d.ascon.ru/doc/math/index.html

Для работы со справочной системой на этом сайте рекомендуется использовать браузер Google Chrome.

В архиве **Example.zip** находятся файлы проекта тестового приложения: его исходный текст, служебные файлы проекта для самостоятельной сборки приложения в среде MS Visual Studio, несколько вариантов скомпилированного исполняемого файла различной разрядности и примеры файлов с геометрическими моделями.

2.1 Задания

1) Распакуйте дистрибутив геометрического ядра в отдельный специально созданный каталог. Найдите все перечисленные элементы ядра.

2) Распакуйте в базовый каталог ядра архив из каталога **Kernel**, соответствующий вашей среде программирования (например, Win64_Visual_Studio_2017_Unicode.zip). В базовом каталоге ядра должны появиться три новых каталога: **Debug**, **Release** и **Include**.
3) Вызовите справочную систему в CHM-формате, в PDF-формате и на сайте C3D. Найдите описание класса `MbSpaceItem` (базовый класс геометрического объекта в трехмерном пространстве). Найдите диаграмму классов, на которой показаны унаследованные классы геометрических объектов, а также описание интерфейса класса.
4) Распакуйте архив Example.zip в каталог <BaseDir>\Example. В каталоге Example\Demo есть несколько версий тестового приложения для разных версий ОС и с поддержкой разных языков интерфейса пользователя (русский или английский). Запустите тестовое приложение test.exe из каталога <BaseDir>\Example\Demo, соответствующего вашей среде программирования. Например, для запуска тестового приложения с сообщениями на русском языке в среде 64-разрядной ОС Windows запустите исполняемый файл <BaseDir>\Example\Demo\Win64_Rus\test.exe.
5) В главном окне запущенного тестового приложения выберите команду меню ***Помощь ⇒ Лицензионный ключ*** и в двух последовательно появляющихся диалоговых окнах введите свои значения лицензионного ключа и сигнатуры, определяющие режим использования вашей версии ядра.
6) В тестовом приложении test.exe откройте файлы с примерами геометрических моделей из каталога Examples\Models.

3. Тестовое приложение

Тестовое приложение test.exe – это оконное приложение для ОС Windows, демонстрирующее способы использования геометрического ядра. В нем реализованы следующие основные возможности:

- хранение в памяти набора геометрических моделей, построенных с помощью C3D;
- отображение геометрических моделей;
- загрузка и сохранение геометрических моделей в файлах различных форматов, поддерживаемых ядром;
- просмотр структуры геометрических моделей;
- команды меню для вызова различных функций ядра с возможностью указания параметров этих функций в диалоговом режиме (например, кривых для построения трехмерных моделей);
- команды меню для вызова функций, реализованных пользователем.

Приложение test.exe можно рассматривать как инструментальное средство низкого уровня для работы с ядром C3D. Под «низкоуровневостью» здесь подразумевается, что возможности реализованного интерфейса пользователя приближены к программному интерфейсу ядра (не следует их сравнивать с интерфейсом пользователя полномасштабных САПР). Тестовое приложение позволяет изменять параметры вызова функций ядра без перекомпиляции приложения, а также выполнять отладку пользовательских функций для работы с ядром (для этого требуется перекомпиляция).

Ниже перечислены некоторые часто используемые команды интерфейса пользователя программы test.exe, предназначенные для манипуляций с геометрическими моделями в окне тестового приложения. Более подробный список команд, связанных со служебными клавишами, можно получить командой верхнего меню ***Помощь ⇒ Список горячих клавиш***.

Таблица 1. Некоторые команды интерфейса пользователя приложения test.exe

Команда	Способ выполнения
Изменение масштаба отображения.	Клавиши + и – на дополнительной цифровой клавиатуре или колесико мыши
Перемещение модели вдоль осей проекционной плоскости.	Клавиши-стрелки
Совмещение начала координат с центром окна отображения.	Клавиша * на дополнительной цифровой клавиатуре.
Совмещение начала координат с центром окна отображения и изменение масштаба для полного отображения модели.	Ctrl + * на дополнительной цифровой клавиатуре.
Вращение модели вокруг координатных осей.	Ctrl + клавиши-стрелки Shift + клавиши-стрелки
Выбор проекционного вида (изометрия, вид сбоку, вид сверху и т.п.)	Команды верхнего меню ***Окно ⇒ Ориентировать***

3.1 Использование готового проекта для сборки тестового приложения

Для сборки приложения test.exe необходимо сформировать проект для используемой версии среды Microsoft Visual Studio. В комплект поставки C3D входят проекты для нескольких версий среды программирования.

Если проект для вашей версии среды программирования имеется, то выполните следующие действия (если нет, то перейдите к п. 3.2):

1) Чтобы не вносить изменений в исходную версию тестового приложения, для дальнейшей работы сделайте копию каталога <BaseDir>\Example. Далее будем обозначать этот каталог TestApp.
2) Откройте проект Test_ NNNNN.sln, где “NNNNN” – версия вашей среды программирования.
3) В окне **Solution Explorer** у проекта Test вызовите контекстное меню и выберите в нем команду ***Properties***. В левой части этого окна в дереве свойств выберите группу каталогов **VC++ Directories** (рис. 1).

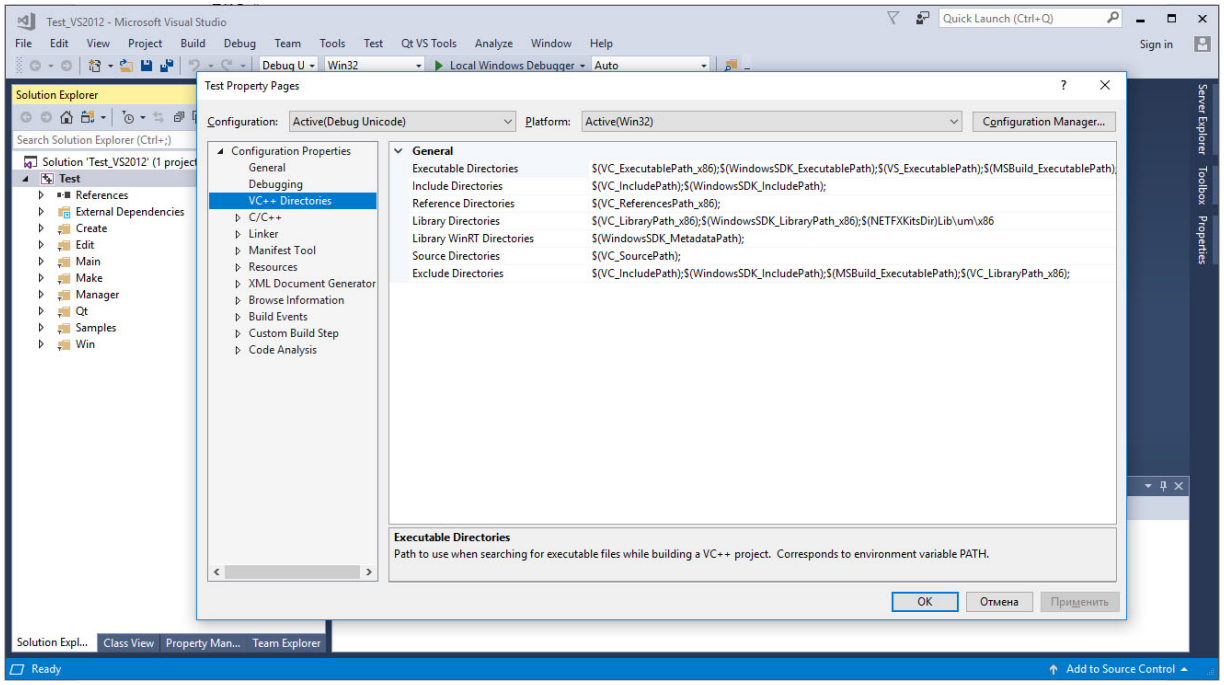

Рис. 1. Среда MS Visual Studio 2017: диалоговое окно свойств проекта, группа свойств «Каталоги» (VC++ Directories).

4) В окне свойств (рис. 1) в правой части в разделе **Include Directories** добавьте путь к каталогу заголовочных файлов <BaseDir>\Include. Будьте внимательны: не удаляйте уже имеющиеся пути. Проделайте эту процедуру для обеих конфигураций сборки – для отладочной Debug и для окончательной Release (тип сборки выбирается в верхней левой части окна в списке Configuration). Обеим конфигурациям указывайте один и тот же каталог заголовочных файлов <BaseDir>\Include.
5) Аналогично п. 4), в разделе **Library Directories** укажите пути к каталогу статических библиотек c3d.lib. Для отладочной версии сборки (Debug) надо указать путь <BaseDir>\Debug, а для окончательной версии (Release) – путь <BaseDir>\Release. Будьте внимательны: (а) не удаляйте уже имеющиеся пути; (б) обратите внимание, что для Debug и Release-версий сборки следует указывать соответствующие каталоги статических библиотек.
6) Для переключения тестового приложения на русский язык нужно указать константу `__NATIVE_LANGUAGE__` в двух местах для каждой конфигурации (Debug и Release): (а) в качестве команды препроцессора, и (б) в качестве команды компилятора ресурсов. Это делается в том же окне свойств проекта, что использовалось ранее (рис. 1). Путь для указания команды препроцессора: ***Configuration Properties ⇒ C/C++ ⇒ Preprocessor***, свойство **Preprocessor Definitions**.
   Путь для указания команды компилятора ресурсов: ***Configuration Properties ⇒ Resources ⇒ General***, свойство **Preprocessor Definitions**.
   Будьте внимательны: при изменении свойств не удаляйте уже существующие команды, а только добавьте дополнительную константу `__NATIVE_LANGUAGE__` (в начале и в конце имени два символа подчеркивания).
7) Откройте входящий в проект исходный файл TestApp\Source\Manager\test_manager.cpp. Найдите вызов функции EnableMathModules() в конструкторе класса Manager::Manager(). В двух строках инициализации переменных str0 и str1 вместо значений “key” и “signature” укажите значения вашего лицензионного ключа и сигнатуры:
```
std::string str0("key");
std::string str1("signature");
```
8) Соберите проект (команда ***Build ⇒ Build Solution***). При попытке запуска приложения будет выдано сообщение об отсутствии динамической библиотеки ядра c3d.dll.

9) Скопируйте файл динамической библиотеки c3d.dll из соответствующего каталога <BaseDir> в каталог, где формируется файл приложения test.exe (в зависимости от варианта сборки – из каталога <BaseDir>\Debug или <BaseDir>\Release). Повторите запуск тестового приложения.

10) Запустите проект в конфигурациях Debug и Release. Убедитесь, что приложение работает и успешно выполняется загрузка примеров моделей из каталога TestApp\Models (рис. 2).

11) Ознакомьтесь с действиями служебных клавиш, перечисленных в таблице 1, а также тех, которые перечислены в информационном окне команды ***Помощь ⇒ Список горячих клавиш***.

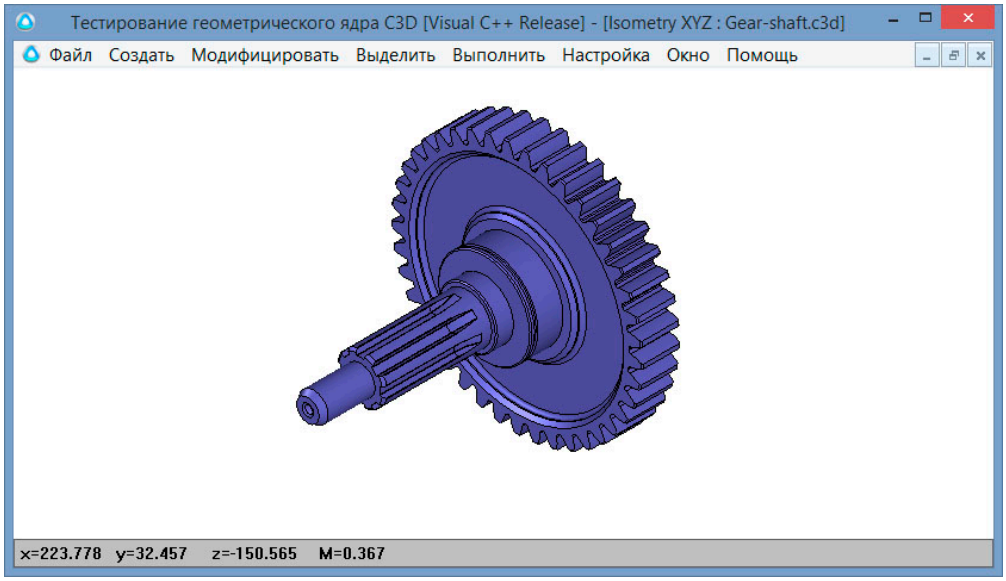

Рис. 2. Главное окно тестового приложения test.exe с открытым файлом-примером трехмерной модели (вал-шестерня).

