МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Г.Г. Хайдаров, В.Т. Тозик

КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ ТРЕХМЕРНОГО МОДЕЛИРОВАНИЯ

Учебное пособие

Санкт-Петербург
2010

УДК 681.3

Хайдаров Г.Г., Тозик В.Т. Компьютерные технологии трехмерного моделирования.: Учебное пособие. - СПб.: СПбГУ ИТМО, 2010. - 80 с.

В данном учебном пособии к самостоятельным работам по дисциплине «Компьютерная геометрия и графика» собраны технологии трехмерного моделирования как с помощью графического редактора «Компас – 3D», так и с помощью сред и языков программирования.

Учебное пособие адресовано студентам 1 курса по направлению подготовки 230200 – Информационные системы и соответствуют рабочей программе дисциплины «Компьютерная геометрия и графика», существующей в СПбГУ ИТМО.

Ил. 55, табл.2, библиогр. 12 назв.

Рецензент: В.П. Большаков, доцент, кафедра прикладной механики и инженерной графики СПбГЭУ «ЛЭТИ» им. В.А. Ульянова (Ленина)

Рекомендовано к печати на ученом совете факультета точной механики и технологий, 11.05.2010 г., протокол № 8

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы.

Введение

Отправной точкой в развитии дисциплины «Компьютерная геометрия и графика» для студентов СПбГУ ИТМО можно считать 1984/85 учебный год, когда на кафедре инженерной и компьютерной графики началось чтение курса под названием «Машинная графика». Курс был ориентирован на ЕС ЭВМ и опубликован в 1986 году. Дальнейшее развитие данная дисциплина получила в процессе чтения курса под названием «Диалоговые системы и машинная графика», ориентированного на СМ ЭВМ и опубликованного в 1990 году. Третьим шагом в развитии этого направления явился курс под названием «Диалоговая геометрическая система и макросредства ДИОМ», ориентированный на персональные компьютеры серии IBM PC и опубликованный в 1991 году. В настоящее время только в сети русскоязычного интернета опубликовано более 20 курсов данного направления в основном на сайтах ведущих университетов России.

В соответствии с требованиями государственного образовательного стандарта по направлению подготовки 230200 – «Информационные системы» содержание дисциплины «Компьютерная геометрия и графика» (КГГ) охватывает круг вопросов, связанных с геометрическим моделированием, компьютерной генерацией и представлением чертежно-графической и видеоинформации, применением интерактивной графики в информационных системах, программированием «быстрой» графики.
К 2006 году в СПбГУ ИТМО был полностью сформирован курс, отвечающий требованиям дисциплины «Компьютерная геометрия и графика» для студентов факультета информационных технологий и программирования [1]. В процессе выполнения работ по данному курсу от студентов требуются не только теоретические знания по черчению, информатике, матеиматике, но и умения комплексного применения этих знаний. Основными работами для первого модуля первого курса являются работы по компьютерному черчению и моделированию в среде «Компас - 3D» . Подробности работы в этой среде можно посмотреть в учебной литературе [2 - 4], имеющейся в библиотеке. Самостоятельные работы второго модуля первого курса связаны с будущей специальностью студентов и основаны на университетских дисциплинах: информатика, программирование, высшая математика, инженерная графика. Кроме того, современное программирование не стоит на месте, а разрабатывает новые языки и визуальные среды. Для данного курса с 2008 года был выбран язык программирования C#, как вобравший в себя все новые идеи при программирование графики. В качестве среды программирования была выбрана среда Visual Studio (или ее модификация свободного распространения (Free Software) Visual Studio C# Express). Для подробного

изучения можно порекомендовать следующие книги по основам языка программирования и по среде программирования Visual Studio [5-12].

Данные методические указания включают набор примеров выполнения работ по всем базовым заданиям дисциплины «Компьютерная геометрия и графика». Дополнительные пояснения даются в местах наиболее часто встречаемых ошибок у студентов.

Часть 1. Теоретические основы

Глава 1. Основы выполнения чертежей в графическом редакторе «Компас -3D»

Применение графического редактора «Компас -3D» требует определенных практических навыков работы на компьютере, а также знания основных положений черчения, без которых теряется всякий смысл использования данного редактора. В данном пособии не ставится задача «с нуля» изучить «Компас -3D». При отсутствии общих знаний следует взять в библиотеке подробную литературу [2-4]. Целью данного раздела является обратить внимание учащихся на особенности использования графического редактора при создании чертежей и трехмерных моделей деталей.

В настоящее время существует два подхода к созданию чертежей. Первый – выполнение двумерного чертежа от начала до конца. Второй – выполнение трехмерной модели детали, а затем создание чертежа из ассоциативных видов трехмерной модели. В обоих случаях начальные построения одинаковы: прежде всего, это вычерчивание контуров детали. Для точной установки размеров контуров следует применять панель свойств и меню привязок.

Панель свойств

Построения линий, дуг, окружностей, эллипсов, кривых, многоугольников производится через компактную инструментальную панель «Геометрия»
Если вызвать команду вычерчивания отрезка прямой линии, нажав кнопку «Отрезок» на Инструментальной панели, то в «Строке параметров» появится несколько полей. Рассмотрим панель свойств на примере, свойства отрезка.

Рисунок 1.1. Панель свойств

Здесь
− т1 – координаты начальной точки,
− т2 – координаты конечной точки,

− Длина - длина отрезка,
− Угол – угол наклона отрезка,
− Стиль – стиль прорисовки линии на чертеже.

Для удобства перевода курсора в нужное поле ввода числового значения рекомендуется использовать комбинацию клавиш <Alt>+горячая клавиша (подчеркнутый символ в названии параметра), после ввода значения, необходимо его зафикировать, нажав клавишу <Enter>.

Слева от названия параметра находится небольшая кнопка. Если в ней отображается галочка, это означает, что система в настоящий момент ожидает ввода данного параметра, то есть он является умолчательным (поле координат начальной точки сразу после запуска команды построения отрезка). После того, как значение введено и параметр зафиксирован, на кнопке появляется изображение перекрестия. Если кнопка пустая, то параметр является вспомогательным (как длина и угол наклона в случае отрезка), при этом он доступен для ввода. Таким образом, Вы видите сразу все его характеристики и можете изменять любую из них непосредственно в процессе построения.

Для того чтобы получить оперативную информацию о поле параметра, поместите на него курсор. Через некоторое время рядом с курсором появится ярлычок-подсказка с названием параметра.

Рисунок 1.2. Панель свойств

При необходимости изменения размера отрезка (после окончания ввода его параметров и вычерчивания) возможна полная редакция или удаление отрезка через «Редактирование». Также можно менять и стиль линии. По умолчанию система назначает новому объекту текущий стиль, который был установлен при выполнении предыдущих команд. Внешний вид текущего стиля отображается в поле стиля в строке параметров.

Локальные и глобальные привязки

Для выполнения чертежа по точным размерам в «Компас - 3D» создан специальный механизм, позволяющий точно задать положение курсора, выбрав условие его позиционирования (например, в ближайшей характерной точке, или на пересечении объектов и т.д.). Данный механизм назван привязками. Привязки подразделяются на локальные (действуют разово во время выполнения определенной операции) и глобальные, действие которых постоянно. Для включения нужной локальной привязки можно также использовать контекстное меню (рис. 1.3), появляющееся при нажатии правой кнопки мыши. В точке, соответствующей выбранной

привязке, появится "крестик", свидетельствующий о срабатывании привязки. Если отображение названия привязки включено, то рядом с "крестиком" появится наименование действующей привязки. Нажмите клавишу <Enter> или левую кнопку мыши. Точка, отмеченная "крестиком", будет зафиксирована. Локальные привязки применяются для одноразовых действий.

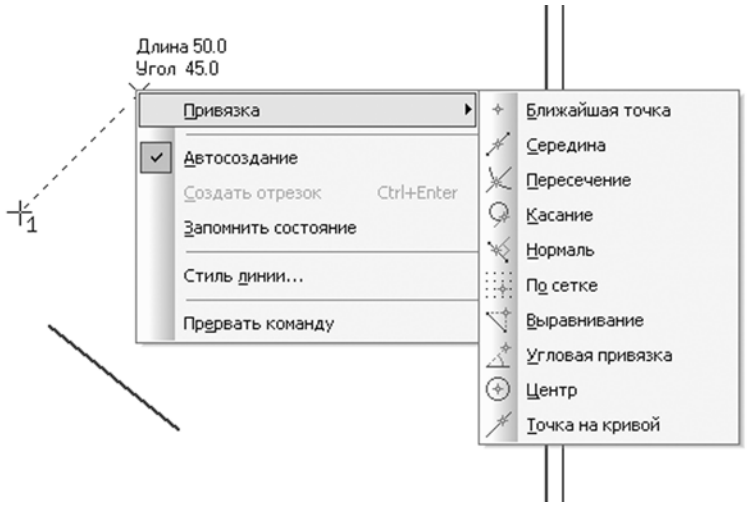


Рисунок 1.3. Локальные привязки

В случае однотипных действий с привязками удобнее применять глобальные привязки. Как правило, включено несколько различных глобальных привязок к объектам, и все они будут работать одновременно (рис. 1.4).

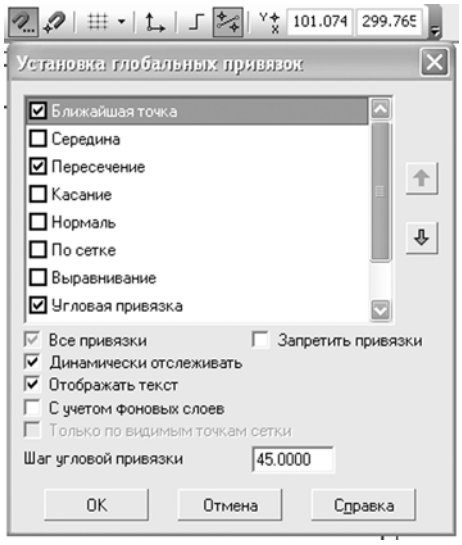


Рисунок 1.4. Глобальные привязки

При этом расчет точки выполняется «на лету», на экране отображается фантом, соответствующий этой точке, и текст с именем действующей в данный момент привязки. Цвет отображения фантома и текста соответствует цвету, установленному для увеличенного курсора. Для

вызова этого диалога служит кнопка «Установка глобальных привязок» на «Панели текущего состояния». Данная кнопка показана на рисунке 1.4. и имеет пиктограмму магнита с тремя точками.

Последовательность создания двухмерного чертежа

Данная последовательность известна из школьного предмета «черчение» при выполнении чертежей ручным способом:

- выбор необходимого количества и расположения видов,
- построение габаритных контуров вспомогательными линиями,
- создание контуров детали основными линиями,
- построение разрезов и сечений,
- простановка размеров

Отличительной особенностью графического редактора является возможность многократного и качественного редактирования чертежа детали.

Глава 2. Основы создания трехмерных моделей в графическом редакторе «Компас - 3D». Ассоциативные виды

Создание трехмерной модели дает новые по сравнению с ручным черчением возможности для проектирования детали. Это не только быстрое и автоматическое редактирование чертежей при изменении параметров трехмерной детали, но в дальнейшем создание сборок и спецификаций.

Основы создания трехмерных моделей в графическом редакторе «Компас - 3D».

Создание новой трехмерной модели начинается с выбора плоскости. Выбор плоскости удобно производить в дереве модели щелчком мыши на имени плоскости (после чего рисунок выбранной плоскость выделяется сплошной линией зеленого цвета). Далее выбирается операция эскиз (либо просто нажимается кнопка с пиктограммой «Эскиз» на «Панели текущего состояния»). После создания эскиза детали в «Компас - 3D» возможны варианты создания трехмерных моделей с помощью следующих операций: выдавливания, вращения, кинематическая, и операции по сечениям. Кроме того, проектирование новой детали может начинаться путем вставки в файл готовой модели заготовки детали.