3.2 Формирование проекта для сборки тестового приложения

Если для вашей версии среды разработки не имеется готового проекта, то его можно сформировать самостоятельно с помощью свободно доступного приложения – утилиты CMake. Она позволяет сформировать файлы проекта для конкретной версии среды программирования на базе текстового описания проекта на языке этой утилиты. Обычно это текстовое описание проекта хранится в файле CMakeLists.txt. Откройте этот файл Example\Source\CMakeLists.txt и ознакомьтесь с его содержимым – вы увидите текст на типичном командном языке с использованием макросов, среди которых заметно перечисление исходных файлов приложения и используемых библиотек.

Для формирования проекта и сборки тестового приложения выполните перечисленные ниже действия.

1) Установите приложение CMake (дистрибутив для ОС Windows можно загрузить с сайта www.cmake.org).

2) Чтобы не вносить изменений в исходную версию тестового приложения, для дальнейшей работы сделайте копию каталога <BaseDir>\Example. Далее будем обозначать этот каталог TestApp.

3) Запустите CMake (рис. 3).

В строке **Where is the source code** укажите каталог с исходными файлами проекта TestApp\Source.
В строке **Where to build the binaries** укажите каталог, где будут сформированы файлы проекта TestApp\Build (введите имя каталога вручную, поскольку пока он отсутствует).
Нажмите кнопку **Configure** для конфигурирования проекта. На запрос **Create Directory** подтвердите создание каталога TestApp\Build. В появившемся диалоговом окне **Specify the generator for this project** укажите требуемую конфигурацию среды разработки (например, Visual Studio 15 2017 Win64).
Нажмите кнопку **Generate** для генерации файлов проекта.

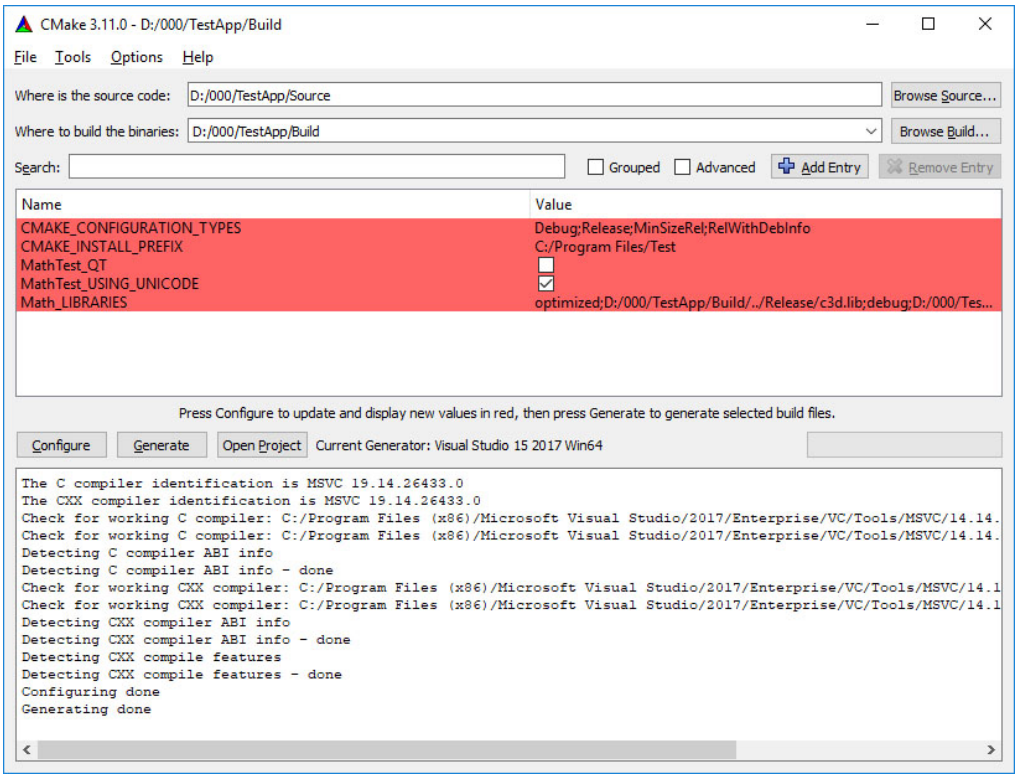

Рис. 3. Главное окно утилиты CMake, предназначенной для автоматизации сборки программных проектов (после выполнения шага 3).

4) В Visual Studio откройте проект (точнее, многопроектное решение – Solution, его описание для Visual Studio располагается в файле *.sln) из файла
TestApp\Build\Test.sln

5) После открытия решения Test.sln сделайте основным проект Test. Для этого в окне **Solution Explorer** (рис. 4) выберите для проекта Test команду контекстного меню **Set As StartUp Project**:

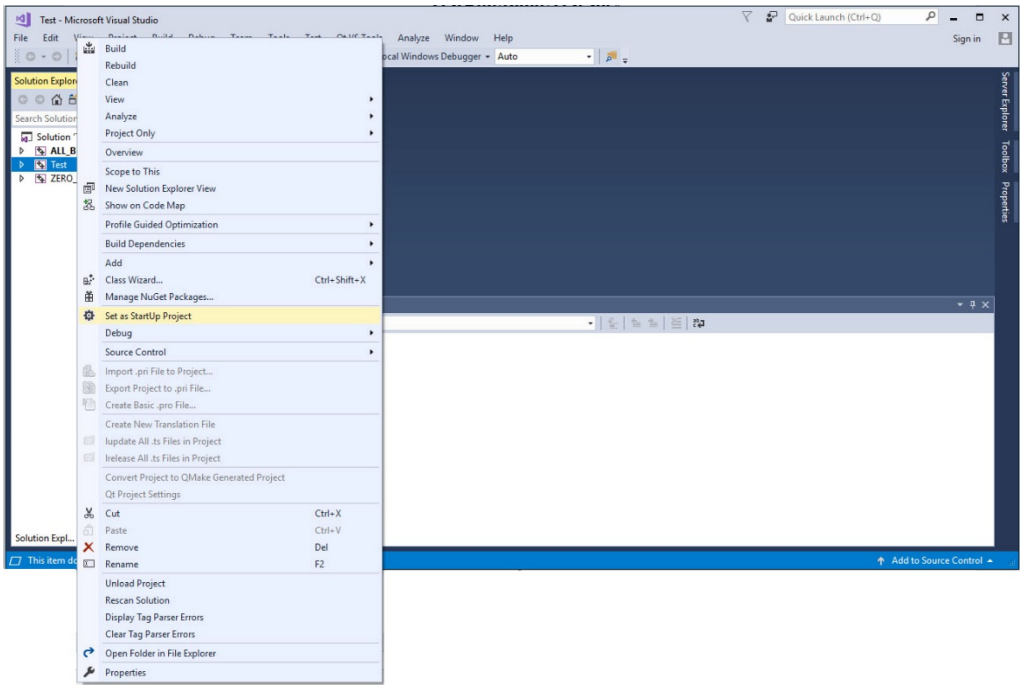

Рис. 4. Окно среды программирования MS Visual Studio 2017 с открытым решением для сборки тестового приложения Test.sln.

6) Для настройки и запуска проекта выполните пункты 3)-9) из раздела 3.1. При выполнении п. 8) выполняйте копирование пар файлов библиотек – и статической библиотеки c3d.lib, и динамической библиотеки c3d.dll.
   Обратите внимание: если после выполнения генерации проекта в CMake в окне будет отмечена переменная-флаг MathTest_USING_UNICODE (как показано на рис. 3), то при копировании библиотек следует использовать версии для символьной кодировки Unicode.
7) Соберите и запустите проект в конфигурациях Debug и Release. Убедитесь, что приложение работает – выполняется загрузка примеров моделей из каталога TestApp\Models (рис. 2).
8) Ознакомьтесь с действиями служебных клавиш, перечисленных в таблице 1, а также тех, которые перечислены в информационном окне команды ***Помощь ⇒ Список горячих клавиш***.

4. Построение точек

В тестовом приложении в меню ***Помощь*** имеется вложенное меню ***Создание новых команд***, в котором предусмотрены 10 команд для вызова функций пользователя. Заготовки этих функций содержатся в исходном файле проекта TestApp – в файле TestApp\Source\Create\test_user.cpp. В Visual Studio откройте этот файл, пользуясь окном **Solution Explorer**. Найдите в нем функции с именами MakeUserCommand0() – MakeUserCommand 9(). Это обработчики команд меню, в которые можно добавлять собственный программный текст. Пожалуй, это наиболее простой способ попробовать работу с геометрическим ядром – в приложении test.exe уже имеется готовый механизм отображения создаваемых геометрических моделей.

В данном разделе будут рассмотрены несколько примеров построения геометрических моделей из простейших объектов. Подробное описание классов этих объектов будет приведено в последующих работах.

4.1. Двумерные точки

В качестве первого примера построим простейший геометрический объект – двумерную точку. Для ее представления в C3D используется класс MbCartPoint (точка в декартовой системе координат).

Попробуем построить в тестовом приложении точку с координатами (1, 0), которая должна попасть на горизонтальную ось используемой для построения системы координат (СК).

В начало файла test_user.cpp после строки с подключением пространства имен c3d добавьте команду подключения пространства имен с переменными и константами тестового приложения TestVariables. Эти две строки в начале файла должны выглядеть так:

```
using namespace c3d;
using namespace TestVariables;
```

Теперь добавьте в тестовое приложение обработчик пользовательской команды MakeUserCommand0 со следующим содержимым:

Пример 4.1.1. Построение двумерной точки

```
void MakeUserCommand0()
{
  // Создание двумерной точки в виде объекта MbCartPoint ядра C3D
  MbCartPoint pnt( 1, 0 );
  // Вызов метода отображения у объекта тестового приложения
  viewManager->AddObject( PPOINT_Style, pnt );
}
```

В приведенном обработчике выполняются две операции – создание простейшей геометрической модели в виде единственной точки и операция отображения. Первая операция выполняется средствами ядра C3D, вторая – с помощью объекта тестового приложения viewManager.

Метод viewManager->AddObject() выполняет довольно много действий: преобразует двумерную точку в трехмерную MbCartPoint3D, строит на ее базе точечный каркас MbPointFrame и добавляет этот каркас в геометрическую модель MbModel, которая и отображается в окне приложения test.exe. Пока эти действия подробно рассматривать не будем, полагая, что мы имеем дело с одной геометрической моделью, в которую в виде объектов ядра C3D можем добавлять новые геометрические объекты и отображать их в окне тестового приложения.

Запустите test.exe и выполните реализованную команду выбором команды меню ***Помощь ⇒ Создание новых команд ⇒ Пользовательская команда 0***. Результат может оказаться неожиданным – точка не будет видна, хотя мировая система координат построена. Для того, чтобы выяснить, произведено ли построение, нажмите клавиши Ctrl+*, чтобы центрировать модель в окне с автоматическим подбором масштаба отображения. Теперь точка окажется в центре окна (рис. 5, слева), однако мировой системы координат не видно. Клавишей “–“ или колесиком мыши отрегулируйте масштаб отображения, чтобы одновременно была видна и точка, и система координат (рис. 5, справа).

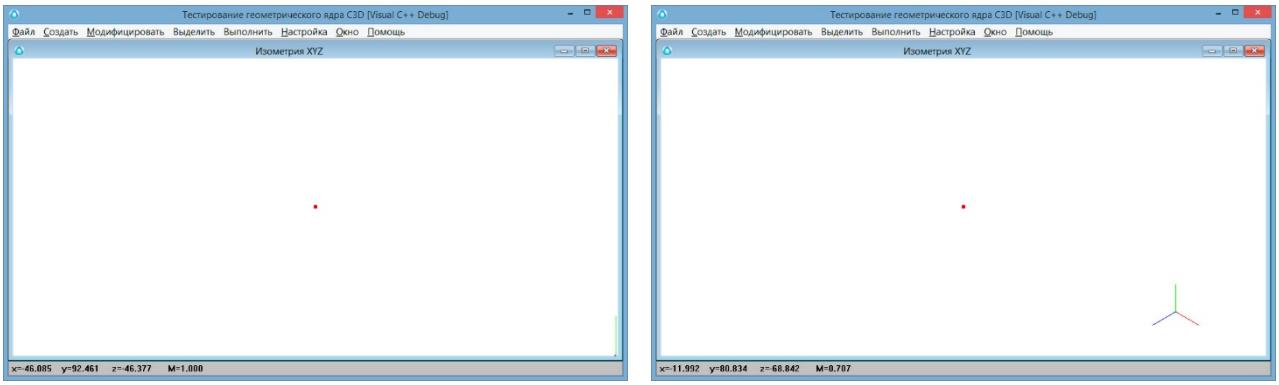

Рис. 5. Отображение точки в окне тестового приложения в двух различных масштабах (пример 4.1.1).