При этом доступны следующие типы операций:

1. Вращение эскиза вокруг оси, лежащей в плоскости эскиза,

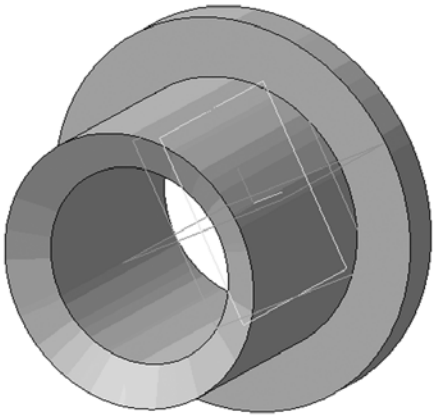


Рисунок 2.1. Элемент, образованный операцией вращения

При создании детали операцией вращения особое внимание следует обратить на следующие положения:

- эскиз контура вращения должен быть выполнен стилем сплошной линии,
- линия контура не должна иметь самопересечений,
- для сплошной детали линия контура вращения должна быть замкнутой (допускается замыкать контур осью вращения),
- для сплошной детали параметр свойства «тип построения тонкой стенки» должен иметь значение «нет»,
- на эскизе должна присутствовать только одна ось вращения,
- ось вращения должна быть выполнена стилем осевой линии

2. Выдавливание эскиза в направлении, перпендикулярном плоскости эскиза,

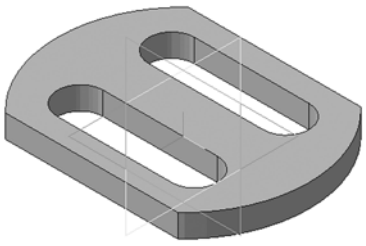


Рисунок 2.2. Элемент, образованный операцией выдавливания

При создании детали операцией выдавливания особое внимание следует обратить на следующие положения:

- эскиз контура основания детали должен быть выполнен стилем сплошной линии,
- линия контура основания детали не должна иметь самопересечений,

- для сплошной детали линия контура основания детали должна быть замкнутой,
- для сплошной детали параметр свойства «тип построения тонкой стенки» должен иметь значение «нет»,
- для сплошной детали допускается внутри контура детали иметь замкнутые контуры вырезов из детали

3. Кинематическая операция – перемещение эскиза вдоль указанной направляющей,

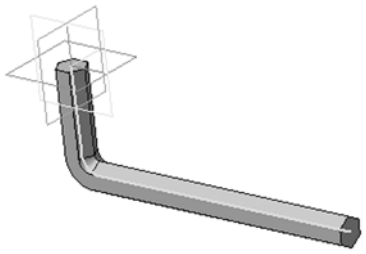


Рисунок 2.3. Элемент, образованный кинематической операцией

При создании детали кинематической операцией особое внимание следует обратить на следующие положения:

- первый эскиз контура сечения детали должен быть выполнен стилем сплошной линии,
- линии контура первого эскиза должны быть замкнуты,
- второй эскиз направляющей линии должен быть выполнен стилем сплошной линии,
- начало траектории второго эскиза направляющей должно лежать в плоскости первого эскиза (например, из начала координат),
- линия контура второго эскиза не должна иметь самопересечений

Вообще говоря, операции выдавливания и вращения являются частными случаями кинематической операции. А именно: при операции выдавливания траектория перемещения эскиза-сечения представляет собой отрезок прямой линии, а при операции вращения – дугу окружности (или полную окружность).

4. Построение тела по нескольким сечениям-эскизам.

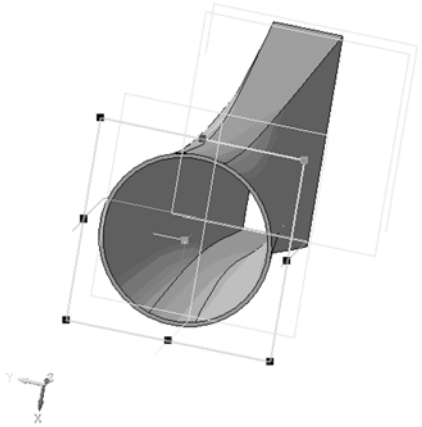


Рисунок 2.4. Элемент, образованный операцией по сечениям

При создании детали операцией по сечениям особое внимание следует обратить на следующие положения:

- каждый эскиз контура сечения детали должен быть выполнен стилем сплошной линии,
- каждый эскиз выполняется в своей плоскости (плоскости можно создать в панели «Вспомогательная геометрия»- «Смещенная плоскость»),
- эскизы контуров сечения добавляются в список сечений в «панели свойств»,
- эскизы сечений должен быть замкнуты,
- эскизы сечений не должна иметь самопересечений

Каждая операция имеет дополнительные опции, позволяющие варьировать правила построения тела.

После создания основания детали производится “приклеивание” или “вырезание” дополнительных объемов. Каждый из них представляет собой элемент, образованный при помощи перечисленных выше операций над новыми эскизами. При выборе типа операции нужно сразу указать, будет создаваемый элемент вычитаться из основного объема или добавляться к нему. Примерами вычитания объема из детали могут быть различные отверстия, проточки, канавки, а примерами добавления объема – бобышки, выступы, ребра.

Ассоциативные виды

Механизм ассоциативных видов позволяет преобразовать трехмерную модель в проекционные виды чертежа. Рассмотрим вопрос, как из готовой трехмерной модели создать чертеж детали? В полной версии «Компас -

3D» в меню трехмерного моделирования необходимо найти и нажать кнопку – «Новый чертеж из модели»

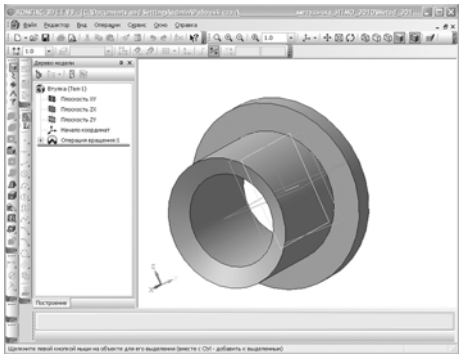


Рисунок 2.5. Новый чертеж из модели

В версии «Компас -3D» LT, применяемых для учебных целей такой кнопки нет, поэтому необходимо закрыть трехмерную модель, перейти в создание нового чертежа. В открытом новом чертеже найти и нажать кнопку «Ассоциативные виды» и далее «Стандартные виды». Опытный пользователь может также использовать кнопку «Проекционный вид».

Вставить модель из файла трехмерной модели. Перед сохранением объекта обратим внимание на свойства «Схема видов» и «Ориентация главного вида»

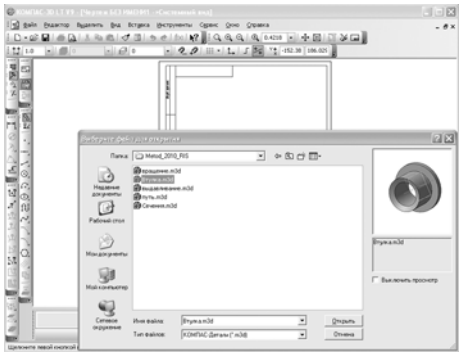


Рисунок 2.6. Выбор модели и ориентация главного вида.

В случае правильного выбора ориентации главного вида при построении трехмерной модели создание ассоциативных видов сводится к выбору количества видов на поле чертежа в «Схеме видов». **Следует обратить особое внимание, что только после правильного выбора главного вида возможна дальнейшая работа с видами. В случае неправильного выбора главного вида при построении трехмерной модели создание ассоциативных видов можно скорректировать, вернувшись в трехмерную модель и добавив в главном меню (вид –> ориентация –> кнопка добавить) имя нового главного вида. После чего в схеме ассоциативных видов поменять вид спереди на новый главный вид.** Далее остается только перенести необходимые виды из схемы видов на чертеж.

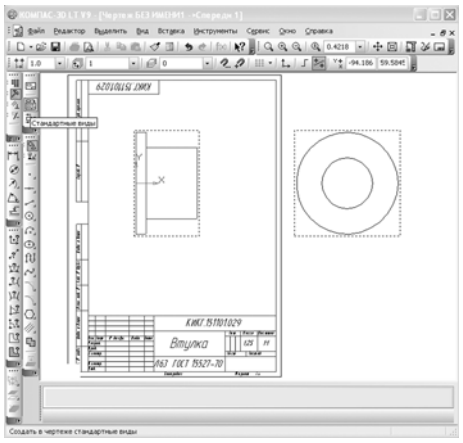


Рисунок 2.7. Выбор видов

В целях детализации внутренних изображений (разрезы, сечения) возможно выделить и разрушить вид. Для чего нажать на габаритном прямоугольнике вида, который подсветиться зеленым цветом. Затем выбрать в главном меню: Редактор –> Разрушить или в контекстном меню строку «Резрушить»

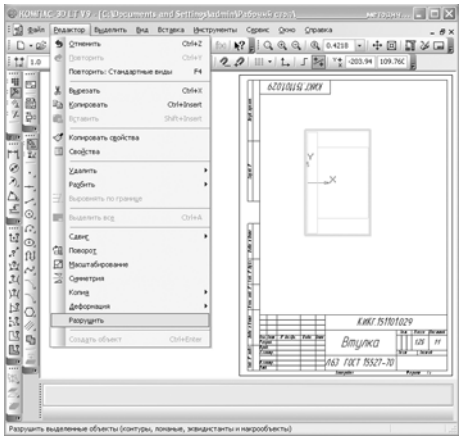


Рисунок 2.8. Разрушение вида

После разрушения единого изображения на набор геометрических элементов можно дорисовать вручную половину разреза детали

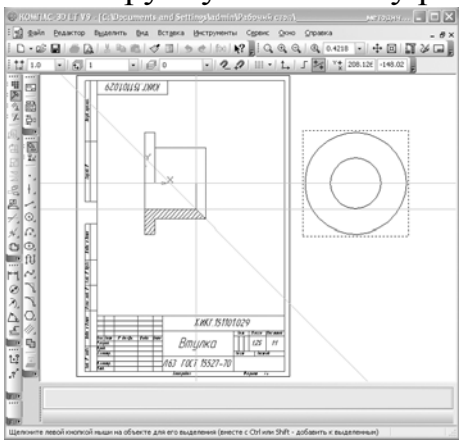


Рисунок 2.9. Ручная дорисовка разреза

Создание простых разрезов

Чтобы не прибегать к разрушению видов можно использовать встроенные в «Компас - 3D» возможности. К ним относятся разрезы (простые, сложные, местные), дополнительные и вспомогательные виды, вынесенные элементы. Приведем последовательность создания простого и сложного разреза.

Нарисовать вспомогательную линию с привязкой по середине. Обозначить разрез: Обозначения –> Линия разреза –> Клик мышкой на точках плоскости сечения (появится обозначение разреза А-А) и появляется фантом проекционного вида разреза. **Особое внимание необходимо уделить статусу разрезаемого вида. Этот вид должен быть текущим (изображен синим цветом). В противном случае вместо изображения разреза детали получается пустое место.**

Далее остается поставить разрез на место главного вида. Сам главный вид лучше не удалять, а погасить в менеджере документов, нажав на пиктограмму с изображением лампочки.

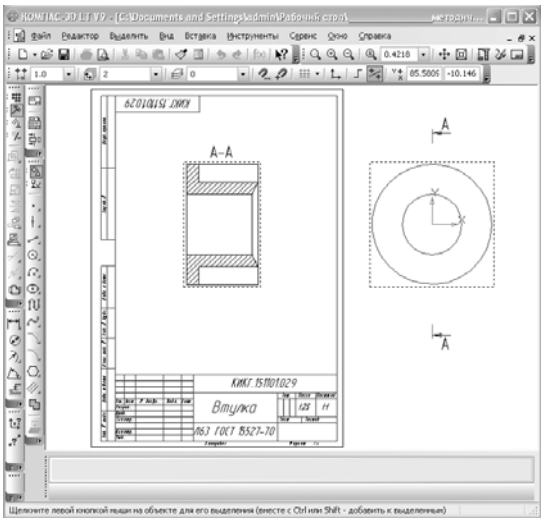


Рисунок 2.10. Простой разрез

Создание сложных разрезов

Однако созданный ранее простой разрез не дает возможности совместить для симметричной детали пол вида с половиной разреза, поэтому в данном случае лучше применить сложный разрез. Действия аналогичные, но обозначение разреза примет ступенчатую форму. После обозначения первой точки разреза необходимо нажать в панели свойств кнопку с пиктограммой «Сложный разрез», а после обозначения последней точки необходимо отжать эту кнопку.

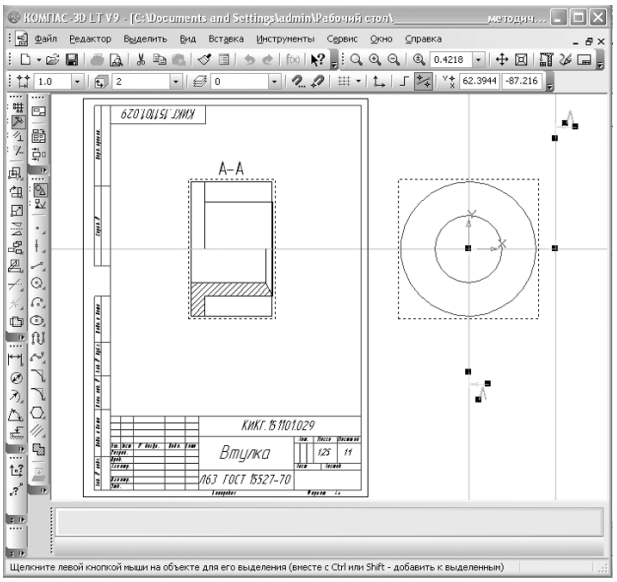


Рисунок 2.11. Сложный разрез

Простановка размеров

- Для симметричных деталей размеры ставятся через центр (при этом ставится не два, а один размер, то есть на один размер меньше)
- Для несимметричных деталей размеры ставятся от базы или нескольких баз. В качестве базы принято выбирать край детали. В случеа нескольких баз размеры ставятся с учетом технологии изготовления детали
- При совмещении вида с разрезом наружные размеры ставятся на виде, а внутренние размеры на разрезе.
- Габаритные размеры ставятся всегда
- Размеры до центров отверстий (пересечения центровых линий) ставятся всегда
- В отверстиях в качестве размера ставятся всегда диаметры (радиусы в отверстиях не ставятся!)
- Радиусы скруглений ставятся при условии, что радиус не превышает половины окружности, в противном случае ставится диаметр.
- Технологические размеры ставятся в зависимости от технологической последовательности изготовления детали.
- Для тел вращения размеры ставятся от баз – краев детали.
- Внутренние диаметры для симметричной детали с половиной разреза ставятся с одной стороны (со стороны разреза) от центровой линии

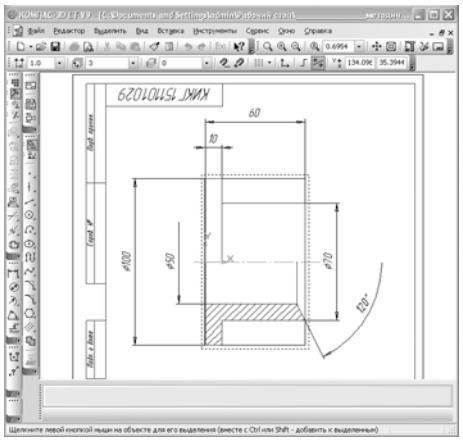


Рисунок 2.12. Простановка размеров

Глава 3. Функции базовой графики на языке программирования C# и работа в среде MS Visual Studio 2005 Express

Для того чтобы студент мог прочесть в литературе листинг (текст программы), содержащий графические функций на языках C++, Java, C# проведем небольшое сравнение на примере функции рисования прямоугольника (Rectangle). Функция Rectangle рисует прямоугольник, используя выбранное перо, и может закрасить его внутренность с помощью выбранной кисти.

Строка кода на языке C++ в среде C++ Builder для рисования прямоугольника (*Rectangle*) на поверхности «холста» (*Canvas*) формы *(*Form1*)* приложения имеет вид:

```
Form1->Canvas->Rectangle(X1, Y1, X2, Y2);
```

Здесь *X1*, *Y1* и *X2*, *Y2* — координаты верхнего левого и правого нижнего угла прямоугольника.

Строка кода на языке C++ в среде Visual C++ для рисования прямоугольника на форме приложения с функцией Win32 API имеет вид:

```
Rectangle(hdc, X1, Y1, X2, Y2);
```

Здесь hdc — идентификатор контекста устройства *HDC* (или дескриптор окна).

Другим вариантом Visual C++ в MS Visual Studio является создание приложения Windows Forms Application. В этом случае фрагмент кода C++ для рисования прямоугольника на форме приложения может выглядеть так:

```
e->Graphics->DrawRectangle(pen, X1, Y1, W, H);
```

Здесь pen – имя объекта класса Pen с параметрами линии, W – ширина и H – высота рисуемого прямоугольника.

На языке Java графические функции используют не только для приложений. Их широко используют для апплетов и мидлетов.

Рассмотрим фрагмент кода на Java для рисования прямоугольника:

```
g.drawRect(X1, Y1, W, H);
```

Здесь g — любое имя для указателя на класс Graphics,

Рассмотрим фрагмент кода на C# для рисования прямоугольника:

```
g.DrawRectangle(pen, X1, Y1, W, H);
```

Здесь g — любое имя для указателя на класс Graphics.

Обобщая данные функции, заметим, что в каждой из них указывается место рисования и задаются размеры прямоугольника, параметры линии, как правило, задаются по умолчанию. Как видно из примера с функцией Rectangle мы имеем большую схожесть графических классов для разных языков программирования. Поэтому нетрудно прочитать и перевести код с графическими функциями с одного языка на другой. Особенно большое структурное сходство классов графических функций между языками Java и C#. В современной литературе графические функции более часто называются методами, что не меняет их назначения.

В современную школьную программу по информатике («Информатика и ИКТ») уже входит раздел создания приложений в объектно-ориентированной среде программирования. Студент, чувствующий себя уверенно в этой области и знающий основы программировании на языках C# или Java, может не читать дальше эту главу.

Чтобы любой студент мог выполнить самостоятельные работы в среде Visual Studio C# Express, подробно рассмотрим последовательность создания приложения WindowsFormsApplication (WindowsApplication) с графическими функциями. Существуют незначительные различия в интерфейсе сред Visual Studio 2005 и 2008 года. В данных методических указаниях приводятся, как правило, названия из обеих версий. Кроме того, версия 2008 года русифицирована, поэтому часть названий приведено в двух вариантах: русском и англоязычном.

3.1 Лабораторная работа 1: «Первая программа. Кнопка button1».

В настоящее время во всех современных средах программирования применяются однотипные наборы визуальных компонентов. Рассмотрим последовательность создания приложения с кнопкой - «button». После

запуска среды программирования Visual Studio C# Express необходимо выполнить следующие шаги:

1. Создать новый проект - New Project с именем WindowsFormsApplication (WindowsApplication1), или любым другим по вашему усмотрению, например, g151701.

2. Выбрать шаблон: Приложение Windows Forms (Windows Application)

3. Выбрать вкладку формы приложения: Form1.cs (Конструктор - Design)

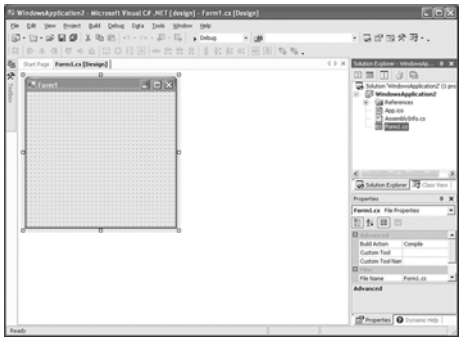


Рисунок 3.1. Шаг 3

4. Открыть визуальные компоненты в окне: Панель элементов - ToolBox

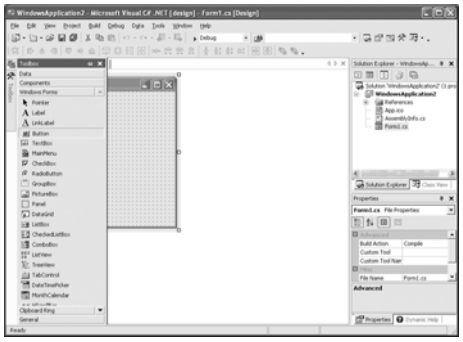


Рисунок 3.2. Шаг 4

5. Перенести из этого окна на форму визуальный компонент Button

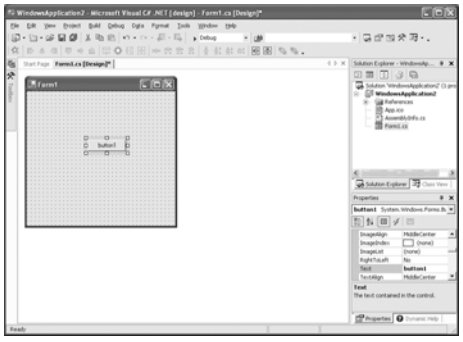


Рисунок 3.3. Шаг 5

6. Сделать двойной щелчок на визуальной кнопке button1, расположенной на форме, тогда по умолчанию откроется окно: вкладка редактора кода с

именем Form1.cs. В данном окне автоматически (по умолчанию) создается метод button1_Click в файле Form1.cs

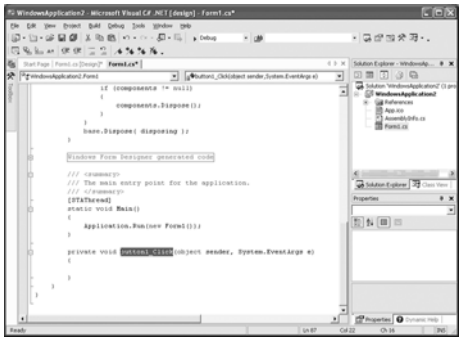


Рисунок 3.4. Шаг 6

7. Вписать внутри фигурных скобок метода button1_Click любую функцию. Например, this.Close( ) - функцию закрытия формы. В случае затруднения в правильности написании имени функции всегда можно получить подсказку с готовым именем возможной функции. Для этого, написав первое слово (в данном случае this – указатель на класс) и поставив после точку, надо подождать секунду до появления всплывающего списка возможных правильных продолжений кода.

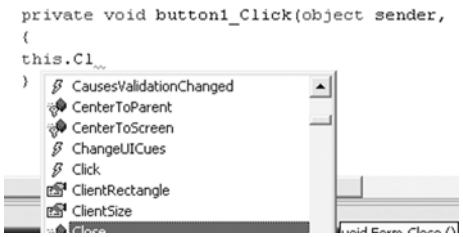


Рисунок 3.5. Шаг 7

Тогда текст метода button1_Click должен выглядеть так:

private void button1_Click(object sender, System.EventArgs e) {

this.Close( ); //Это и есть вписанная нами единственная строчка

}

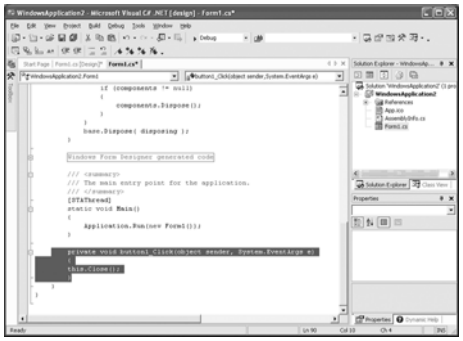


Рисунок 3.6. Шаг 7

8. Запустить программу на выполнение через отладку Debug

Если в программе нет ошибок, то после компиляции кода запустится приложение. После нажатия на кнопку с названием button1 начнет работать метод this.Close() закрытия формы.

Рисунок 3.7. Шаг 8

3.2 Способы вывода графики на форму приложения

Для вывода графических методов можно применять три метода инициализации графики, что связано с получением указателя на область вывода графических функций. В самом простом случае в качестве области вывода выберем форму приложения (по умолчанию Form1)

3.2.1 Инициализация графики методом Paint

Включим в свою программу событие формы Paint:

1. После создания приложения WindowsApplication откроем окно свойств формы (Свойства - Properties).
2. В данном окне перейдем на вкладку методов, нажав на пиктограмме «молния» с всплывающей подсказкой (события - Events).
3. Из таблицы методов выберем метод Paint.

Рисунок 3.8 Окно свойств формы Form1

4. Выполним двойной щелчок в пустом поле таблицы справа от (выделенной на рисунке) ячейки со словом «Paint». При правильной работе автоматически должен сгенерироваться метод Form1_Paint и в файл Form1.cs вписывается фрагмент кода.

```
private void Form1_Paint(object sender, PaintEventArgs e)
  {
  }
```

5. Впишем в метод Form1_Paint программный код:

```
private void Form1_Paint(object sender, PaintEventArgs e)
  {
Graphics g = e.Graphics; // «Бесплатная» инициализация графики
      Pen bluePen = new Pen(Color.Blue, 2);
      Pen blackPen = new Pen(Color.Black, 10);
      g.DrawLine(blackPen, 10, 20, 110, 30);
      g.DrawRectangle(bluePen, 10, 40, 100, 50);
  }
```

6. Получим работающее приложение:

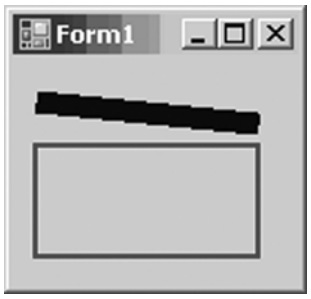


Рисунок 3.9 Работа приложения с рисованием на Form1

Достоинством данной инициализации является «бесплатная» инициализация графики. Недостатком такого вида инициализации графики является отсутствие управляющей последовательности вывода графики на форму.

Обратим особое внимание на то, что на данном примере выбора события Paint для формы приложения Form1, было показано, как вставлять в код программы стандартные события. По такой последовательности выбора события из таблицы событий возможно включать в свою программу любые доступные события для данного компонента.

3.2.2 Инициализация графики методом Create

Теперь впишем инициализацию графики в управляющую кнопку через событие: «нажать кнопку» - `button1_Click`. В этом случае удается избежать предыдущего недостатка, то есть графика появляется на форме Form1 только после нажатия кнопки button1. Последовательность действий аналогична пункту 3.1.

```
private void button1_Click(object sender, EventArgs e)
  {
```

```
        Graphics g = CreateGraphics();// инициализация графики
        Pen bluePen = new Pen(Color.Blue, 2);
        Pen blackPen = new Pen(Color.Black, 10);
        g.DrawLine(blackPen, 10, 10, 110, 20);
        g.DrawRectangle(bluePen, 20, 30, 100, 50);
    }
```

3.2.3 Инициализация графики методом FromHwnd

Теперь для инициализации графики применим метод **`FromHwnd`**, который позволяет указывать на место вывода графики.

```
  private void button1_Click(object sender, EventArgs e)
    {
        Graphics g = Graphics.FromHwnd(this.Handle);
// инициализация графики
        Pen bluePen = new Pen(Color.Blue, 2);
        Pen blackPen = new Pen(Color.Black, 10);
        g.DrawLine(blackPen, 10, 10, 110, 20);
        g.DrawRectangle(bluePen, 20, 30, 100, 50);
    }
```

В данном конкретном случае местом вывода является указатель (**`Handle`**) на созанный нами объект (**`this`**) приложения. В общем случае с использованием метода **`FromHwnd()`** мы можем применять указатель не только на форму приложения, но и на любой визуальный объект, поддерживающий класс **`Graphics.`**

3.3 Лабораторная работа 2: «Функции графики System.Drawing»

Графика в C# основана на графических интерфейсах (Graphics Device Interface) - подсистеме Windows для вывода графических изображений. Пространства имен в C# для работы с графикой следующие: System.Drawing; System.Drawing.Drawing2D; System.Drawing.Imaging; System.Drawing.Printing; System.Drawing.Text.