Как и во многих других библиотеках компьютерной графики, при построении геометрических объектов в C3D необходимо учитывать расположение и ориентацию используемых систем координат. При построении двумерных объектов их необходимо преобразовывать в трехмерные, для чего требуется явно указывать используемую трехмерную СК.

До сих пор локальная система координат не указывалась, так что ее положение было не определено (оно зависит от действий пользователя в окне тестового приложения). Измените обработчик для построения точки так, чтобы явно указать систему координат, используемую при построении – она будет совпадать с мировой СК (пример 4.1.2).

Пример 4.1.2. Построение двумерной точки в локальной СК

```
void MakeUserCommand0()
{
  // Локальная система координат - по умолчанию совпадает с мировой СК
  MbPlacement3D pl;
  // Создание двумерной точки в виде объекта MbCartPoint ядра C3D
  MbCartPoint pnt( 1, 0 );
  // Добавление точки в геометрическую модель тестового приложения
  viewManager->AddObject( PPOINT_Style, pnt, &pl );
}
```

Для использования класса MbPlacement3D в начало исходного файла с функцией `MakeUserCommand0()` добавьте команду для включения соответствующего заголовочного файла:

```
#include "mb_placement3d.h"
```

Повторите запуск тестового приложения и убедитесь, что точка отображается на оси X мировой СК. Измените обработчик так, чтобы на двух осях этой СК отображались точки различных цветов: красным на оси X и зеленым на оси Y (пример 4.1.3). Вместо константы PPOINT_Style теперь для каждой точки будет передаваться стиль, с помощью которого в тестовом приложении можно указать толщину линии (в пикселях, а не в единицах геометрической СК) и цвет для отображения геометрического объекта. Запустите приложение и увеличьте масштаб, чтобы убедиться в корректности отображения двух точек (рис. 6).

Пример 4.1.3. Построение двумерных точек с указанием стиля отображения

```
void MakeUserCommand0()
{
  // Система координат - по умолчанию совпадает с мировой СК
  MbPlacement3D pl;
  // Создание двух двумерных точек на осях X и Y
```

```
  MbCartPoint pntX( 1, 0 ), pntY( 0, 1 );
  // Добавление точек в модель с назначением им стилей отображения различными цветами
  viewManager->AddObject( Style(5, RGB(255,0,0)), pntX, &pl );
  viewManager->AddObject( Style(5, RGB(0,255,0)), pntY, &pl );
}
```

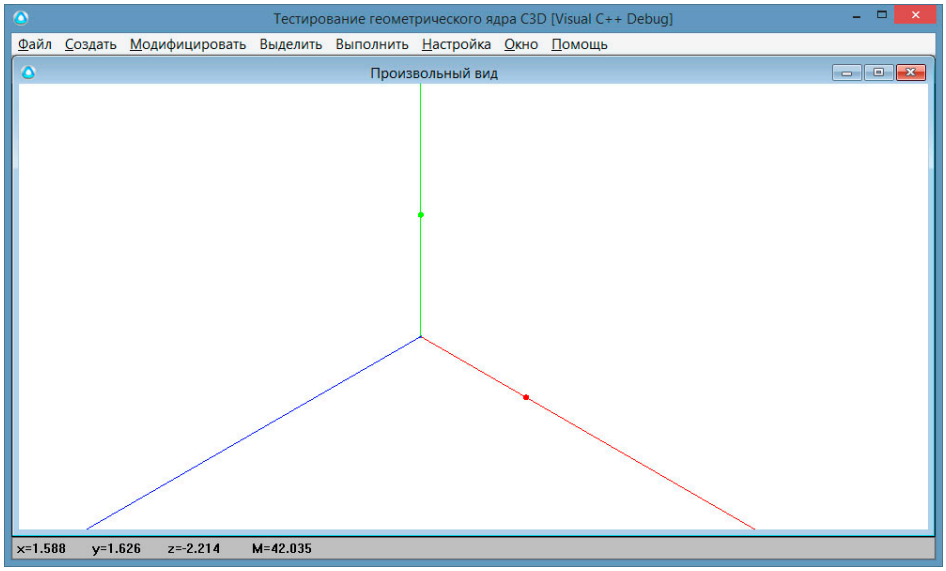

Рис. 6. Отображение пары точек на осях X и Y мировой СК (пример 4.1.3).

Теперь рассмотрим, как можно изменить ориентацию системы координат, в которой выполняется построение двумерных точек (пример 4.1.4). Используем локальную СК, полученную из мировой СК правым поворотом на 45° вокруг оси Z. Выполните приведенные далее вызовы и убедитесь, что радиус-вектор точки pntX будет повернут на 45° в плоскости XY, а точки pntY – на 135°.

Пример 4.1.4. Построение двумерных точек в локальной СК, повернутой относительно мировой СК

```
void MakeUserCommand0()
{
  // Создание локальной СК, повернутой на 45 градусов относительно оси Z мировой СК
  MbPlacement3D pl;
  MbAxis3D axisZ( MbVector3D(0, 0, 1) );
  pl.Rotate( axisZ, 45*M_PI/180 );

  // Создание двух двумерных точек на осях X и Y
  MbCartPoint pntX( 10, 0 ), pntY( 0, 10 );
  // Добавление точек в модель с указанием локальной СК
  viewManager->AddObject( Style(5, RGB(255,0,0)), pntX, &pl );
  viewManager->AddObject( Style(5, RGB(0,255,0)), pntY, &pl );
}
```

4.2 Трехмерные точки

Теперь изменим обработчик так, чтобы построить три точки на трех осях СК. Для этого вместо класса двумерной точки MbCartPoint потребуется использовать класс трехмерной точки MbCartPoint3D. При создании трехмерной точки используется мировая СК.

Каждая трехмерная точка будет добавляться в виде вершины, принадлежащей геометрическому объекту «точечный каркас» MbPointFrame. Потребуется три таких объекта, чтобы можно было каждому из них назначить собственный стиль отображения (рис. 7).

Пример 4.2.1. Построение трехмерных точек

```
void MakeUserCommand0()
{
  // Создание трех трехмерных точек на осях X, Y и Z
  MbCartPoint3D pntX( 1, 0, 0 ), pntY( 0, 1, 0 ), pntZ( 0, 0, 1 );
  // Добавление точек в модель с назначением им стилей отображения различными цветами
  // Каждая точка добавляется в виде "вырожденного" точечного каркаса, состоящего
  // из единственной точки
  viewManager->AddObject( Style(5, RGB(255,0,0)), new MbPointFrame(pntX) );
  viewManager->AddObject( Style(5, RGB(0,255,0)), new MbPointFrame(pntY) );
  viewManager->AddObject( Style(5, RGB(0,0,255)), new MbPointFrame(pntZ) );
}
```

Обратите внимание, что эти объекты MbPointFrame создаются динамически, но освобождения памяти не производится. При удалении геометрической модели MbModel, которая поддерживается внутри тестового приложения, все добавленные в нее геометрические объекты будут удалены автоматически (например, при закрытии приложения или при загрузке новой модели из файла *.c3d).

Для использования класса MbPointFrame добавьте в начало исходного файла команду включения соответствующего заголовочного файла:

```
#include "point_frame.h"
```

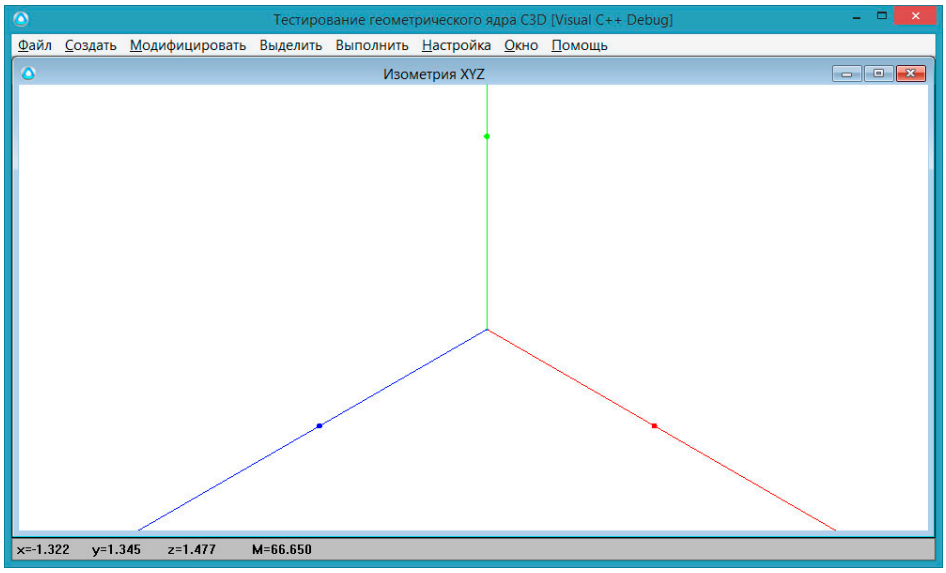

Рис. 7. Отображение трех точек на осях мировой СК – пример 4.2.1 (после увеличения масштаба).

Рассмотрим пример формирования точечного каркаса MbPointFrame, состоящего из некоторого множества точек. Выполните следующий пример, в котором выполняется размещение точек на плоскости XY мировой СК посредством равномерной выборки на одном периоде синусоиды.

Пример 4.2.2. Построение набора трехмерных точек в виде одного точечного каркаса

```
void MakeUserCommand0()
{
```

```
  // Динамическое создание геометрического объекта "Точечный каркас"
  MbPointFrame* pPnts = new MbPointFrame();

  // Добавление точек в каркас pPnts - равномерная выборка одного периода синусоиды
  const int STEP_COUNT = 10;
  for (int i = 0; i <= STEP_COUNT; i++)
  {
    double x = 2*M_PI/STEP_COUNT * i;
    MbCartPoint3D pnt( x, sin(x), 0 );
    pPnts->AddVertex( pnt );
  }

  // Добавление точечного каркаса в геометрическую модель
  viewManager->AddObject( Style(5, RGB(255,0,0)), pPnts );
}
```

Результат выполнения этого примера показан на рис. 8. На этом рисунке также показано окно свойств модели, вызываемое в тестовом приложении с помощью правой кнопки мыши. Чтобы вызвать это окно, надо щелкнуть правой кнопкой мыши в любом месте белого фона в окне построения и затем двойным щелчком открыть единственный объект геометрической модели – объект «Точечный каркас». (Это окно также можно сразу вызвать, щелкнув правой кнопкой на одной из точек каркаса.) В этом окне видно, что объект геометрической модели класса «Точечный каркас» содержит 11 вершин. Двойным щелчком на каждой вершине можно открыть окно для просмотра их свойств (например, координат). Среди свойств «Атрибуты объекта» можно убедиться, что в модели корректно запомнены атрибуты стиля отображения точечного каркаса – красный цвет и толщина линии, равная 5 пикселям.

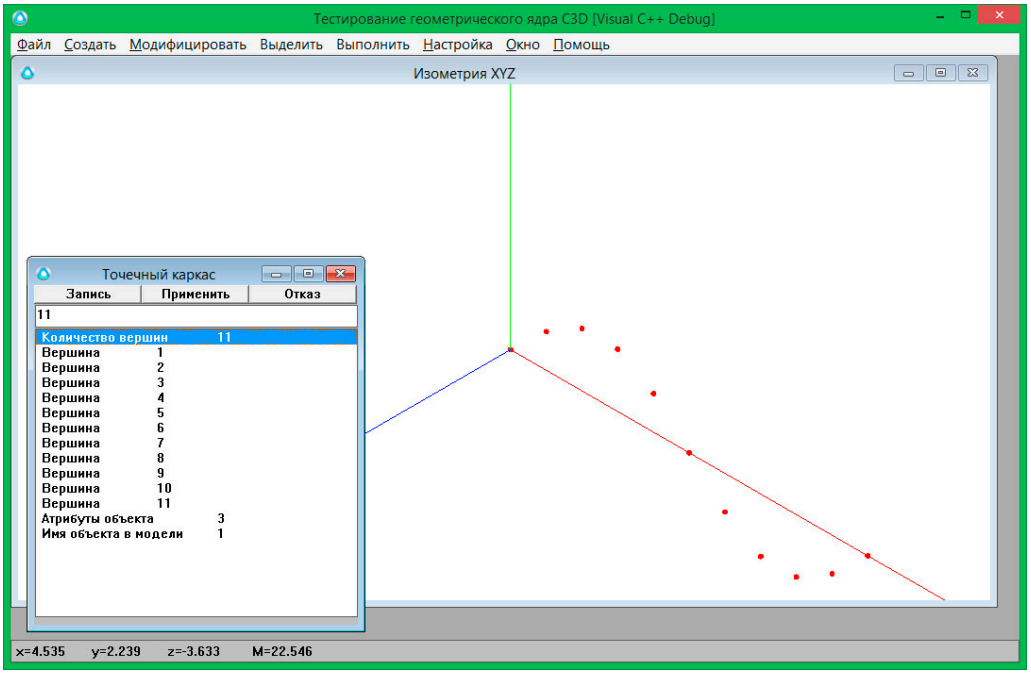

Рис. 8. Точечный каркас, содержащий 11 точек, и окно свойств для просмотра содержимого модели (пример 4.2.2).

4.3 Задания

1) Выполните все примеры построения двумерных и трехмерных точек из п. 4.1 и 4.2. Для каждого примера вызовите окно свойств геометрической модели и выясните, из каких объектов она состоит.

2) Воспользуйтесь справочной системой C3D и найдите описание классов, которые использовались при построении двумерных и трехмерных точек: MbCartPoint, MbCartPoint3D, MbPointFrame, MbPlacement3D, MbAxis3D, MbVector3D. Если при работе со справкой в HTML-формате не работают предусмотренные функции поиска, воспользуйтесь браузером Google Chrome.
3) При отображении точек (пример 4.2.2) в окне тестового приложения попробуйте двумя способами изменить атрибуты одной из точек. (*а*) Переместите точку с помощью мыши и в окне свойств геометрической модели просмотрите новые значения координат. (*б*) В окне свойств геометрической модели в атрибутах точечного каркаса измените стиль отображения – цвет и пиксельную толщину линии. После ввода нового значения в окне свойств модели обязательно нажмите кнопку **Применить**. Чтобы увидеть изменения, соответствующие введенным и примененным значениям атрибутов, в окне тестового приложения нажмите клавишу F9 для перерисовки модели.
4) Постройте точки, лежащие в вершинах четырехгранной пирамиды, основание которой лежит в плоскости XZ, а ось направлена вдоль оси Y.
5) На основе примера 4.2.2 реализуйте построение точечного каркаса с равномерным размещением точек по окружности с центром в начале координат, лежащей в плоскости XY. В окне свойств геометрической модели убедитесь, что в ней хранятся все построенные точки.

5. Построение кривых

Кривые являются базовыми геометрическими объектами ядра C3D. Они используются для построения поверхностей, для представления линий пересечения поверхностей, для построения вспомогательных объектов. В C3D есть два абстрактных базовых класса для представления двумерных и трехмерных кривых: MbCurve и MbCurve3D. В ядре имеются наборы классов, унаследованных от этих базовых классов, которые предназначены для представления кривых конкретного вида: прямых, отрезков, ломаных, эллипсов, сплайнов, и др. Допустимо описывать новые классы, унаследованные от базовых – их можно будет использовать для построения геометрических моделей C3D наряду с встроенными классами кривых.

Далее приведены несколько примеров, демонстрирующих построение простейших кривых с использованием готовых классов C3D. Для ознакомления с их действием, поместите приведенные программные фрагменты в обработчики пользовательских команд тестового приложения.

5.1 Прямые линии в двумерном пространстве

Прямая линия является частным простейшим видом кривых линий. Прямые часто используются в качестве вспомогательных объектов при построении геометрических моделей и решении геометрических задач. В C3D есть отдельные классы для построения прямых в двумерном и трехмерном пространстве. Сначала рассмотрим построение двумерных прямых.

В начало исходного файла добавьте команды для включения заголовочных файлов:

```
#include "curve.h"        // Класс двумерной кривой MbCurve
#include "cur_line.h"     // Класс двумерной прямой MbLine
```

В приведенном далее примере 5.1 показаны вызовы для построения пары прямых двумя различными способами – по двум точкам и по точке и направлению.

Пример 5.1 Построение прямой линии в двумерном пространстве

```
void MakeUserCommand0()
{
  MbPlacement3D pl; // Локальная СК (по умолчанию совпадает с мировой СК)
```

```
  // ПОСТРОЕНИЕ ДВУМЕРНОЙ ПРЯМОЙ – ПО ДВУМ ТОЧКАМ
  MbCartPoint p1(0, 0), p2(1, 1);            // Пара точек для построения прямой
  // Динамическое создание объекта-прямой посредством вызова конструктора MbLine
  MbLine* pLine1 = new MbLine( p1, p2 );
  // В случае успешного создания объекта-прямой этот объект передается для сохранения
  // в геометрической модели тестового приложения и для последующего отображения
  // (будет отображаться красным цветом)
  if ( pLine1 != NULL )
    viewManager->AddObject( Style( 1, RGB(255,0,0)), pLine1, &pl );
  // ПРИМЕЧАНИЕ: удаление pLine1 не производится - за это отвечает
  // объект - геометрическая модель внутри тестового приложения

  // ПОСТРОЕНИЕ ДВУМЕРНОЙ ПРЯМОЙ – ПО ТОЧКЕ И УГЛУ
  // Коэффициент для преобразования значений углов из градусов в радианы
  const double DEG_TO_RAD = M_PI/180.0;
  // Точка на оси Y
  MbCartPoint pnt( 0, 5 );
  // Направление (нормированный вектор) 15 градусов относительно оси X
  MbDirection dir( 15*DEG_TO_RAD );
  // Создание объекта-прямой с указанием точки и угла
  MbLine* pLine2 = new MbLine( pnt, dir );
  // Передача объекта-прямой тестовому приложению (будет отображаться синим цветом)
  if ( pLine2 )
    viewManager->AddObject( Style( 1, RGB(0,0,255)), pLine2, &pl );
}
```

В приведенном фрагменте демонстрируется типичный порядок действий по созданию геометрических объектов:

1) Создание вспомогательных объектов для представления параметров, необходимых для создания геометрического объекта (например, локальная СК, описание точек и векторов).
2) Динамическое создание геометрического объекта в виде объекта класса, унаследованного от базового MbPlaneItem (для двумерных объектов, в данном примере это класс MbLine) или MbSpaceItem (для трехмерных объектов).
3) Добавление созданного объекта в геометрическую модель (в данном примере это действие выполняется внутри метода тестового приложения AddObject).

Запустите тестовое приложение и выполните пользовательскую команду для построения пары прямых. Чтобы увидеть расположение прямых в плоскости XY (рис 1.9), выполните команду меню ***Окно ⇒ Ориентировать ⇒ Вид спереди***.

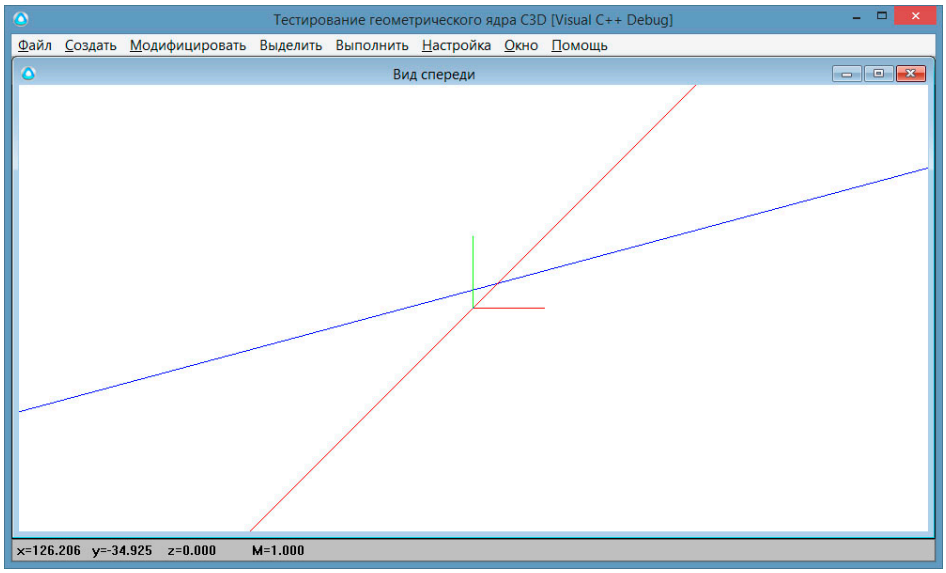

Рис. 9. Две прямые в плоскости XY, построенные с использованием класса двумерных прямых MbLine (пример 5.1).

5.2 Отрезки в двумерном пространстве

Двумерный отрезок представляется классом MbLineSegment (цепочка наследования: MbLineSegment→MbCurve→MbPlaneItem). Ниже приведен пример построения отрезка по двум точкам в плоскости XZ (указание плоскости построения выполняется с помощью локальной СК). Чтобы в тестовом приложении увидеть перпендикулярную проекцию плоскости XZ мировой СК, выберите команду меню ***Окно ⇒ Ориентировать ⇒ Вид сверху***.

Пример 5.2 Построение двумерного отрезка

```
#include "cur_line_segment.h"   // Класс двумерного отрезка MbLineSegment

void MakeUserCommand0()
{
  // Локальная СК (расположена так, чтобы ее плоскость XY совпадала с
  // плоскостью XZ мировой СК)
  MbPlacement3D pl;
  pl.SetAxisY( MbVector3D(0, 0, 1) );

  // Вершины отрезка
  MbCartPoint p1(5, 5), p2(15, 7);
  // Динамическое создание объекта-отрезка
  MbLineSegment* pSeg = new MbLineSegment( p1, p2 );
  // Сохранение и отображение в тестовом приложении
  // LINE_SEG_Style - стиль, принятый в тестовом приложении для отображения отрезков
  if ( pSeg )
       viewManager->AddObject( LINE_SEG_Style, pSeg, &pl );
}
```

5.3 Прямые линии в трехмерном пространстве

С помощью функций ядра C3D решим задачу нахождения точки пересечения пары прямых в трехмерном пространстве. Прямые будут строиться по двум точкам.

Если прямые пересекаются, то будет отображаться точка пересечения. Если прямые не пересекаются, то отображаются ближайшие точки этих прямых, а в окне сообщения показывается расстояние между этими точками.

Заголовочные файлы, которые потребуются для рассматриваемого примера:

```
#include "cur_line3d.h"   // Класс прямой MbLine3D
#include "point_frame.h"  // Класс MbPointFrame для отладочного отображения точек
#include "action_point.h" // Описание алгоритмов трехмерной вычислительной геометрии
```

Решение задачи показано ниже (пример 5.3). Первая прямая строится по точкам p1(0, 5, 0) и p2(3, 7, 5). Вторая прямая строится по точкам p3(0, 3, 0) и p4(3, –7, 5). Эти прямые пересекаются. Для удобного отображения пространственного построения, в тестовом приложении пользуйтесь клавишами для поворота модели и для изменения масштаба (рис. 10).

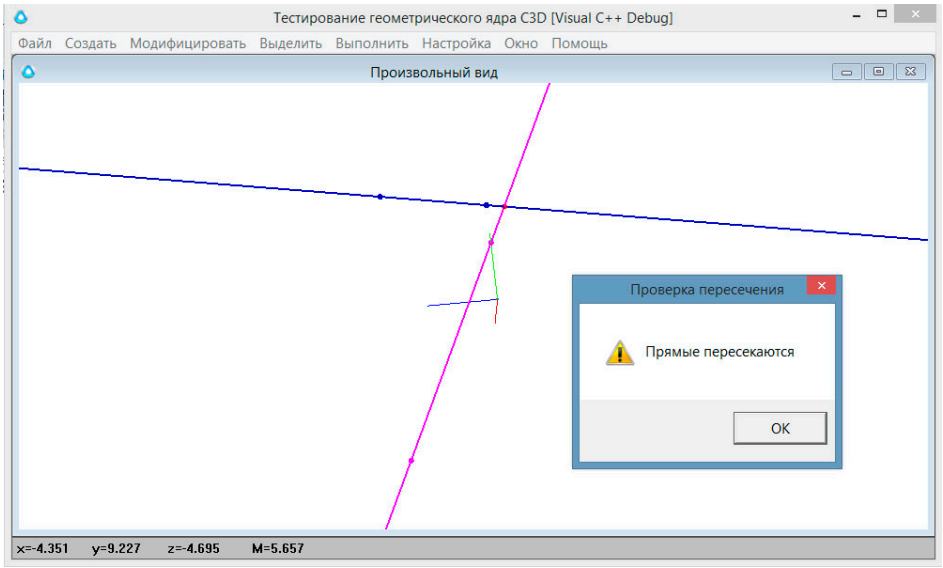

Рис. 10. Две пересекающиеся прямые. Показаны точки, по которым построены прямые, а также точка их пересечения (пример 5.3).