Разберем текст кода программы внутри события `button1_Click` с использование базовой графики на языке C#. Для желающих более подробно познакомиться с возможностями данного класса `System.Drawing` на языке C# предлагается выполнить данную работу, используя help online или учебник по C#. К функциям базовой графики System.Drawing относятся функции данной программы. Пример программного кода приводится.

Листинг 3.1. Фрагмент кода графики System.Drawing

```
int x0 = 200,y0=100;float xn = 20,xk=140; float yn=10, yk = 20;
//this.BackColor=Color.FromArgb(10,0,0,10);
SolidBrush b=new SolidBrush(Color.FromArgb(150,0,0,255));
Font f=new Font("Areal",15);
Graphics canvas= Graphics.FromHwnd(this.Handle);
// где canvas - любое имя для указателя
Color c3=Color.Red; Pen pen=new Pen(c3,7.0f);
canvas.DrawRectangle(pen,5,5,280,150);
canvas.DrawString("using System.Drawing", f, b, 25, 70);
Pen pen2 = new Pen(Color.Green, 14.0f);
canvas.DrawLine(pen2,x0-xn,y0+yn,x0-xk,y0+yk);
canvas.DrawEllipse(pen2, 100, 170, 80, 40);
```

Рисунок 3.10 Работа приложения с рисованием базовой графики

3.4 Лабораторная работа 3: «Функции графики System.Drawing.Drawing2D»

В данном программном коде разберем расширенные возможности класса на конкретных примерах `System.Drawing.Drawing2D`. Для желающих более подробно познакомиться с возможностями данного класса в C# предлагается выполнить данную работу, используя help online или учебник по C#.

Обратим особое внимание на то, что класс System.Drawing автоматически включается в программу при создании приложения, а для класса System.Drawing.Drawing2D программисту необходимо обязательно вписать объявление этого класса вручную.

Рассмотрим пример в C# с графикой System.Drawing и графикой System.Drawing.Drawing2D

Листинг 3.2. Фрагмент кода графики System.Drawing.Drawing2D

```
Graphics canvas=Graphics.FromHwnd(this.Handle);
Pen  pen=new Pen(Color.Green,2);
canvas.DrawRectangle(pen,10,20,50,50);
canvas.DrawLine(pen,100,20,200,40);

Point p1=new Point(100,40);Point p2=new Point(200,60);
// вписать вручную до начала использования функций класса
using System.Drawing.Drawing2D;
pen.Color=Color.Red;
pen.Width=8;
pen.DashStyle=DashStyle.DashDotDot;
//
Array obj=Enum.GetValues(typeof(LineCap));
LineCap t1=(LineCap)obj.GetValue(6);   pen.StartCap=t1;
LineCap t2=(LineCap)obj.GetValue(8);   pen.EndCap=t2;
//
canvas.DrawLine(pen,p1,p2);
//
SolidBrush b=new SolidBrush(Color.Blue);
canvas.FillPie(b,10,150,50,60,0,270);
//
Array obj2=Enum.GetValues(typeof(HatchStyle));
HatchStyle t3=(HatchStyle)obj2.GetValue(8);
HatchBrush b3=new HatchBrush(t3,Color.Blue,Color.Gold);
canvas.FillEllipse(b3,100,150,50,60);
//
Font font=new Font("Arial",20,FontStyle.Bold|FontStyle.Underline);
float f1=font.GetHeight(); String s1="";s1=s1+f1.ToString();
int f2=(int)f1; String s2="";s2=s2+f2.ToString();
SolidBrush br=new SolidBrush(Color.Gray);
canvas.DrawString("Текст Text",font,br,10,110);
float f3=font.Size; int f4; f4=(int)f3;s2=s2+" "+f3.ToString();
canvas.DrawString(s1,font,br,160,110);
canvas.DrawString(s2,font,br,160,150);
```

Рисунок 3.11 Работа приложения с рисованием базовой графики и графики `Drawing2D`

Глава 4. Основы программирования трехмерной графики в C# с использованием библиотеки OpenGL

Открытая графическая библиотека OpenGL хороша тем, что имеет международный стандарт и разработана для всех платформ и основных языков программирования, в том числе для языков си-язычного направления: C++, Java, C#.

Основы OpenGL

OpenGL является популярным прикладным программным интерфейсом (API – Application Programming Interface) для разработки приложений в области двумерной и трехмерной графики. Стандарт OpenGL (Open Graphics Library – открытая графическая библиотека) был разработан и утвержден в 1992 году как эффективный аппаратно-независимый интерфейс, пригодный для реализации на различных платформах. Основой стандарта стала библиотека IRIS GL, разработанная фирмой Silicon Graphics Inc.

OpenGL состоит из набора библиотек. Все базовые функции хранятся в основной библиотеке, для обозначения которой в дальнейшем мы будем использовать аббревиатуру *GL*. Первая из них – *библиотека утилит GL*. Все функции этой библиотеки вошли графические примитивы, например: точки, линии, полигоны. Все функции имеют префикс gl. Вторая *GLU – GL Utility*. В состав GLU вошла реализация более сложных функций, таких

как набор популярных геометрических примитивов (куб, шар, цилиндр, диск), функции построения сплайнов, реализация дополнительных операций над матрицами и т.п.

OpenGL не включает в себя никаких специальных команд для работы с окнами или ввода информации от пользователя. Поэтому были созданы специальные переносимые библиотеки для обеспечения часто используемых функций взаимодействия с пользователем и для отображения информации с помощью оконной подсистемы. Наиболее популярной является библиотека GLUT (GL Utility Toolkit). Формально GLUT не входит в OpenGL, но de facto включается почти во все его дистрибутивы и имеет реализации для различных платформ. GLUT предоставляет только минимально необходимый набор функций для создания OpenGL-приложения.

Графические команды (функции) OpenGL

Все команды (функции) библиотеки GL начинаются с префикса gl, все константы – с префикса GL_. Соответствующие команды и константы библиотек GLU и GLUT аналогично имеют префиксы glu (GLU_) и glut (GLUT_)

Синтаксис команд

Кроме того, в имена команд входят суффиксы, несущие информацию о числе и типе передаваемых параметров. В OpenGL полное имя команды имеет вид:

type glCommand_name[1 2 3 4][b s i f d ub us ui][v](type1 *arg1*, …, typeN *argN*)

Имя состоит из нескольких частей:

имя библиотеки, в которой описана эта функция: для базовых функций OpenGL, функций из библиотек GL, GLU, GLUT это gl, glu, glut соответственно.

Command_name имя команды (функции)

[1 2 3 4] число аргументов команды

[b s i f d ub us ui] тип аргумента: символ b – GLbyte (аналог char в C\C++), символ s – short, символ i – GLint (аналог int), символ f – GLfloat (аналог float), символ d – double и так далее.

[v] наличие этого символа показывает, что в качестве параметров функции используется указатель на массив значений

Символы в квадратных скобках в некоторых названиях не используются. Например, команда glVertex2i() описана в библиотеке GL, и использует в качестве параметров два целых числа, а команда glColor3fv() использует в качестве параметра указатель на массив из трех вещественных чисел.

Вершины и примитивы

Под вершиной понимается точка в трехмерном пространстве, координаты которой можно задавать следующим образом:

void glVertex[2 3 4][s i f d](type coords)

void glVertex[2 3 4][s i f d]v(type *coords)

Координаты точки задаются максимум четырьмя значениями: x, y, z, w, при этом можно указывать два (x,y) или три (x,y,z) значения, а для остальных переменных в этих случаях используются значения по умолчанию: z=0, w=1. Как уже было сказано выше, число в названии команды соответствует числу явно задаваемых значений, а последующий символ – их типу.

Координатные оси расположены так, что точка (0,0) находится в левом нижнем углу экрана, ось x направлена влево, ось y- вверх, а ось z- из экрана. Это расположение осей мировой системы координат, в которой задаются координаты вершин объекта. Другие системы координат в OpenGL мы здесь рассматривать не будем.

Чтобы задать какую-нибудь фигуру одних координат вершин недостаточно, и эти вершины надо объединить в одно целое, определив необходимые свойства. Для этого в OpenGL используется понятие примитивов, к которым относятся точки, линии, связанные или замкнутые линии, треугольники и так далее. Задание примитива происходит внутри командных скобок:

void **glBegin**(GLenum mode)

void **glEnd**(void)

Параметр mode определяет тип примитива, который задается внутри и может принимать следующие значения:

GL_POINTS каждая вершина задает координаты некоторой точки.

GL_LINES каждая отдельная пара вершин определяет отрезок; если задано нечетное число вершин, то последняя вершина игнорируется.

GL_LINE_STRIP каждая следующая вершина задает отрезок вместе с предыдущей.

GL_LINE_LOOP отличие от предыдущего примитива только в том, что последний отрезок определяется последней и первой вершиной, образуя замкнутую ломаную.

GL_TRIANGLES каждая отдельная тройка вершин определяет треугольник; если задано не кратное трем число вершин, то последние вершины игнорируются.

GL_TRIANGLE_STRIP каждая следующая вершина задает треугольник вместе с двумя предыдущими.

GL_TRIANGLE_FAN треугольники задаются первой и каждой следующей парой вершин (пары не пересекаются).

GL_QUADS каждая отдельная четверка вершин определяет четырехугольник; если задано не кратное четырем число вершин, то последние вершины игнорируются.

GL_QUAD_STRIP четырехугольник с номером n определяется вершинами с номерами 2n-1, 2n, 2n+2, 2n+1.

GL_POLYGON последовательно задаются вершины выпуклого многоугольника.

Для задания текущего цвета вершины используются команды

void glColor[3 4][b s i f](GLtype components)

void glColor[3 4][b s i f]v(GLtype components)

Первые три параметра задают R, G, B компоненты цвета, а последний параметр определяет alpha-компоненту, которая задает уровень прозрачности объекта. Если в названии команды указан тип ‘f’ (float), то значения всех параметров должны принадлежать отрезку [0,1], при этом по умолчанию значение alpha-компоненты устанавливается равным 1.0, что соответствует полной непрозрачности. Если указан тип ‘ub’ (unsigned byte), то значения должны лежать в отрезке [0,255].

Разным вершинам можно назначать различные цвета и тогда будет проводиться линейная интерполяция цветов по поверхности примитива.

Для управления режимом интерполяции цветов используется команда void **glShadeModel**(GLenummode) вызов которой с параметром GL_SMOOTH включает интерполяцию (установка по умолчанию), а с GL_FLAT отключает.

Например, чтобы нарисовать треугольник с разными цветами в вершинах, можно написать так:

```
GLfloat BlueCol[3]={0,0,1};
glBegin(GL_TRIANGLE);
glColor3f(1.0, 0.0, 0.0); //красный
glVertex3f(0.0, 0.0, 0.0);
glColor3ub(0,255,0); //зеленый
glVertex3f(1.0, 0.0, 0.0);
glColor3fv(BlueCol); //синий
glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

Для задания цвета фона используется команда **void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)**. Значения должны находиться в отрезке [0,1] и по умолчанию равны нулю. После этого вызов команды **void glClear(GLbitfield mask)** с параметром **GL_COLOR_BUFFER_BIT** устанавливает цвет фона во все буфера, доступные для записи цвета.

Скелет программы OpenGL.

Существует два варианта использования библиотеки OpenGL для языка программирования C#. Первый вариант – это импорт системных библиотек Microsoft Windows. Второй вариант – это применение библиотек OpenGL, разработанных под язык программирования C#.

Применение библиотек OpenGL для языка C#

Библиотечные файлы OpenGL для C# можно скачать с сайта **http://csgl.sourceforge.net**. Нашему заданию соответствует пятый урок по созданию трехмерных объектов с использованием библиотеки OpenGL с сайта **http://nehe.gamedev.net**. Разберем данный урок с кодом исходного файла. Обратите особое внимание на строчку кода **using CsGL.Basecode;** и метод **private override void Draw()** с командами (функциями) OpenGL. Преимуществом данного варианта является простота инициализации базового кода при работе с командами OpenGL. Особое внимание следует обратить на наличие рядом с исполняемым файлом динамических библиотек CsGL.Basecode.dll, csgl.dll, csgl.native.dll. Кроме того, мы должны в явном виде подключить в тексте программы ссылку, записав строку **using CsGL.Basecode;**

Листинг 4.1. Текст программы применения библиотеки OpenGL.

```
//////////////
/* Lesson 05 of NeHe Productions in C# created by Sabine Felsinger*/
using System;
using System.Windows.Forms;
using System.Drawing;
using CsGL.OpenGL;

namespace lesson05
{
public class OurView : OpenGLControl
{
public float rtri; // rtri is for rotating the pyramid
public float rquad; // rquad is for rotating the quad
public bool finished;
```

```
            public OurView() : base()
            {
this.KeyDown += new KeyEventHandler(OurView_OnKeyDown);
                  finished = false;
            }

protected void OurView_OnKeyDown(object Sender, KeyEventArgs kea)
            {
//if escape was pressed exit the application
                  if (kea.KeyCode == Keys.Escape)
                  {
                        finished = true;
                  }
            }

            public override void glDraw()
            {
GL.glClear(GL.GL_COLOR_BUFFER_BIT |
GL.GL_DEPTH_BUFFER_BIT);
// Clear the Screen and the Depth Buffer
GL.glMatrixMode(GL.GL_MODELVIEW);    // Modelview Matrix
GL.glLoadIdentity();
// reset the current modelview matrix
GL.glTranslatef(-1.5f,0.0f,-6.0f);
      // move 1.5 Units left and 6 Units into the screen
GL.glRotatef(rtri,0.0f,1.0f,0.0f);
      // rotate the Pyramid on it's Y-axis
rtri+=0.2f;                        // rotation angle

GL.glBegin(GL.GL_TRIANGLES);
//start drawing a triangle, always counterclockside (top-left-right)
GL.glColor3f(1.0f,0.0f,0.0f);   // Red
GL.glVertex3f(0.0f,1.0f,0.0f);// Top of Triangle (Front)
GL.glColor3f(0.0f,1.0f,0.0f);   // green
GL.glVertex3f(-1.0f,-1.0f,1.0f);// left of Triangle (front)
GL.glColor3f(0.0f,0.0f,1.0f);   // blue
GL.glVertex3f(1.0f,-1.0f,1.0f);// right of triangle (front)

GL.glColor3f(1.0f,0.0f,0.0f);   // red
GL.glVertex3f(0.0f,1.0f,0.0f);// top of triangle (right)
```

```
GL.glColor3f(0.0f,0.0f,1.0f);   // blue
GL.glVertex3f(1.0f,-1.0f,1.0f);// left of triangle (right)
GL.glColor3f(0.0f,1.0f,0.0f);   // green
GL.glVertex3f(1.0f,-1.0f,-1.0f);// right of triangel (right)

GL.glColor3f(1.0f,0.0f,0.0f);   // red
GL.glVertex3f(0.0f,1.0f,0.0f);// top of triangle (back)
GL.glColor3f(0.0f,1.0f,0.0f);   // green
GL.glVertex3f(1.0f,-1.0f,-1.0f);// left of triangle (back)
GL.glColor3f(0.0f,0.0f,1.0f);   // blue
GL.glVertex3f(-1.0f,-1.0f,-1.0f);// right of triangle (back)

GL.glColor3f(1.0f,0.0f,0.0f);   // red
GL.glVertex3f(0.0f,1.0f,0.0f);// top of triangle (left)
GL.glColor3f(0.0f,0.0f,1.0f);          // blue
GL.glVertex3f(-1.0f,-1.0f,-1.0f);      // left of triangle (left)
GL.glColor3f(0.0f,1.0f,0.0f);          // green
GL.glVertex3f(-1.0f,-1.0f,1.0f);       // right of triangle (left)
GL.glEnd();

GL.glLoadIdentity();    // reset the current modelview matrix
   GL.glTranslatef(1.5f,0.0f,-7.0f);
// move 1.5 Units right and 7 into the screen
       GL.glRotatef(rquad,1.0f,1.0f,1.0f);
       // rotate the quad on the X,Y and Z-axis
       rquad-=0.15f;              // rotation angle

GL.glBegin(GL.GL_QUADS);       // start drawing a quad
GL.glColor3f(0.0f,1.0f,0.0f);          // green top
GL.glVertex3f(1.0f,1.0f,-1.0f);        // top right (top)
GL.glVertex3f(-1.0f,1.0f,-1.0f);       // top left (top)
GL.glVertex3f(-1.0f,1.0f,1.0f);        // bottom left (top)
GL.glVertex3f(1.0f,1.0f,1.0f);         // bottom right (top)

GL.glColor3f(1.0f,0.5f,0.0f);          // orange
GL.glVertex3f(1.0f,-1.0f,1.0f);        // top right (bottom)
GL.glVertex3f(-1.0f,-1.0f,1.0f);       // top left (bottom)
GL.glVertex3f(-1.0f,-1.0f,-1.0f);      // bottom left (bottom)
GL.glVertex3f(1.0f,-1.0f,-1.0f);       // bottom right (bottom)
```

```
GL.glColor3f(1.0f,0.0f,0.0f);          // red
GL.glVertex3f(1.0f,1.0f,1.0f);         // top right (front)
GL.glVertex3f(-1.0f,1.0f,1.0f);        // top left (front)
GL.glVertex3f(-1.0f,-1.0f,1.0f);       // bottom left (front)
GL.glVertex3f(1.0f,-1.0f,1.0f);        // bottom right (front)

GL.glColor3f(1.0f,1.0f,0.0f);          // yellow
GL.glVertex3f(-1.0f,1.0f,-1.0f);       // top right (back)
GL.glVertex3f(1.0f,1.0f,-1.0f);        // top left (back)
GL.glVertex3f(1.0f,-1.0f,-1.0f);       // bottom left (back)
GL.glVertex3f(-1.0f,-1.0f,-1.0f);      // bottom right (back)

GL.glColor3f(0.0f,0.0f,1.0f);          // blue
GL.glVertex3f(-1.0f,1.0f,1.0f);        // top right (left)
GL.glVertex3f(-1.0f,1.0f,-1.0f);       // top left (left)
GL.glVertex3f(-1.0f,-1.0f,-1.0f);      // bottom left (left)
GL.glVertex3f(-1.0f,-1.0f,1.0f);       // bottom right (left)

GL.glColor3f(1.0f,0.0f,1.0f);          // violett
GL.glVertex3f(1.0f,1.0f,-1.0f);        // top right (right)
GL.glVertex3f(1.0f,1.0f,1.0f);         // top left (right)
GL.glVertex3f(1.0f,-1.0f, 1.0f);       // bottom left (right)
GL.glVertex3f(1.0f,-1.0f,-1.0f);       // bottom right (right)
GL.glEnd();

            }

            protected override void InitGLContext()
            {
GL.glShadeModel(GL.GL_SMOOTH);// enable smooth shading
GL.glClearColor(0.90f, 0.90f, 0.90f, 0.5f);// background
GL.glClearDepth(1.0f);         // depth buffer setup
GL.glEnable(GL.GL_DEPTH_TEST);// enables depth testing
GL.glDepthFunc(GL.GL_LEQUAL);      // type of depth test
GL.glHint(GL.GL_PERSPECTIVE_CORRECTION_HINT,
GL.GL_NICEST);
// nice perspective calculations
//rtri = 30.0f;              define the rotation angle in the start position of the
triangle
//rquad = 30.0f;    define the rotation angle in the start position of the quad
```

```
            }

            protected override void OnSizeChanged(EventArgs e)
            {
                  base.OnSizeChanged(e);
                  Size s = Size;

                  GL.glMatrixMode(GL.GL_PROJECTION);
                  GL.glLoadIdentity();
GL.gluPerspective(45.0f, (double)s.Width /(double) s.Height, 0.1f, 100.0f);

                  GL.glMatrixMode(GL.GL_MODELVIEW);
                  GL.glLoadIdentity();
            }
      }
      public class MainForm : System.Windows.Forms.Form
      {
            private lesson05.OurView view;
            public MainForm()
            {
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(640, 480);
this.Name = "MainForm";
this.Text = "NeHe lesson 05 in C# (by Sabine Felsinger)";
this.view = new lesson05.OurView();
this.view.Parent = this;
this.view.Dock = DockStyle.Fill; // Will fill whole form
this.Show();
            }
            static void Main()
            {
                  MainForm form = new MainForm();

while ((!form.view.finished) && (!form.IsDisposed))
// refreshing the window, so it rotates
                  {
                        form.view.glDraw();
                        form.Refresh();
                        Application.DoEvents();
                  }
```

```
                form.Dispose();
            }
        }
}
```

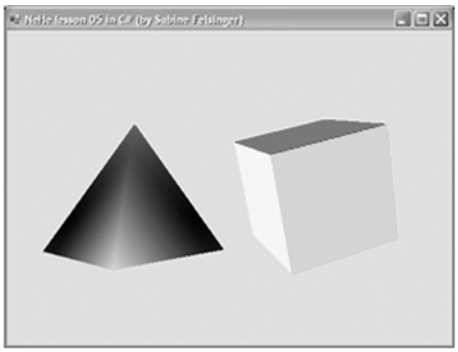


Рисунок 4.1. Пример работы программы

Глава 5. Интернет технологии для трехмерного моделирования на языке VRML и графика на языке Java

Быстрое развитие интернет-технологий не обошло вниманием и компьютерную графику. Наиболее известной и простой технологией программирования и создания трехмерных моделей для интернета являетя язык VRML. В настоящее время в интернет-технологиях стал применяться межплатформенный язык Java, в котором уделено место разделу компьютерной графики. Термин "виртуальная реальность" вошел в наш лексикон в конце 70-ых годов. Однако и ранее это понятие применялось для тренажеров. Область их применения в те времена была достаточно обширна: подготовка танкистов, артиллеристов, моряков, подводников, космонавтов, авиадиспетчеров, операторов АЭС. Но наиболее передовыми с технической точки зрения были и остаются авиационные тренажеры.

История создания языка VRML

VRML: аббревиатура от Virtual Reality Modeling Language. Стандарт языка описания трехмерных сцен, содержащих объекты, а также возможных взаимодействий между объектами и наблюдателем. Существует три распространенных версии языка VRML: 1.0, 2.0 и 97. В январе 1994 года, Mark Pesce и Tony Parisi придумывают концепцию трехмерного HTML, своеобразного языка описания трехмерных сцен с поддержкой гиперссылок, и создают пакет программ и экспериментальный формат,

названные ими Labyrinth - первый прообраз VRML. В апреле 1994 года их проект участвует в First International Conference on the World Wide Web (Первой Интернациональной Конференции по WWW), где и звучит впервые термин VRML. За основу языка VRML был взят язык описания трехмерных сцен, разработанный в Silicon Graphics Incorporation из формата файлов программной библиотеки Open Inventor. Gavin Bell, инженер SGI модифицировал формат Open Inventor, создав первый вариант языка VRML, который был анонсирован в октябре 1994 года на "Второй Всемирной Конференции по WWW" и 5 ноября превратился в официальный проект стандарта VRML 1.0.

В августе 1995 года начинается разработка проекта стандарта VRML 2.0. В декабре 1996 стандарт ISO/IEC DIS 14772-1 (VRML 2.0) принят окончательно.

В октябре 1997 года был завершен проект стандарта новой, расширенной версии языка, названной, в соответствии с последними веяниями - VRML 97. Следует отметить, что VRML абсолютный рекордсмен по скорости разработки и принятия официального стандарта, среди всех компьютерных языков. Развитие VRML продолжается, хоть и не столь бурно, и по сей день.

Поскольку большинство браузеров не имеет встроенных средств поддержки VRML, для просмотра VRML-документов необходимо подключить вспомогательную программу - VRML -браузер. Как и в случае с HTML, один и тот же VRML -документ может выглядеть по-разному в разных VRML-браузерах. Кроме того, многие разработчики VRML-браузеров добавляют нестандартные расширения VRML в свой браузер.

Несложные трехмерные модели для VRML-браузеров можно создать при помощи самого простого текстового редактора.

Структура язык VRML

VRML представляет из себя набор узлов (в терминологии VRML - node) дополненный информацией о связях между ними. Узлы похожи на обычные структуры данных, они содержат поля (fields) и события (events). Связи между узлами реализуются с помощью полей, имеющих тип указателя (на узел). Фактически, события также представляют собою указатель на объект способный эти события генерировать, или воспринимать, что сразу разделяет их на два типа - исходящие (EventOut) и входящие (EvevnIn), соответственно. Благодаря подобной концепции, VRML, несомненно, может считаться объектно-ориентированным языком.

Листинг 5.1. Демонстрация фрагмента на языке VRML

#VRML V2.0 utf8

```
Group {
children [
Shape {
appearance Appearance {
material Material { diffuseColor 0.8 0.15 0 }
}
geometry Box {}
}
]
}
```

В VRML приняты следующие единицы измерения: расстояние и размер: метры, углы: радианы, остальные значения: выражаются, как часть от 1. Координаты берутся в трехмерной декартовой системе координат.

Для того чтобы VRML-браузер распознал файл с VRML-кодом, в начале файла ставится специальный заголовок - file header, например

```
#VRML V2.0 utf8
```

Такой заголовок обязательно должен находиться в первой строке файла, кроме того, перед знаком диеза не должно быть пробелов.

Примитивы и узлы язык VRML

В VRML определены четыре базовые примитивные фигуры: куб (верней не куб, а прямоугольный параллепипед), сфера, цилиндр и конус. Эти фигуры называются примитивами (primitives). Набор примитивов невелик, однако комбинируя их, можно строить достаточно сложные трехмерные изображения. Рассмотрим поподробней каждый из примитивов.

Куб. Возможные параметры: width - ширина, height - высота, depth - глубина.

```
Cube {
  width  2   # ширина
  height 3   # высота
  depth  1   # глубина
   }
```

Сфера. Параметр у сферы только один, это radius.

```
Sphere {
  radius 1   # радиус
   }
```

Конус. Возможные параметры: bottomRadius - радиус основания, height - высота, parts - определяет, какие части конуса будут видны. Параметр parts может принимать значения ALL, SIDES или BOTTOM.