Пример 5.3 Построение прямых линий в трехмерном пространстве

```
// Построение пары прямых по двум точкам и проверка их пересечения
void MakeUserCommand0()
{
  // Построение трехмерной прямой по двум точкам
  MbCartPoint3D p1(0, 5, 0), p2(3, 7, 5);
  MbLine3D* pLine1 = new MbLine3D( p1, p2 );
  // Прямая и точки, по которым она построена, отображаются синим цветом
  if (pLine1)
  {
       viewManager->AddObject( Style( 2, BLUE ), pLine1 );
       viewManager->AddObject( Style( 1, BLUE ), new MbPointFrame(p1) );
       viewManager->AddObject( Style( 1, BLUE ), new MbPointFrame(p2) );
  }

  // Построение второй прямой по двум точкам
  MbCartPoint3D p3(0, 3, 0), p4(3, -7, 5);
  MbLine3D* pLine2 = new MbLine3D( p3, p4 );
  // Отображение второй прямой и пары ее точек пурпурным цветом
  if ( pLine2 )
```

```
  {
      viewManager->AddObject( Style( 2, LIGHTMAGENTA ), pLine2 );
      viewManager->AddObject( Style( 1, LIGHTMAGENTA ), new MbPointFrame(p3) );
      viewManager->AddObject( Style( 1, LIGHTMAGENTA ), new MbPointFrame(p4) );
  }

  // Применение алгоритма для нахождения ближайших точек двух прямых и для
  // вычисления расстояния между ними
  MbCartPoint3D pntRes1, pntRes2;
  double distMin = LineLineNearestPoints(  *pLine1, *pLine2, pntRes1, pntRes2 );

  // Поскольку точность компьютерных вычислений конечна, то для проверки
  // пересечения прямых минимальное расстояние следует сравнивать не с
  // точным нулевым значением 0.0, а с малой конечной величиной - допуском
  if (distMin > 1e-10 )
  {
      // Отображение красным цветом ближайших точек пары прямых
      viewManager->AddObject( Style( 1, LIGHTRED ), new MbPointFrame(pntRes1) );
      viewManager->AddObject( Style( 1, LIGHTRED ), new MbPointFrame(pntRes2) );

      // Формирование сообщения для информационного окна
      TCHAR sBuf[200];
      _stprintf( sBuf, _T("Расстояние между прямыми: %.3lf"), distMin );
      MessageBoxEx( sBuf, _T("Прямые не пересекаются") );
  }
  else
  {
      // Отображение красным цветом точки пересечения прямых
      viewManager->AddObject( Style( 1, LIGHTRED ), new MbPointFrame(pntRes1) );
      MessageBoxEx( _T("Прямые пересекаются"), _T("Проверка пересечения") );
  }
}
```

Попробуйте изменить точки, по которым строится вторая прямая, для построения скрещивающейся (например, p3(0, 3, 0) и p4(5, 0, 0)) и параллельной прямой (например, p3(0, 2.5, 0) и p4(1.5, 3.5, 2.5)).

5.4 Эллиптические дуги

Дуги эллипсов и окружностей в трехмерном пространстве представляются классом MbArc3D (цепочка наследования: MbArc3D→MbCurve3D→MbSpaceItem). В классе предусмотрены несколько конструкторов для построения дуги с использованием различных наборов параметров. В каноническом виде в декартовой системе координат эллипс с центром в начале координат описывается каноническим уравнением $\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$. В интерфейсах классов геометрических объектов поддерживается работа с параметрами канонических уравнений. Для расчетов в ядре используется общепринятый в области вычислительный геометрии подход с использованием параметрических уравнений. Возможно, оно менее наглядно для восприятия человеком, но обладает достоинствами с точки зрения организации вычислений. Классы, представляющие геометрические объекты, могут поддерживать оба способа работы с объектами – и в каноническом виде, и в параметрическом.

Далее рассмотрим, как построить несколько различных эллиптических дуг. В приведенном примере описываются окружность (эллипс с равными полуосями), эллипс (замкнутая эллиптическая дуга), и дуга эллипса (составляющая четверть от полного эллипса).
Заголовочный файл для подключения описания класса MbArc3D:

```
#include "cur_arc3d.h"
```

Пример 5.4 Построение эллиптических дуг

```
void MakeUserCommand0()
{
  // Коэффициент для преобразования значений углов из градусов в радианы
  const double DEG_TO_RAD = M_PI/180.0;

  // Построение окружности в плоскости XY с центром в точке (5, 0) и радиусом 2
  MbCartPoint3D pntCenter( 5, 0, 0 );
  // Пара точек на окружности
  MbCartPoint3D pntOnCircle1( 7, 0, 0 );
  MbCartPoint3D pntOnCircle2( 5, 2, 0 );
  // Вызываемый конструктор позволяет строить как окружности, так и дуги окружностей
  // (в зависимости от передаваемых параметров)
  // Четвертый параметр определяет смысл первого параметра – при n=0 его следует
  // рассматривать в качестве центра окружности
  // Пятый параметр при closed=true задает построение замкнутой окружности (а не дуги)
  MbArc3D* pCircle = new MbArc3D(pntCenter, pntOnCircle1, pntOnCircle2,
                                 0 /* n */, true /* closed */ );
  if (pCircle)
       viewManager->AddObject( Style( 1, LIGHTBLUE ), pCircle );

  // Построение эллипса в плоскости XY с поворотом осей на 45 градусов
  // pl - локальная система координат, в которой эллипс имеет каноническую форму
  // (в ней оси эллипса параллельны осям координат)
  MbPlacement3D pl;
  pl.Rotate( MbAxis3D(pl.GetAxisZ()), 45*DEG_TO_RAD );
  // Построение эллипса с указанием локальной СК и величин полуосей
  // Последний параметр задает конец дуги в случае построения эллиптической дуги,
  // для замкнутого эллипса должно быть angle=0
  MbArc3D* pEllipse = new MbArc3D( pl, 10, 5, 0 /* angle */ );
  if (pEllipse)
       viewManager->AddObject( Style( 1, LIGHTRED ), pEllipse );

  // Построение четверти эллиптической дуги для уже построенного эллипса.
  // Начало и конец дуги расположены под углами 60 и 150 градусов относительно
  // большой полуоси эллипса.
  // Последний параметр initSense=1 задает направление движения против часовой стрелки
  // от начальной до конечной точки дуги.
  MbArc3D* pQuarterArc = new MbArc3D( *pEllipse, 60*DEG_TO_RAD,
                                      150*DEG_TO_RAD, 1 /* initSense */ );
  if (pQuarterArc)
       viewManager->AddObject( Style( 2, LIGHTMAGENTA ), pQuarterArc );
}
```

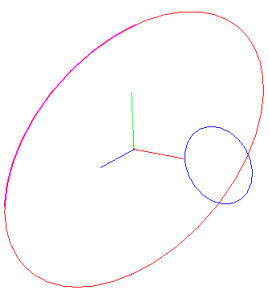

Рис. 11. Пример построения объектов класса MbArc3D: окружность, эллипс и дуга эллипса (пример 5.4).

Результаты работы примера 5.4 показаны на рис. 11. Найдите в справочной системе описание конструкторов MbArc3D, использовавшихся в приведенном примере и уточните назначение параметров этих конструкторов.

5.5 Пространственные кривые

Ядро C3D содержит более 10 классов-кривых, унаследованных от базового MbCurve3D. До сих пор рассматривались классы для построения плоских кривых. Однако кривые совершенно не обязательно должны быть плоскими, и в качестве примера рассмотрим построение пространственной кривой.

С использованием алгоритмов ядра решим задачу по обнаружению точек пересечения конической спирали и заданной плоскости. Коническая спираль и вычисленные точки пересечения будут отображаться в тестовом приложении (рис. 12).

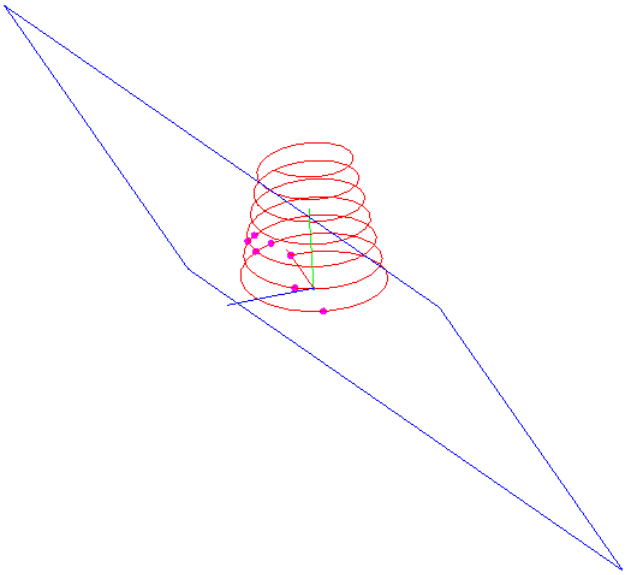

Рис. 12. Пример построения пространственной кривой MbConeSpiral и вычисления ее точек пересечения с плоскостью (пример 5.5).

Потребуется использовать классы: коническая спираль (цепочка наследования MbConeSpriral → MbSpiral → MbCurve3D), плоскость (MbPlane → MbElementarySurface → MbSurface → MbSpaceItem), точка MbCartPoint3D. Для вычисления точек пересечения поверхности и кривой будет использоваться алгоритм IntersectionPoints из набора операций с точками.

Подключаемые заголовочные файлы:

```
#include "mb_placement3d.h"           // Система координат MbPlacement3D
#include "cur_cone_spiral.h"          // Коническая спираль MbConeSpiral
#include "cur_polyline.h"             // Ломаная двумерная линия MbPolyline
#include "cur_polyline3d.h"           // Ломаная линия MbPolyline3D
#include "mb_axis3d.h"                // Ось в пространстве MbAxis3D
#include "surf_plane.h"               // Плоскость MbPlane
#include "point_frame.h"              // Точечный каркас MbPointFrame
#include "action_point.h"             // Операции с точками
```

Пример 5.5 Трехмерная кривая и ее точки пересечения с заданной плоскостью

```
void MakeUserCommand0()
{
  // Построение конической спирали, ось которой направлена вдоль оси Y мировой СК.
```

```
  // При построении ось конической спирали направлена вдоль оси Z локальной СК.
  // Поэтому требуется локальная СК, у которой ось Z направлена вдоль оси Y мировой СК.
  MbPlacement3D plSpiral( MbVector3D(1, 0, 0),   /* Ось X локальной СК */
                          MbVector3D(0, 0, -1),  /* Ось Y локальной СК */
                          MbCartPoint3D(0,0,0) /* Начало координат локальной СК */ );

  // Создание конической спирали.
  // Параметры конструктора:
  //   1) Локальная СК, в направлении +Z которой будет ориентирована ось спирали
  //   2) Радиус нижнего основания усеченного конуса, на боковой поверхности
  //      которого располагается спираль. Основание конуса располагается в
  //      плоскости XY локальной СК
  //   3) Радиус верхнего основания усеченного конуса
  //   4) Высота конуса
  //   5) Шаг спирали
  MbConeSpiral* pSpiral = new MbConeSpiral( plSpiral, 5, 3, 10, 1.5 );
  // Добавление в модель и отображение спирали красным цветом
  if (pSpiral)
       viewManager->AddObject( Style( 1, LIGHTRED ), pSpiral );

  // Создание плоскости - она совпадает с плоскостью XY локальной СК.
  // Локальная СК для плоскости - мировая СК, повернутая на 45 градусов вокруг оси X
  MbPlacement3D plPlane( MbVector3D(1,0,0), MbVector3D(0, 1, 1), MbCartPoint3D(0,0,0));
  MbPlane* pPlane = new MbPlane( plPlane );
  if (pPlane)
  {
       // Для отображения плоскости построим в виде ломаной линии прямоугольник
       // со стороной 40
       MbRect rRect(-20, -20, 20, 20); // Двумерные координаты вершин прямоугольника
       // Прямоугольник на плоскости в виде двумерной ломаной линии.
       // Это вспомогательный геометрический объект, который не добавляется в модель.
       // Поэтому он создается в виде автоматической стековой переменной, а не в виде
       // динамического объекта
       MbPolyline rect2D ( rRect );
       // Прямоугольник в трехмерном пространстве, лежащий в плоскости plPlane
       MbPolyline3D* pRect3D = new MbPolyline3D( rect2D, plPlane );
       // Добавление прямоугольника в модель и отображение синим цветом
       if (pRect3D)
         viewManager->AddObject(Style(1, LIGHTBLUE), pRect3D );
  }

  // Вычисление точек пересечения спирали и плоскости.
  // Результат возвращается в виде значений параметров параметрических уравнений
  // плоскости и спирали, которые соответствуют точкам пересечения.
  SArray<MbCartPoint> uvPlane;
  SArray<double> ttSpiral;
  IntersectionPoints( *pPlane, true, *pSpiral, false, uvPlane, ttSpiral );
  // Количество точек пересечения
  int pntCnt = ttSpiral.Count();
  // Отображение точек пересечения
  for (int i = 0; i < pntCnt; i++)
  {
       // Вычисление декартовых координат i-й точки пересечения посредством
       // подстановки параметра ttSpiral[i] в параметрическое уравнение спирали.
       MbCartPoint3D pnt;
       pSpiral->_PointOn(ttSpiral[i], pnt);
       viewManager->AddObject( Style(2, LIGHTMAGENTA), new MbPointFrame(pnt) );
  }
}
```