```
Cone {
   parts        ALL  #видны и основание, и боковая поверхность конуса
   bottomRadius  1    #радиус основания
   height        2    #высота
  }
```

Цилиндр. Для цилиндра можно задать параметры radius и height. Кроме того, с помощью параметра parts для цилиндра можно определить будут ли отображаться основания цилиндра и его боковая поверхность. Параметр parts может принимать значения ALL, SIDES, BOTTOM или TOP.

```
    Cylinder {
       parts   ALL     #видны все части цилиндра
       radius  1       #радиус основания
       height  2       #высота цилиндра
          }
```

Кроме описанных выше примитивных фигур формы строятся с помощью точек, линий и граней. Использование точек (Points) , линий (Lines) и граней(Faces) позволяет создавать более сложные формы, чем примитивы и генерировать более реальные VRML миры. Формы, созданные с помощью точек, линий и граней имеют больше функциональных возможностей, чем примитивы.

Рассматриваемый принцип построения форм характеризуется тем, что координаты точек и связь между ними задаются отдельно. Данный метод описания форм характеризуется большой гибкостью. Описание геометрии форм происходит в два этапа:

1.Описание координат точек

2.Соеденение точек

Координаты точек описываются с помощью узла Coordinate:

```
Coordinate {
point [ 1.0 2.0 3.0 ,
4.0 1.5 5.3 ,
. . . ]
}
```

Три типа узлов описывают геометрию форм, основанную на точках и связях между ними:

PointSet
IndexedLineSet
IndexedFaceSet

Все они имеют поле coord, значением которого является узел Coordinate . Рассмотрим эти типы узлов более подробно.

Узел PointSet. Этот узел определяет массив точек в трехмерном пространстве в локальной системе координат :

```
PointSet {
coord Coordinate { point [ . . . ] }
color . . .
}
```

Каждой точке может быть присвоен свой цвет. Он задается в поле color. Если поле color не определено, то используется цвет, заданный в поле emissiveColor узла Material. Размер точек постоянный и его нельзя изменять.

Узел IndexedLineSet. Этот узел определяет множество ломанных линий в трехмерном пространстве, соединяющих точки, описанные в поле coord :

```
IndexedLineSet {
coord Coordinate { point [ . . . ] }
coordIndex [ 1 , 0 , 2 , -1 , . . . ]
. . .
}
```

Поле coordIndex описывает соединенения между точками, описанными в поле coord . Числа в поле coordIndex являются индексами точек в поле coord :

```
coordIndex [ 1 , 0 , 3 , -1 , . . . ]
```

При этом справедливы следующие условия:

порядок точек произвольный; **точки индексируются, начиная с нуля;** ломанная линия может содержать несколько точек; конец ломанной определяется числом -1; можно описывать множество ломанных.

Узел IndexedFaceSet. Этот узел определяет форму в трехмерном пространстве, созданную из граней, полученных путем соединения по периметру грани точек, описанных в поле coord.

```
IndexedFaceSet {
coord Coordinate { point [ . . . ] }
coordIndex [ 1 , 0 , 2 , -1 , . . . ]
. . .
}
```

Описание форм с помощью точек и связей между ними позволяет создавать сложные формы. Узлы PointSet, IndexedLineSet и IndexedFaceSet описывают геометрию форм с помощью точек.

Примеры создания трехмерной модели

Листинг 5.2. Демонстрация файла kg0.wrl

```
#VRML V2.0 utf8
# Example kg0.wrl
Group {
   children [
        Shape {
           appearance Appearance {
                material DEF _DefMat Material {
                }
             }
           geometry IndexedFaceSet {
                coord Coordinate {
                     point [
# Array points #############
 0.0  0.0 0.0,
 50.0  10.0  10.0,
 10.0  10.0  10.0,
 10.0  50.0  10.0,
 50.0  50.0  10.0,
 40.0  20.0  20.0,
 20.0  20.0  20.0,
 20.0  40.0  20.0,
 40.0  40.0  20.0,
###################
]
                    }
                solid FALSE
                creaseAngle     0.5
```

```
                coordIndex [
# Array points ########
 1, 2, 3, 4, -1,
 5, 6, 7, 8, -1,
 1, 2, 6, 5, -1,
 2, 6, 7, 3, -1,
 3, 7, 8, 4, -1,
 1, 5, 8, 4, -1,
##########################
]
            }
        }
    ]
}
```

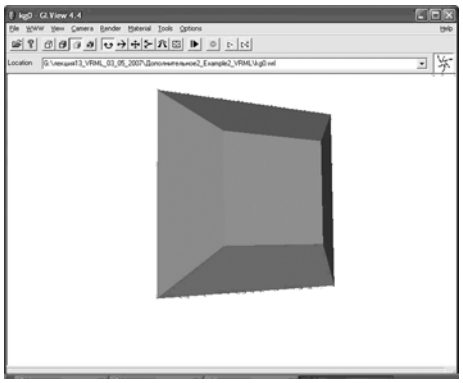


Рисунок 5.1. Пример работы программы

Программирование графики на языке Java

Рассмотрим два понятия необходимые для нашего курса: SDK и applet (апплет).

SDK (Java Developers Kit) — это базовая среда разработки программ на Java. Она является невизуальной и имеет бесплатную лицензию на использование. Есть и визуальные платные среды разработки (например, JBuilder и др.).

Апплет — это небольшая программа, выполняемая браузером (например, на Internet Explorer). Апплет встраивается специальным образом в web-страничку. При подкачке такой странички браузером он выполняется виртуальной Java-машиной самого браузера.

На языке Java возможно создание как самостоятельных приложений, так и создание интернет апплетов. Здесь мы рассмотрим создание апплетов с компьютерной графикой, применяемых в интернет технологиях. В 1995 году компания Sun Microsystems приняла решение объявить о новом продукте, переименовав его в Java (единственное разумное объяснение названию—любовь программистов к кофе). Когда Java оказалась в руках Internet, стало необходимым запускать Java-аплеты—небольшие программы, загружаемые через Internet.

Особенностью языка Java является компиляция исходного кода в так называемый байт-код, независимый от платформы. При компиляции апплета создается файл с расширением class.

Рисунок 5.2. компиляция и запуск Java программы

Исходная Java-программа должна быть в файле с расширением java. Программа транслируется в байт-код компилятором javac.exe. Оттранслированная в байт-код программа имеет расширение class. Для запуска программы нужно вызвать интерпретатор java.exe, указав в параметрах вызова, какую программу ему следует выполнять. Кроме того, ему нужно указать, какие библиотеки нужно использовать при выполнении программы. Библиотеки размещены в файлах с расширением jar

Первая программа и проверка установки Java машины

Для того чтобы работать на языке Java нужно скачать по сети из Sun Microsystems и установить Java Developers Kit—пакет для разработки Java-приложений (http://java.sun.com/products/jdk).

Напишем первую программу для получения обычного исполняемого файла. Данная программа даст нам возможность проверить правильность работы Java машины компьютера и правильности работы компиляторов. Исходный код программы можно создать даже в обычном блокноте, а скомпилировать и выполнить программу через командную строку.

Итак, вот текст первой Java-программы

Листинг 5.3. Первая программа

```
class HelloWorld {
public static void main (String args []) {
System.out.println ("Hello World!");
                    }
            }
```

. Для того чтобы оттранслировать этот пример необходимо запустить транслятор Java—javac.exe, указав в качестве параметра имя файла с исходным текстом. Например:

C: \> javac HelloWorld.java

Транслятор создаст файл HelloWorld.class с независимым от процессора байт-кодом нашего примера. Для запуска программы на выполнение необходимо запустиь файл java.exe, в которую надо загрузить новый класс для исполнения. Подчеркнем, что указывается имя класса, а не имя файла, в котором этот класс содержится.

C: > java HelloWorld

Результатом работы программы станет вывод на консоль строки текста

Hello World!

Если текст появился на экране, установленный Java-транслятор и среда времени выполнения работают.

Первый апплет

Апплеты—это маленькие приложения, которые размещаются на серверах Internet, транспортируются клиенту по сети, автоматически устанавливаются и запускаются на месте, как часть документа HTML. Когда апплет прибывает к клиенту, его доступ к ресурсам ограничен.

Ниже приведен исходный код программы HelloWorld, оформленной в виде апплета:

Листинг 5.4. Первый апплет

```
import java.awt.*;
import java.applet.*;
    public class HelloWorldApplet extends Applet {
        public void paint(Graphics g) {
        g.drawString("Hello World!", 20, 20);
        }
    }
```

Этот апплет начинается двумя строками, которые импортируют все пакеты иерархий java.applet и java.awt. Дальше в нашем примере присутствует метод paint, замещающий одноименный метод класса Applet. При вызове этого метода ему передается аргумент, содержащий ссылку на объект класса Graphics. Последний используется для прорисовки нашего апплета. С помощью метода drawString, вызываемого с этим объектом типа Graphics, в позиции поля апплета (20,20) выводится строка "Hello World".

Для того, чтобы с помощью браузера запустить этот апплет, нам придется написать несколько строк html-текста.

<applet code="HelloWorldApplet" width=200 height=40>

</applet>

Вы можете поместить эти строки в отдельный html-файл (HelloWorldApplet.html), либо вставить их в текст этой программы в виде комментария и запустить программу appletviewer с его исходным текстом в качестве аргумента.

Листинг 5.5. HTML файл для первого апплета

```
<html>
<head><title> Applet  </title></head> <body>
Ниже выполняется апплет.<br>
<applet code = "HelloWorldApple.class"
codebase = ""
alt = "Не могу загрузить апплет"
width = "200" height = "40">
<hr>
Не читаю тег applet
<hr>
</applet>
</body>
</html>
```

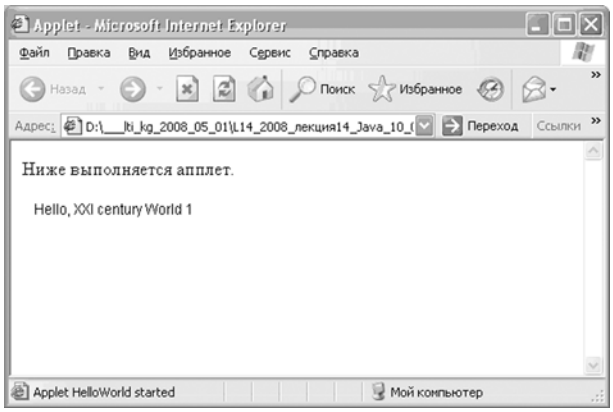


Рисунок 5.3. Пример работы программы

Если апплет с текстом появился на экране, можно себя поздравить еще и с правильной настройкой интернет браузера.

Графика в апплете

Следующей задачей является правильная работа графических функций, при возникновении события, например, функции actionPerformed(ActionEent event) при нажатии кнопки. Обратим особое внимание на инициализацию графических функций с помощью функции getGraphics().

Листинг 5.6. Фрагмент Java кода для рисования в апплете

```
Graphics canvas=getGraphics();
Color bg=getBackground();
    setBackground(bg);
 //  рисование цветом фона
    canvas.setColor(bg);
    canvas.drawRect(160,50,20,10);
//  рисование цветом RGB
    int r2=255,g2=0,b2=255;
canvas.setColor(new Color(r2, g2,b2));
    canvas.drawLine(10,30,150,50);
//  установка параметров шрифта
    canvas.setColor(Color.red);
     Font f=new Font("Areal",1,40);
    canvas.setFont(f);
    canvas.drawString("Текст Text",10,100);
//  рисование цветом RGBA
```

```
    int r1=0,g1=255,b1=255,a1=30;
canvas.setColor(new Color(r1, g1,b1,a1));
canvas.fillRect(10,90,220,40);
//  рисование системным цветом
    canvas.setColor(SystemColor.desktop);
 //  установка параметров пера
// import java.awt.Graphics2D;
Graphics2D g=(Graphics2D)canvas;
BasicStroke pen1=new BasicStroke(10, BasicStroke.CAP_ROUND, BasicStroke.CAP_ROUND);
g.setStroke(pen1);
// import java.awt.geom.*;
 g.draw(new Line2D.Double(60,120,140,160));
```

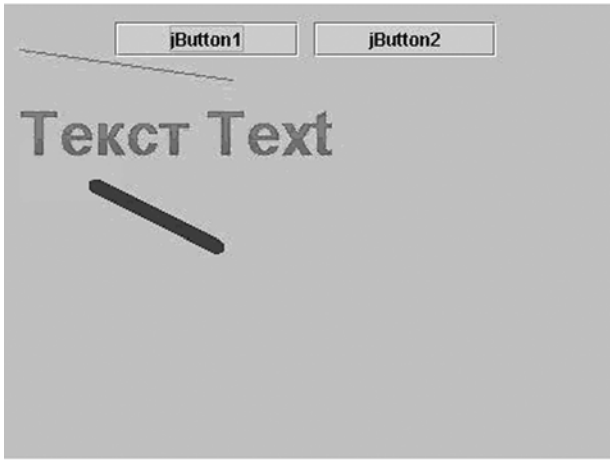


Рисунок 5.4. Пример работы программы

Глава 6. Основы программирования трехмерной графики с использованием DirectX

DirectX — это набор API функций, разработанных для решения задач, связанных с игровым и видеопрограммированием под Microsoft Windows. Наиболее широко используется при написании компьютерных игр. Преимущество использования DirectX заключается в его аппаратурной поддержке совеременными видеокартами для компьютеров. Пакет средств разработки DirectX под Microsoft Windows бесплатно доступен на сайте

Microsoft. Ранее DirectX вкладывался разработчиками в дистрибутивы игр, но сейчас он включён в стандартный набор ПО Windows.
В целом, DirectX подразделяется на:

- **DirectX Graphics**, это набор интерфейсов, ранее (до версии 8.0) делившихся на:
- DirectDraw: интерфейс вывода растровой графики.
- Direct3D (D3D): интерфейс вывода трёхмерных примитивов.
- DirectInput: интерфейс, используемый для обработки данных, поступающих с клавиатуры, мыши, джойстика и пр. игровых контроллеров.
- DirectPlay: интерфейс сетевой коммуникации игр.
- DirectSound: интерфейс низкоуровневой работы со звуком (формата Wave)
- DirectMusic: интерфейс воспроизведения музыки в форматах Microsoft.
- DirectShow: интерфейс, используемый для ввода/вывода аудио и/или видео данных.
- DirectSetup: часть, ответственная за установку DirectX.
- DirectX Media Objects: реализует функциональную поддержку потоковых объектов (например, энкодеры/декодеры)

Более подробную информацию о DirectX можно легко найти в интернете.
Для выполнения самостоятельной работы нас интересует прежде всего вопрос о версии DirectX. В настоящее время с учетом компьютерной базы для выполнения работы рекомендуется использовать версии: DirectX 9.0 или DirectX 9.0c. Информацию о версиях DirectX можно посмотреть в википедии.
Следует также отметить, что DirectX имеет свой формат для трехмерных моделей. Это так называемый «X» формат, имеющий расширение «x».

Последовательность создания трехмерной модели с использованием DirectX:

1. За основу программного кода рекомендуется взять один из готовых примеров документации по DirectX. Возьмем готовый пример Matrices из документации DirectX SDK в папке **Tutorials\Tutorial3**. В данном примере показан вращающийся цветной треугольник на черном фоне. Это своеобразное «Hello, world!» для DirectX.
2. Необходимо поверить работоспособность данного примера на вашем компьютере. Убедиться в правильности подключения ссылок на динамические библиотеки DirectX.
3. При необходимости перенастроить пути к ссылкам (References) в среде Visuel Studio C#

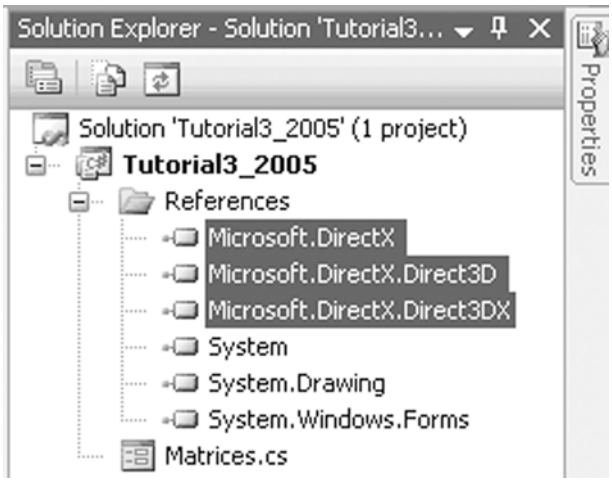


Рисунок 6.1. Пример перенастойки пути к ссылкам
(на рисунке выделенные заливкой три строчки)

4. Далее добавить в готовый пример как минимум четыре дополнения:
   a. включить использование метода **`ZBuffer для определения видимости граней:`**
      device.RenderState.ZBufferEnable = true;
   b. установить флаг перерисовки **`ZBuffer:`** **ClearFlags.ZBuffer**
   c. подготовить массив для ввода произвольного n- количества треугольников (граней трехмерной модели) **const int n = 3;**
   d. изменить число вершин в массиве verts на свое значение равное (число точек n * 3) в функции **`VertexBuffer()`**
   e. После этих изменений исходный файл **Matrices.cs** примет вид, указанный в листинге 6.1.

Листинг 6.1. Программный код с использованием DirectX

```
//-----------------------------------------------------
// File: Matrices.cs
//-----------------------------------------------------
using System;
using System.Drawing;
using System.Windows.Forms;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
using Direct3D=Microsoft.DirectX.Direct3D;
namespace MatricesTutorial
{
```

```
public class Matrices : Form
{       // Our global variables for this project
        Device device = null; // Our rendering device
        VertexBuffer vertexBuffer = null;
        PresentParameters presentParams = new PresentParameters();
        bool pause = false;
    const int n = 3; ////// !!!
        public Matrices()
        {
            // Set the initial size of our form
            this.ClientSize = new System.Drawing.Size(400,300);
            // And it's caption
            this.Text = "Direct3D Tutorial 3 - Matrices";
        }
        public bool InitializeGraphics()
        {       try
            {       // Now let's setup our D3D stuff
                    presentParams.Windowed=true;
                    presentParams.SwapEffect = SwapEffect.Discard;
////////
//  Discard the frames
presentParams.EnableAutoDepthStencil = true;
// Turn on a Depth stencil
presentParams.AutoDepthStencilFormat = DepthFormat.D16;
// And the stencil format
//////////
                    device = new Device(0, DeviceType.Hardware, this,
CreateFlags.SoftwareVertexProcessing, presentParams);
                    device.DeviceReset += new
System.EventHandler(this.OnResetDevice);
                    this.OnCreateDevice(device, null);
                    this.OnResetDevice(device, null);
```

```
                    pause = false;
                    return true;
                }
            catch (DirectXException)   {   return false;     }
            }
            public void OnCreateDevice(object sender, EventArgs e)
            {
            Device dev = (Device)sender;
            // Now Create the VB
/////////////// !!!
vertexBuffer = new VertexBuffer(typeof(CustomVertex.PositionColored),
3*n, dev, 0, CustomVertex.PositionColored.Format, Pool.Default);
////////////////// !!!
        vertexBuffer.Created += new
System.EventHandler(this.OnCreateVertexBuffer);
            this.OnCreateVertexBuffer(vertexBuffer, null);
            }
            public void OnResetDevice(object sender, EventArgs e)
            {
                Device dev = (Device)sender;
            // Turn off culling, so we see the front and back of the triangle
            dev.RenderState.CullMode = Cull.None;
////////////////// !!!
 device.RenderState.ZBufferEnable = true; // Turn on the ZBuffer
////////////////// !!!
// Turn off D3D lighting, since we are providing our own vertex colors
            dev.RenderState.Lighting = false;
            }
            public void OnCreateVertexBuffer(object sender, EventArgs e)
            {
            VertexBuffer vb = (VertexBuffer)sender;
```

```
            CustomVertex.PositionColored[] verts =
(CustomVertex.PositionColored[])vb.Lock(0,0);
        verts[0].X=-1.0f; verts[0].Y=-1.0f; verts[0].Z=0.0f; verts[0].Color =
System.Drawing.Color.Red.ToArgb();
        verts[1].X=1.0f; verts[1].Y=-1.0f ;verts[1].Z=0.0f; verts[1].Color =
System.Drawing.Color.Green.ToArgb();
        verts[2].X=0.0f; verts[2].Y=1.0f; verts[2].Z = 0.0f; verts[2].Color =
System.Drawing.Color.Blue.ToArgb();
         verts[3].X = -1.0f; verts[3].Y = -1.0f; verts[3].Z = 0.5f; verts[3].Color
= System.Drawing.Color.DarkGoldenrod.ToArgb();
      verts[4].X = 1.0f; verts[4].Y = -1.0f; verts[4].Z = 1.5f; verts[4].Color =
System.Drawing.Color.MediumOrchid.ToArgb();
      verts[5].X = 0.0f; verts[5].Y = 1.0f; verts[5].Z = 0.2f; verts[5].Color =
System.Drawing.Color.Cornsilk.ToArgb();
      verts[6].X = -1.0f; verts[6].Y = -1.0f; verts[6].Z = -1.0f; verts[6].Color =
System.Drawing.Color.Yellow.ToArgb();
      verts[7].X = 1.0f; verts[7].Y = -1.0f; verts[7].Z = -1.0f; verts[7].Color =
System.Drawing.Color.Magenta.ToArgb();
      verts[8].X = 0.0f; verts[8].Y = 1.0f; verts[8].Z = -0.2f; verts[8].Color =
System.Drawing.Color.Cyan.ToArgb();
       vb.Unlock();
              }
              private void Render()
              {       if (device == null)         return;
                      if (pause)                  return;

//Clear the backbuffer to a blue color
   device.Clear(ClearFlags.Target | ClearFlags.ZBuffer  /* !!! */,
System.Drawing.Color.Blue, 1.0f, 0);

                      device.BeginScene();     //Begin the scene
                      // Setup the world, view, and projection matrices
                      SetupMatrices();
```

```
        device.SetStreamSource(0, vertexBuffer, 0);
        device.VertexFormat = CustomVertex.PositionColored.Format;
device.DrawPrimitives(PrimitiveType.TriangleList, 0, n);    ///// !!!
                        //End the scene
                        device.EndScene();
                        device.Present();
                }

                private void SetupMatrices()
                {
// For our world matrix, we will just rotate the object about the y-axis.
// Set up the rotation matrix to generate 1 full rotation (2*PI radians)
// every 1000 ms. To avoid the loss of precision inherent in very high
// floating point numbers, the system time is modulated by the rotation
// period before conversion to a radian angle.
                int  iTime  = Environment.TickCount % 1000;
                float fAngle = iTime * (2.0f * (float)Math.PI) / 1000.0f;
                device.Transform.World = Matrix.RotationY( fAngle );
                device.Transform.View = Matrix.LookAtLH( new Vector3(
0.0f, 3.0f,-5.0f ), new Vector3( 0.0f, 0.0f, 0.0f ), new Vector3( 0.0f, 1.0f, 0.0f )
);

                device.Transform.Projection = Matrix.PerspectiveFovLH(
(float)Math.PI / 4, 1.0f, 1.0f, 100.0f );
                }

                protected override void
OnPaint(System.Windows.Forms.PaintEventArgs e)
                {       this.Render(); }    // Render on painting
                protected override void
OnKeyPress(System.Windows.Forms.KeyPressEventArgs e)
                {
if ((int)(byte)e.KeyChar == (int)System.Windows.Forms.Keys.Escape)
```