5.6 Задания

1) В тестовое приложение добавьте обработчики пользовательских команд для выполнения примеров, рассмотренных в п. 5.1-5.5.
2) На плоскости постройте множество из 10 прямых, пересекающихся в одной точке, с одинаковым угловым шагом между соседними прямыми (используйте класс MbLine, конструктор с указанием точки и угла). В тестовом приложении обеспечьте отображение прямых и точки пересечения.
   Замечание. В тестовом приложении при просмотре прямых в окрестности точки пересечения не используйте слишком большой масштаб. Например, при масштабном коэффициенте более 100000 (значение коэффициента M выводится в строке состояния) при отображении могут проявляться ошибки приближенного представления экранных координат. Этот эффект не связан с точностью вычислений в ядре C3D и является следствием упрощенного способа отображения геометрических объектов в тестовом приложении.
3) Постройте треугольник (с использованием класса двумерных отрезков MbLineSegment) и вписанную в него окружность (MbArc). Центр вписанной окружности лежит на пересечении биссектрис внутренних углов. Построение окружности можно разделить на три этапа: (1) построение биссектрис любых двух углов треугольника; (2) нахождение точки пересечения биссектрис; (3) вычисление радиуса вписанной окружности.
   Для построения биссектрис используйте функцию LineBisector (из набора операций по построению кривых в двумерном пространстве, заголовочный файл alg_curve_distance.h). Для поиска точки пересечения биссектрис используйте функцию LineLine (заголовочный файл action_point.h), а для вычисления радиуса окружности – метод DistanceToLineSeg класса MbCartPoint.
   Замечание о функции LineBisector. При пересечении пары прямых образуются две пары смежных углов. Указание угла, для которого надо получить биссектрису с помощью LineBisector, выполняется за счет порядка указания прямых-сторон угла. Например, в треугольнике $\mathbf{P_1P_2P_3}$ требуется получить биссектрису угла при вершине $\mathbf{P_1}$. Тогда при вызове LineBisector надо передать следующие параметры: (а) вершину угла – точку $\mathbf{P_1}$; (б) прямую $\mathbf{P_1P_2}$, построенную по точкам $\mathbf{P_1}$ и $\mathbf{P_2}$ (в конструктор прямой MbLine эти точки должны быть переданы именно таком порядке); (в) прямую $\mathbf{P_1P_3}$, построенную по точкам $\mathbf{P_1}$ и $\mathbf{P_3}$; (г) прямую-выходной параметр, в виде которого будет возвращена вычисленная биссектриса угла.
   Замечание о функции LineLine. При вызове LineLine надо в качестве параметров передать две прямые (в данном примере – биссектрисы двух углов треугольника) и выходной параметр – двумерную точку – для получения вычисленной точки пересечения прямых. Эта функция возвращает значение 1, если прямые пересекаются, или 0, если они параллельны или совпадают.
   Замечание о методе MbCartPoint::DistanceToLineSeg. Этот метод предназначен для вычисления расстояния от точки до отрезка. Допустим, вершины стороны треугольника $\mathbf{P_1P_2}$ представлены в виде объектов MbCartPoint с именами p1 и p2, а центр вписанной окружности хранится в виде объекта MbCartPoint с именем cnt. Тогда вызов метода будет иметь вид: cnt.DistanceToLineSeg(p1, p2). Полученное значение можно использовать в качестве искомого радиуса вписанной окружности.

4) Постройте куб из отрезков, представляющих его стороны, с использованием класса MbLineSegment3D.
5) Постройте набор из нескольких параллельных прямых, расположенных на равном расстоянии друг от друга, с использованием класса MbLine3D.
6) Постройте два эллипса с непараллельными осями, пересекающиеся в четырех точках (класс MbArc3D). Вычислите и отобразите точки пересечения эллипсов (функция CurveCurveIntersection).

7) Постройте коническую спираль (MbConeSpiral) и эллипс (MbArc3D). Найдите ближайшие точки этих кривых (функция NearestPoints, заголовочный файл curve3d.h). Отобразите кривые и найденные точки. Постройте отрезок между этими точками. Вычислите расстояние между точками и отобразите его в диалоговом окне. Вычислите длину отрезка, пользуясь данными из окна свойств геометрической модели.

6. Построение поверхностей

Поверхности, как и кривые, относятся к базовым объектам геометрической модели ядра C3D. Для описания поверхностей используются параметрические уравнения, зависящие от двух параметров (обычно обозначаются *u* и *v*). Базовым классом для представления поверхностей в трехмерном пространстве является класс MbSurface. Он унаследован от класса «Геометрический объект в трехмерном пространстве» MbSpaceItem. Наследование от MbSurface обеспечивает унифицированную обработку поверхностей функциями ядра (например, для реализации вычислительных алгоритмов определения пересечений геометрических объектов, расстояний между ними и т.п.)

В C3D имеется набор готовых классов, унаследованных от MbSurface и предназначенных для представления различных поверхностей. При необходимости можно разрабатывать новые классы, унаследованные от MbSurface.

6.1 Элементарные поверхности

Рассмотрим построение нескольких различных поверхностей из набора элементарных поверхностей. К ним относятся поверхности, для которых известны непараметрические аналитические уравнения, представляемые в каноническом виде в локальной СК. Базовым классом для элементарных поверхностей является MbElementarySurface (цепочка наследования: MbElementarySurface → MbSurface → MbSpaceItem). В ядре представлены следующие унаследованные классы элементарных поверхностей: MbPlane (плоскость, этот класс уже использовался в примере 5.5), MbConeSurface (коническая поверхность), MbCylinderSurface (цилиндрическая поверхность), MbSphereSurface (сферическая поверхность), MbTorusSurface (тороидальная поверхность).

Для построения элементарных поверхностей можно использовать такой же подход, как в рассмотренных выше примерах – явное создание объектов класса с вызовом подходящего конструктора. Конструкторов в каждом классе-поверхности, как правило, несколько, из которых можно выбрать наиболее подходящий для решаемой задачи. Вторым способом создания объектов-поверхностей является использование функции ядра ElementarySurface() (входит в набор функций построения поверхностей, заголовочный файл action_surface.h). Преимуществом второго способа является то, что функции ядра по созданию геометрических объектов (в т.ч. ElementarySurface) выполняют внутреннюю диагностику успешности процедуры построения, и возвращают результат в виде значения MbResultType. Тип MbResultType используется в ядре в качестве кода выполнения геометрических операций (код успешного выполнения rt_Success). Использование функций, возвращающих значения MbResultType, упрощает обнаружение и обработку логических ошибок.

В приведенном ниже примере 6.1 демонстрируется создание нескольких элементарных поверхностей. В плоскости XZ мировой СК создается плоская поверхность. На ней размещаются две сферические, две цилиндрические и две конические поверхности. Поверхности каждого типа создаются двумя различными способами (они отображаются различными цветами).

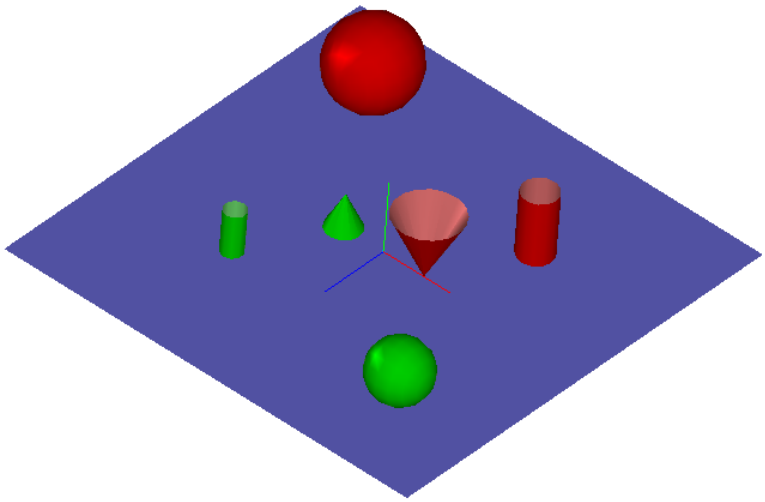

Рис. 13. Построение элементарных поверхностей MbSphereSurface, MbCylinderSurface и MbConeSurface (пример 6.1).

Пример 6.1 Построение элементарных поверхностей

```
#include "mb_axis3d.h"                // Ось в пространстве MbAxis3D
#include "mb_placement3d.h"           // Система координат MbPlacement3D
#include "point_frame.h"              // Точечный каркас MbPointFrame
#include "cur_line_segment3d.h"       // Отрезок MbLineSegment3D
#include "surf_plane.h"               // Плоскость MbPlane
#include "surf_sphere_surface.h"      // Сферическая поверхность MbSphereSurface
#include "surf_cylinder_surface.h"    // Цилиндрическая поверхность MbCylinderSurface
#include "surf_cone_surface.h"        // Коническая поверхность MbConeSurface
#include "action_surface.h"           // Функции построения поверхностей


void MakeUserCommand0()
{
  // Коэффициент для преобразования значений углов из градусов в радианы
  const double DEG_TO_RAD = M_PI/180.0;

  // Построение плоскости, совпадающей с координатной плоскостью XZ.
  // Производится вызов конструктора плоскости по трем точкам.
  // В данном случае могут быть переданы произвольные точки, лежащие в плоскости XZ.
  // Значения координат влияют на размеры плоскости при отображении в
  // тестовом приложении.
  MbCartPoint3D planePnt1( 0, 0, 0 );
  MbCartPoint3D planePnt2( 50, 0, 0 );
  MbCartPoint3D planePnt3( 0, 0, 50 );
  MbPlane* pPlane = new MbPlane( planePnt1, planePnt2, planePnt3 );
  // Отображение плоскости синим цветом
  if ( pPlane )
    viewManager->AddObject( Style( 1, LIGHTBLUE ), pPlane );

  // Далее производится построение элементарных поверхностей, касающихся плоскости XZ
  // и лежащих в ее положительном полупространстве.

  // Построение сферы посредством явного вызова конструктора с указанием центра и
  // радиуса сферической поверхности
  double radius_Sph1 = 10;
  MbCartPoint3D centerPnt_Sph1( -radius_Sph1*3, radius_Sph1, -radius_Sph1*3 );
  MbSphereSurface* pSphere1 = new MbSphereSurface( centerPnt_Sph1, radius_Sph1 );
```

```
if ( pSphere1 )
     viewManager->AddObject( Style( 1, LIGHTRED ), pSphere1 );

// Построение сферы вызовом функции ядра для создания элементарной поверхности
// Назначение точек-параметров:
//   1) начало локальной СК поверхности;
//   2) точка, определяющая направление оси X локальной СК и радиус поверхности;
//   3) точка, определяющая направление оси Y локальной СК.
// Параметры задаются так, чтобы центры двух сфер располагались по разные стороны и
// на равном расстоянии от оси OY мировой СК
MbSurface* pSphere2 = NULL;
double radius_Sph2 = 7;
MbCartPoint3D centerPnt_Sph2( -centerPnt_Sph1.x, radius_Sph2, -centerPnt_Sph1.z );
MbCartPoint3D pntOX_Sph2 = centerPnt_Sph2 + MbVector3D( radius_Sph2, 0, 0 );
MbCartPoint3D pntOY_Sph2 = centerPnt_Sph2 + MbVector3D( 0, radius_Sph2, 0 );
MbResultType resSph2 = ::ElementarySurface( centerPnt_Sph2, pntOX_Sph2,
                                            pntOY_Sph2, st_SphereSurface, pSphere2 );
if ( resSph2 == rt_Success )
     viewManager->AddObject( Style( 1, LIGHTGREEN ), pSphere2 );

// Построение первой цилиндрической поверхности
// Вызов конструктора с указанием центра локальной СК, радиуса основания и высоты.
// Ось цилиндра ориентируется вдоль оси Z локальной СК
double radius_Cyl1 = 4;
double height_Cyl1 = 15;
MbPlacement3D plCyl1;
// Локальная СК цилиндра: мировая СК смещается на заданный вектор и поворачивается
// так, чтобы ось Z локальной СК была параллельна оси OY мировой СК
plCyl1.Rotate(MbAxis3D(MbVector3D(1,0,0)), -90*DEG_TO_RAD );
plCyl1.Move( MbVector3D( radius_Cyl1*5, 0, -radius_Cyl1*5 ) );
MbCylinderSurface* pCyl1 = new MbCylinderSurface( plCyl1, radius_Cyl1, height_Cyl1 );
if ( pCyl1 )
     viewManager->AddObject( Style( 1, LIGHTRED ), pCyl1 );

// Построение второй цилиндрической поверхности
// Вызов функции ядра для создания элементарной поверхности
MbSurface* pCyl2 = NULL;
double radius_Cyl2 = 2.5;
double height_Cyl2 = 10;
// Центр нижнего основания цилиндра (начало координат локальной СК) получаем
// отражением центра основания первого цилиндра относительно оси OY мировой СК
MbCartPoint3D centerPnt_Cyl2 = plCyl1.GetOrigin();
centerPnt_Cyl2.x = -centerPnt_Cyl2.x;
centerPnt_Cyl2.z = -centerPnt_Cyl2.z;
// Точка, задающая ось X локальной СК и определяющая высоту цилиндра
// В данном случае ось X локальной СК параллельна оси Y мировой СК
MbCartPoint3D pntOX_Cyl2 = centerPnt_Cyl2 + MbVector3D( 0, height_Cyl2, 0 );
// Точка, задающая ось Y локальной СК и определяющая радиус основания
// В данном случае ось Y локальной СК параллельна оси Z мировой СК
MbCartPoint3D pntOY_Cyl2 = centerPnt_Cyl2 + MbVector3D( 0, 0, radius_Cyl2 );
MbResultType resCyl2 = ::ElementarySurface( centerPnt_Cyl2, pntOX_Cyl2,
                                            pntOY_Cyl2, st_CylinderSurface, pCyl2 );
if ( resCyl2 == rt_Success )
     viewManager->AddObject( Style( 1, LIGHTGREEN ), pCyl2 );

// Построение первой конической поверхности
// Эта поверхность касается плоскости XZ мировой СК в одной точке
// Вызывается конструктор конической поверхности по трем точкам.
double height_Cone1 = 15;
double radius_Cone1 = 7.5;
// Вершина конической поверхности
MbCartPoint3D pntCone1_origin( 10, 0, 0 );
```

```
  // Центр основания
  MbCartPoint3D pntCone1_base = pntCone1_origin + MbVector3D(0, height_Cone1, 0 );
  // Точка на боковой поверхности в плоскости основания
  MbCartPoint3D pntCone1_surf = pntCone1_base + MbVector3D(radius_Cone1, 0, 0 );
  MbConeSurface* pCone1 = new MbConeSurface( pntCone1_origin, pntCone1_base,
                                             pntCone1_surf );
  if (pCone1)
       viewManager->AddObject( Style( 1, LIGHTRED ), pCone1 );

  // Построение второй конической поверхности
  // Основание этой поверхности лежит на плоскости XZ мировой СК
  double height_Cone2 = 8;
  double radius_Cone2 = 4;
  // Вершина конической поверхности
  MbCartPoint3D pntCone2_origin( -pntCone1_origin.x, height_Cone2, 0 );
  // Центр основания
  MbCartPoint3D pntCone2_base = pntCone2_origin + MbVector3D(0, -height_Cone2, 0 );
  // Точка на боковой поверхности в плоскости основания
  MbCartPoint3D pntCone2_surf = pntCone2_base + MbVector3D(radius_Cone2, 0, 0 );
  MbSurface* pCone2 = NULL;
  MbResultType resCone2 = ::ElementarySurface( pntCone2_origin, pntCone2_base,
                                               pntCone2_surf, st_ConeSurface, pCone2 );
  if ( resCone2 == rt_Success )
       viewManager->AddObject( Style( 1, LIGHTGREEN ), pCone2 );

  // Уменьшение счетчика ссылок на динамически созданные объекты.
  // При достижении счетчиком 0 объект будет удален вызовом деструктора.
  ::DeleteItem( pPlane );
  ::DeleteItem( pSphere1 );
  ::DeleteItem( pSphere2 );
  ::DeleteItem( pCyl1 );
  ::DeleteItem( pCyl2 );
  ::DeleteItem( pCone1 );
  ::DeleteItem( pCone2 );
}
```

Результат работы обработчика из примера 6.1 показан на рис. 13. Убедитесь, что поверхности касаются плоскости XZ, поворачивая модель в окне тестового приложения. С помощью клавиши F8 в тестовом приложении переключите режимы отображения поверхностей между режимами триангуляции и тонированном режимом. С помощью окна свойств геометрической модели определите типы построенных геометрических объектов и их свойства.

В завершении обработчика обратите внимание на вызовы функции ядра DeleteItem. Выше отмечалось, что динамически созданные геометрические объекты, добавленные в модель C3D, будут автоматически удалены при создании модели. Несмотря на это, для предотвращения случайных утечек памяти лучше следовать правилу вызова функций удаления динамически созданных геометрических объектов при выходе из области видимости указателя на этот объект. Для этой цели следует использовать функцию ядра DeleteItem, а не оператор Си++ delete. Функция DeleteItem корректно работает с механизмом подсчета ссылок на динамические объекты и вызывает деструктор тогда, когда счетчик ссылок становится равным 0. После вызова DeleteItem переданный указатель на геометрический объект возвращается равным значению NULL.

6.2 Операции с поверхностями

Как и для кривых, в ядре C3D для работы с поверхностями предусмотрен большой набор геометрических алгоритмов. Они реализованы в виде глобальных функций ядра, например, функции построения поверхностей (заголовочный файл action_surface.h) и функции вычислений с кривыми и поверхностями в трехмерном пространстве (action_surface_curve.h).

Типичной операцией с поверхностями является определение линии пересечения двух поверхностей. Результатом выполнения этой операции могут быть одна или несколько кривых. Для их представления в ядре предусмотрен специальный класс MbSurfaceIntersectionCurve (cur_surface_intersection.h).

В качестве примера выполнения операций с поверхностями рассмотрим задачу определения линии пересечения сферы и плоскости (это окружность, рис. 14).

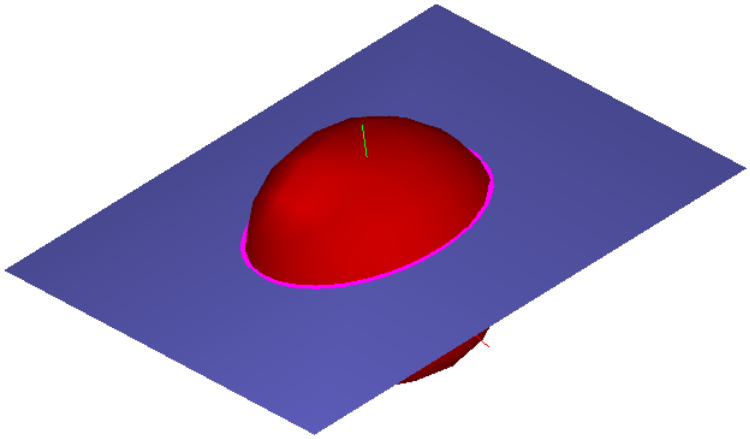

Рис. 14. Сечение сферической поверхности плоскостью (пример 6.2).

Пример 6.2. Определение линии пересечения сферической поверхности и плоскости

```
#include "mb_placement3d.h"            // Система координат MbPlacement3D
#include "curve3d.h"                   // Кривая в трехмерном пространстве MbCurve3D
#include "surf_plane.h"                // Плоскость MbPlane
#include "surf_sphere_surface.h"       // Сферическая поверхность MbSphereSurface
#include "action_surface_curve.h"      // Функции построения трехмерных кривых
#include "wire_frame.h"                // Проволочный каркас MbWireFrame

void MakeUserCommand0()
{
  // Сфера с центром в начале координат
  double radius_Sphere = 15;
  MbPlacement3D plSphere;
  MbSphereSurface* pSphere = new MbSphereSurface( plSphere, radius_Sphere );
  if ( pSphere )
       viewManager->AddObject(Style(1, LIGHTRED), pSphere);

  // Плоскость, пересекающая сферу
  MbCartPoint3D planePnt1( 0, 5, 0 );
  MbCartPoint3D planePnt2( 30, 15, 0 );
  MbCartPoint3D planePnt3( 30, 15, 30 );
  MbPlane* pPlane = new MbPlane( planePnt1, planePnt2, planePnt3 );
  if ( pPlane )
    viewManager->AddObject( Style( 1, LIGHTBLUE ), pPlane );

  // Генератор имен для именования геометрических объектов
  MbSNameMaker snMaker( -1, MbSNameMaker::i_SideNone, 0 );
  // Проволочный каркас для приема результатов определения пересечений поверхностей
```

```
  MbWireFrame* pTmpFrame = NULL;
  // Массив указателей на объекты класса MbCurve3D
  RPArray<MbCurve3D> arrCurves( 4 /* Начальная емкость массива */ );
  // Вызов функции для вычисления кривых пересечения двух поверхностей
  MbResultType res = ::IntersectionCurve( *pSphere, *pPlane, snMaker, pTmpFrame );
  if (res == rt_Success)
  {
       // Извлечение результатов из проволочного каркаса в виде массива
       // указателей на кривые. В случае сечения сферы плоскостью там
       // должна оказаться одна кривая.
       ::ExtractCurvesDeleteFrame( pTmpFrame, arrCurves );
       for (int i=0; i < arrCurves.size(); i++)
             viewManager->AddObject(Style(5, LIGHTMAGENTA), arrCurves[i] );
  }
}
```

В обработчике, показанном в примере 6.2, обратите внимание на организацию вызова функции ядра ::IntersectionCurve для вычисления кривых пересечения. Перед вызовом выполняется подготовка вспомогательных объектов, которые необходимы данной функции, а после вызова производится извлечение результатов обработки в виде набора указателей на геометрические объекты (указатели на объекты MbCurve3D).

Кривая пересечения возвращается в виде указателя на базовый класс MbCurve3D. Попробуйте вызвать у этой кривой следующие методы, предусмотренные в классе MbCurve3D:

```
MbeSpaceType curveType = arrCurves[0]->IsA();
bool isPlanar = arrCurves[0]->IsPlanar();
bool isClosed = arrCurves[0]->IsClosed();
```

Проверьте возвращаемые значения с помощью отладчика Visual Studio. Значение curveType должно оказаться равным st_SurfaceIntersectionCurve. Значит, указатель arrCurves[0] является указателем на класс пространственной кривой MbSurfaceIntersectionCurve. Этот класс позволяет представлять кривые пересечения поверхностей произвольного вида.

6.3 Задания

1) Постройте три пересекающихся плоскости, совпадающих с координатными плоскостями мировой СК. Отобразите эти плоскости различными цветами.
2) Постройте пересекающиеся коническую и цилиндрическую поверхности.
3) Постройте 5 непересекающихся тороидальных поверхностей с общим центром. Используйте класс MbTorusSurface или функцию ElementarySurface для построения поверхности типа st_TorusSurface.
4) Разработайте функцию drawFrame(const MbPlacement3D& pl, double axLength) для отображения системы координат. Параметрами функции являются объект-система координат и параметр axLength, задающий длину отрезков, изображающих оси. Для изображения осей используйте цилиндрические поверхности, в начало координат поместите сферическую поверхность, в качестве стрелок на осях используйте конические поверхности. Размеры этих поверхностей вычисляйте пропорционально значению axLength.
5) С использованием функции из задания (4) отобразите локальную систему координат, смещенную относительно мировой на вектор (10, 5, 3) и повернутую на 60° вокруг оси OX.
6) Постройте сечения конической поверхности плоскостями для получения конических сечений трех различных типов – эллипса, параболы и гиперболы. Постройте кривые пересечения, вычисленные с помощью функции ::IntersectionCurve.

7. Построение твердых тел

Выше рассматривались примеры построения точек, кривых и поверхностей. Это базовые геометрические объекты, которые в ядре C3D могут использоваться как в качестве самостоятельных элементов геометрической модели, так и в качестве вспомогательных элементов для построения более сложных объектов – в частности, твердых тел. В отличие от кривых и поверхностей, форму произвольного тела не удается описать с помощью одного уравнения. Для описания тела требуется комбинировать несколько разнотипных геометрических элементов. В ядре C3D для построения моделей формы тел используется подход B-Rep – граничное представление. В рамках этого подхода для представления тел используются геометрические объекты специальных типов, обладающие дополнительными по сравнению с кривыми и поверхностями свойствами (например, направлением). Эти свойства необходимы для построения корректной комбинации геометрических объектов, образующих геометрическую модель твердого тела в виде набора граней (грани могут иметь вид произвольных поверхностей), которые соединяются друг с другом вдоль ребер (ребра могут быть произвольными кривыми). Подробнее эти объекты и связанные с ними операции будут рассматривать в последующих работах

Модели тел в C3D представляются в виде объектов класса MbSolid. Кратко перечислим основные геометрические объекты, которые используются в качестве составных элементов при построении моделей тел.

1) Вершина (класс MbVertex) – точка, в которой стыкуется некоторое количество ребер.
2) Ребро (класс MbEdge) – сегмент кривой, которому приписан признак направления. У ребра есть начальная и конечная вершины MbVertex.
3) Грань (класс MbFace) – фрагмент поверхности, ограниченный замкнутой последовательностью ребер (эта последовательность называется циклом грани). Для грани указывается направление нормали, которое позволяет отличать две стороны грани – внешнюю и внутреннюю.
4) Оболочка (класс MbFaceShell) – набор граней, образующих связную поверхность. Каждое твердое тело MbSolid состоит из одной или нескольких оболочек MbFaceShell.

Одной из основных задач, решаемых ядром C3D, является построение и поддержание в корректном состоянии моделей тел, состоящих из перечисленных выше составных элементов. Это математическое численное описание формы объекта можно преобразовывать из представления B-Rep в другие виды – например, получать полигональное представление в виде набора треугольников (триангуляцию) для экранного отображения.

Ядро содержит реализацию всех необходимых вычислительных алгоритмов по созданию и компоновке объектов в составе граничных моделей тел. Для их использования необходимо соблюдать определенный порядок действий и правила по компоновке составных элементов модели и по их модификации (в случае, если созданная геометрическая модель подвергается модификации в программном или интерактивном режиме, что требуется во многих приложениях).

7.1 Элементарные твердые тела

В качестве первого примера по построению модели тела рассмотрим построение цилиндра. В примере 6.1 производилось построение цилиндрической поверхности MbCylinderSurface как одной из разновидностей элементарных поверхностей. Построенный таким образом объект моделью тела считать нельзя – явно заметно (рис. 13), что цилиндрическая поверхность представляет собой боковую поверхность цилиндра. Для построения модели цилиндра

необходимо создать геометрические объекты, представляющие основания цилиндра и скомбинировать их с боковой поверхностью так, чтобы они имели общие ребра (в виде окружностей). В терминах граничного представления тел, твердотельная модель цилиндра будет состоять из следующих элементов[2]:

1) Одна оболочка, содержащая три грани. Нормали граней заданы так, чтобы у всех трех граней видимыми были внешние стороны.
2) Три грани – боковая поверхность и два основания.
3) Три ребра (два ребра – окружности, третье прямолинейное ребро – шов, по которому замыкается боковая поверхность)
4) Две вершины – точки пересечения шва и ребер-окружностей (обе вершины лежат на основаниях цилиндра).

В ядре C3D применяется унифицированный подход к формированию наборов геометрических операций и классов в модули одного назначения. В п. 6.1 рассматривалось построение цилиндрической поверхности вызовом функции ядра ::ElementarySurface в виде объекта класса MbSurface. Для выполнения этой операции требовалось указывать вспомогательные объекты – точки в трехмерном пространстве, задающие размеры и положение требуемой поверхности. Для построения элементарных твердых тел в ядре также есть подобная функция ::ElementarySolid (заголовочный файл action_solid.h). Предусмотрены два варианта этой функции, которые отличаются типами передаваемых параметров. Одна из них предназначена для построения твердого тела на основе элементарной поверхности, а вторая – на основе массива точек. Применение этих функций демонстрируется в примере 7.1 для построения цилиндра (на базе элементарной поверхности) и пирамиды (по точкам).

Функция построения элементарного тела ::ElementarySolid скрывает выполнение всех необходимых операций. Для построения тел используются объекты-строители. В виде объектов классов-строителей реализованы алгоритмы построения твердых тел, которые генерируют или используют готовые вспомогательные объекты и возвращают указатель на новый объект-тело. Функция ::ElementarySolid пользуется объектом-строителем класса MbElementarySolid. Важно не путать классы-строители и классы, представляющие геометрические объекты. Классы-строители – это операции построения тел. Применяемые классы строители запоминаются в геометрической модели C3D, они фиксируются в журнале операций (в большинстве САПР твердотельного моделирования такие журналы обычно доступны пользователю, например, в виде дерева построения).

Выполните обработчик из примера 7.1. Построенные тела показаны на рис. 15 вместе с открытыми окнами свойств геометрической модели. Для первого тела – цилиндра – показано содержимое журнала построения, из которого следует, что функция ::ElementarySolid выполнила построение твердого тела с использованием операции вращения.

Пример 7.1. Построение элементарных твердых тел – цилиндра и пирамиды

```
#include "surface.h"            // Поверхность MbSurface
#include "solid.h"              // Твердое тело MbSolid
#include "action_surface.h"     // Функции построения поверхностей
#include "action_solid.h"       // Функции построения твердых тел
#include "creator.h"            // Построитель элементов твердотельных моделей

void MakeUserCommand0()
{
  // Построение цилиндрической поверхности
  double height_Cyl = 10;
  double radius_Cyl = 3;
  MbCartPoint3D baseCenter1( 0, 0, 0 );            // Центр первого основания
  MbCartPoint3D baseCenter2( 0, height_Cyl, 0 );   // Центр второго основания
```

[2] Подробнее математическое описание B-Rep модели цилиндра приведено в книге Голованов Н.Н. Геометрическое моделирование. М.: ИНФРА-М, 2016, стр. 248.

```
  // Точка на втором основании для указания радиуса цилиндра
  MbCartPoint3D pntOnBase2( radius_Cyl, height_Cyl, 0 );
  // Вызов функции ядра для создания элементарной поверхности
  MbSurface* pCylSurf = NULL;
  MbResultType resCylSurf = ::ElementarySurface( baseCenter1, baseCenter2, pntOnBase2,
                                                 st_CylinderSurface, pCylSurf );

  // Построение цилиндрического тела
  MbSolid* pCyl = NULL;
  if (resCylSurf == rt_Success)
  {
       // Вспомогательный объект для именования составных элементов твердого тела
       MbSNameMaker namesCyl( ct_ElementarySolid, MbSNameMaker::i_SideNone, 0 );
       // Вызов функции ядра для построения тела на основе элементарной поверхности
       MbResultType resSolid = ::ElementarySolid( *pCylSurf, namesCyl, pCyl );
       if ( resSolid == rt_Success )
             viewManager->AddObject( Style( 1, LIGHTRED ), pCyl );
  }

  // Построение тела по характерным точкам (четырехгранная равносторонняя пирамида)
  MbSolid* pPyr = NULL;
  // Массив характерных точек – 4 вершины основания + вершина пирамиды
  SArray<MbCartPoint3D> arrPnts(5);
  arrPnts.Add(MbCartPoint3D( -5, 0, -5 ));
  arrPnts.Add(MbCartPoint3D(  5, 0, -5 ));
  arrPnts.Add(MbCartPoint3D(  5, 0,  5 ));
  arrPnts.Add(MbCartPoint3D( -5, 0,  5 ));
  arrPnts.Add(MbCartPoint3D( 0, 15,  0 ));
  MbSNameMaker namesPyr( ct_ElementarySolid, MbSNameMaker::i_SideNone, 0 );
  // Вызов функции ядра для построения тела типа et_Pyramid по массиву точек
  MbResultType resPyr = ::ElementarySolid( arrPnts, et_Pyramid, namesPyr, pPyr );
  if (resPyr == rt_Success)
  {
       // Смещение твердого тела на заданный вектор
       pPyr->Move( MbVector3D( 15, 0, 0 ) );
       viewManager->AddObject( Style( 1, LIGHTBLUE ), pPyr );
  }

  // Уменьшение счетчиков ссылок на динамически созданные объекты
  ::DeleteItem( pCylSurf );
  ::DeleteItem( pCyl );
  ::DeleteItem( pPyr );
}
```

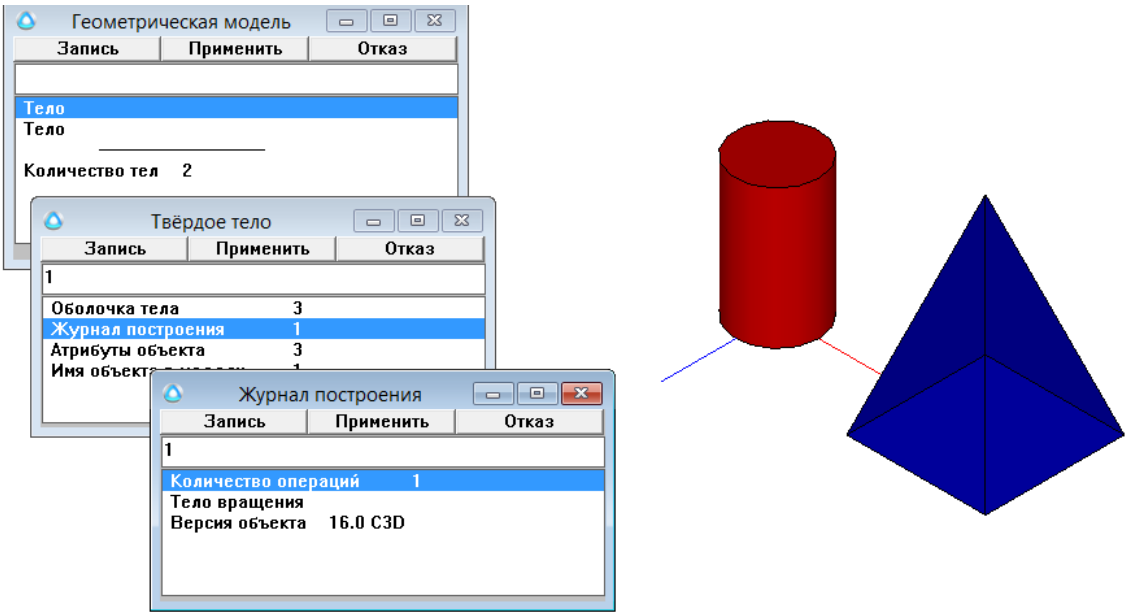

Рис. 15. Элементарные твердые тела – цилиндр и четырехгранная равносторонняя пирамида (пример 7.1).

7.2 Задания

1) Найдите в справочной системе описание двух функций ::ElementarySolid и выясните назначение передаваемых параметров.
2) На основе примера 7.1 выполните построение конуса на основе элементарной конической поверхности.
3) Постройте треугольную призму с помощью функции ::ElementarySolid, предназначенной для построения элементарного твердого тела по характерным точкам.
4) Разместите сферические твердые тела в вершинах произвольного куба. Все восемь сфер создавайте в одной и той же локальной СК (например, с центром в начале мировой СК) и перемещайте их в требуемое положение, вызывая метод Move класса MbSolid с указанием соответствующего вектора смещения (аналогично перемещению пирамиды в примере 7.1).

8. Заключение

Основной целью данной работы является знакомство со средой для работы с ядром C3D. В качестве такой среды используется тестовое приложение, входящее в состав поставки ядра. Для выполнения практических действий с ядром необходимо иметь возможность перекомпиляции этого приложения, в котором предусмотрены функции-обработчики команд пользователя, которые можно наполнять собственными программными фрагментами.

В работе были рассмотрены следующие основные вопросы:

- Назначение основных файлов и каталогов, входящих в состав поставки ядра C3D.
- Способы сборки тестового приложения на основе готового проекта из поставки ядра и посредством формирования нового проекта для среды MS Visual Studio 2017.
- Построение некоторых геометрических объектов в среде тестового приложения: точек, кривых, элементарных поверхностей и элементарных твердых тел.