```
            this.Close(); // Esc was pressed
            }
        protected override void OnResize(System.EventArgs e)
        {
          pause = ((this.WindowState == FormWindowState.Minimized) ||
!this.Visible);
        }
            /// <summary>
            /// The main entry point for the application.
            /// </summary>
            static void Main()
            {
          using (Matrices frm = new Matrices())
          {
            if (!frm.InitializeGraphics()) // Initialize Direct3D
            {
         MessageBox.Show("Could not initialize Direct3D.  This tutorial will
exit.");
              return;
            }
            frm.Show();
// While the form is still valid, render and process messages
            while(frm.Created)
            {
              frm.Render();
              Application.DoEvents();
            }
          }
        }
         }
}
```

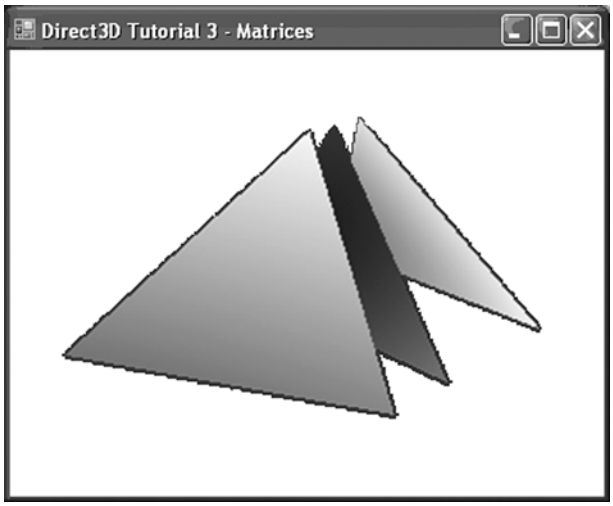


Рисунок 6.2. Пример работы программы по листингу 6.1.

Взять координаты треугольников для своей модели и вставить в пример выполнения. Полученную копию экрана обязательно вставить в отчет о работе

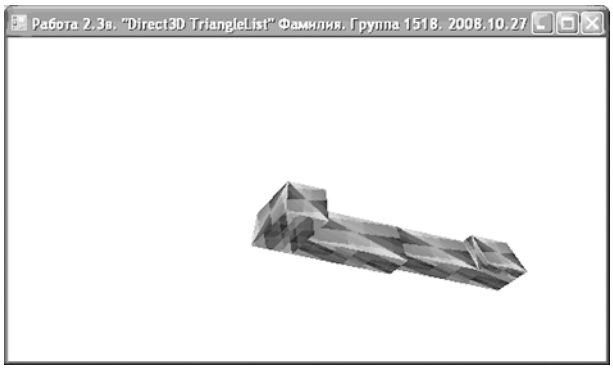


Рисунок 6.3. Пример выполнения трехмерной модели в DirectX

Часть 2. Примеры выполнения самостоятельных работ студентов

В данной части данных указаний к каждому заданию показан пример выполнения.. Кроме того, дается подробное пояснение по выполнению и оформлению самостоятельной работы 1.2.

Глава 7. Примеры заданий и выполненных работ по компьютерной геометрии и графике.

Пример задания по компьютерной геметрии и графике для самостоятельных работ S11 -S13 и S21 -S25. Вариант 282.

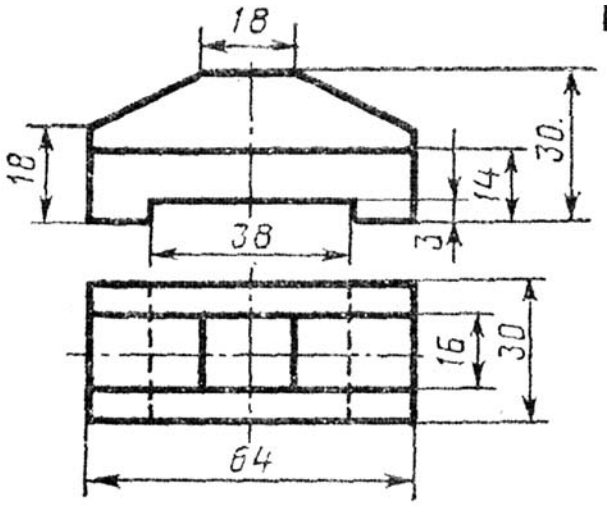


Рисунок 7.1. Пример задания по геометрическому моделированию

Пример задания для самостоятельных работ S14 и S15. Вариант 47.

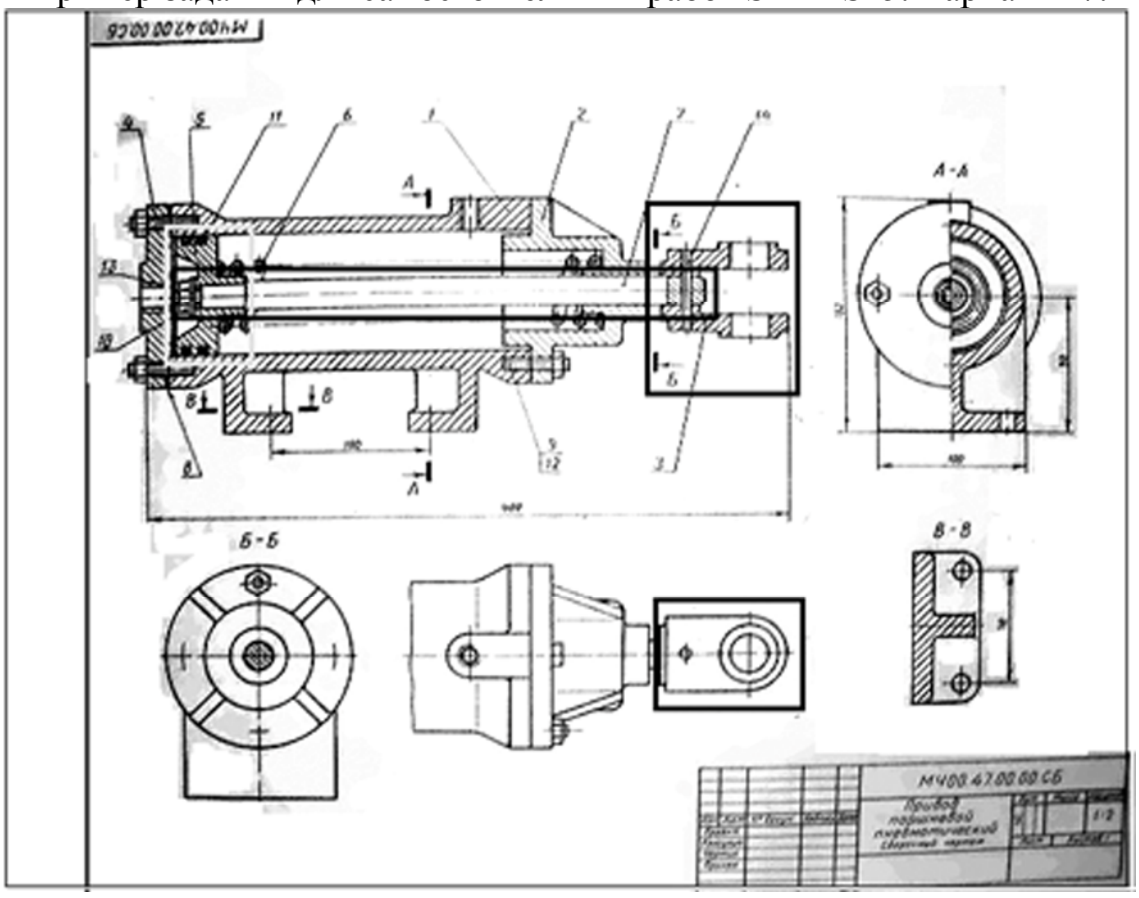


Рисунок 7.2. Пример задания по компьютерному черчению

Так как в СПб ГУ ИТМО с 2006 года введена балльно- рейтинговая система обучения все задания семестра разделены на два модуля. Далее приведены примеры выполненных работ.

Самостоятельная работа 1.1. Создание чертежа ортогональных и аксонометрической проекций детали в «Компас - 3D» (без использования трехмерной модели и ассоциативных видов).

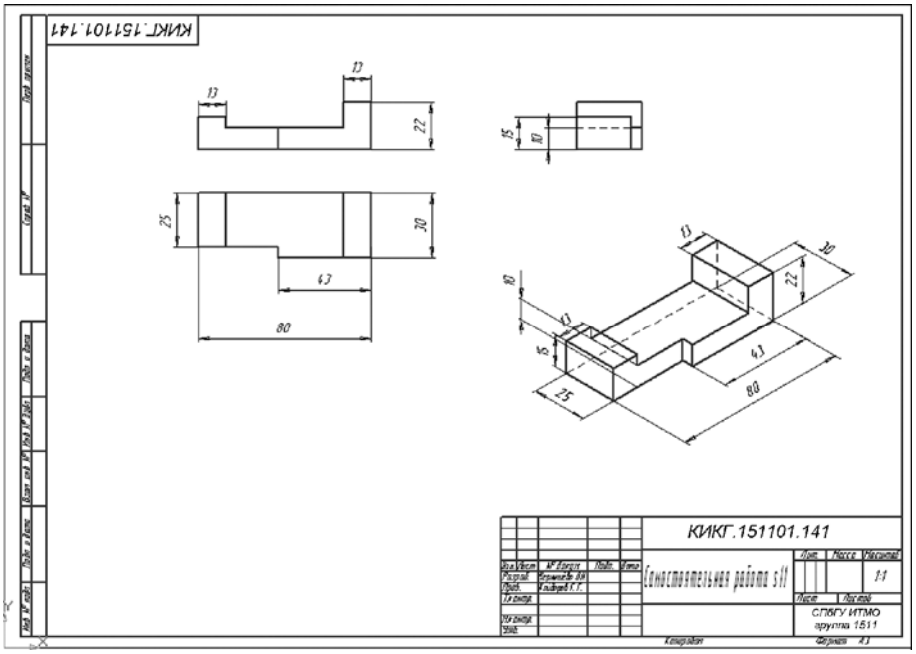


Рисунок 7.3. Пример выполнения самостоятельной работы s11

Самостоятельная работа 1.2.Трехмерная каркасная модель детали средствами базовой графики

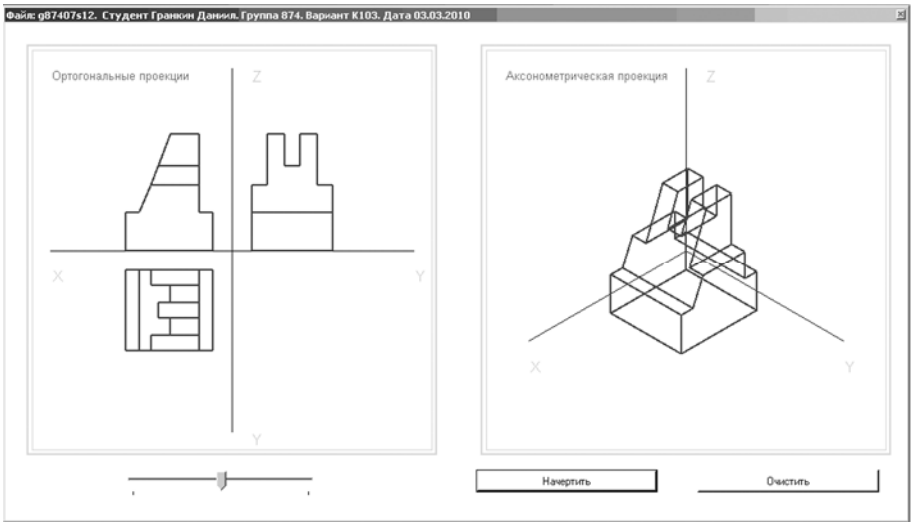


Рисунок 7.4. Пример выполнения самостоятельной работы s12

Самостоятельная работа 1.3. Создание простой трехмерной модели детали в «Компас 3D» и ассоциативных видов

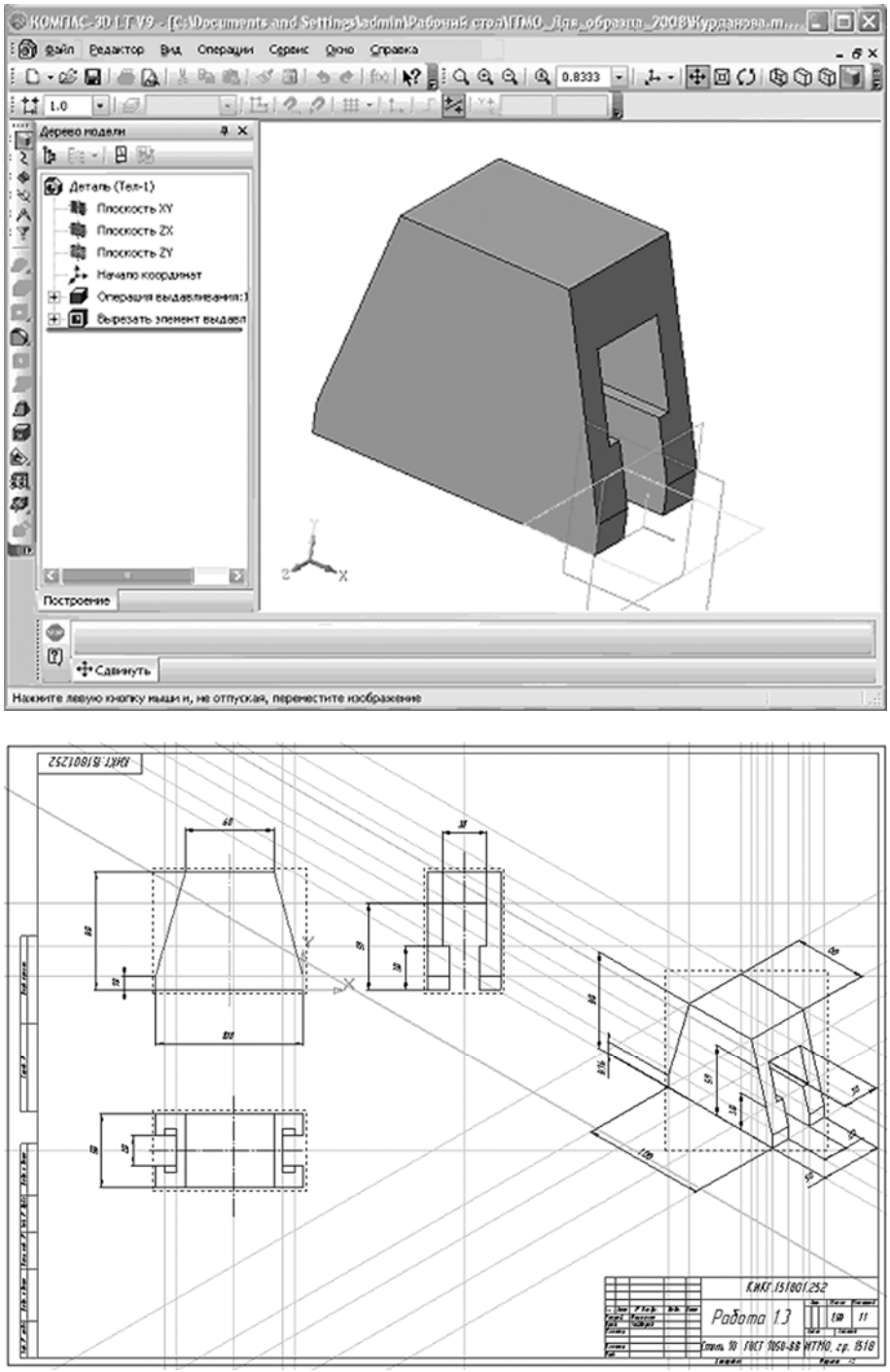


Рисунок 7.5. Пример выполнения самостоятельной работы s13

Самостоятельная работа 1.4. Чтение сборочного чертежа. Создание трехмерных моделей и ассоциативных чертежей в «Компас - 3D»

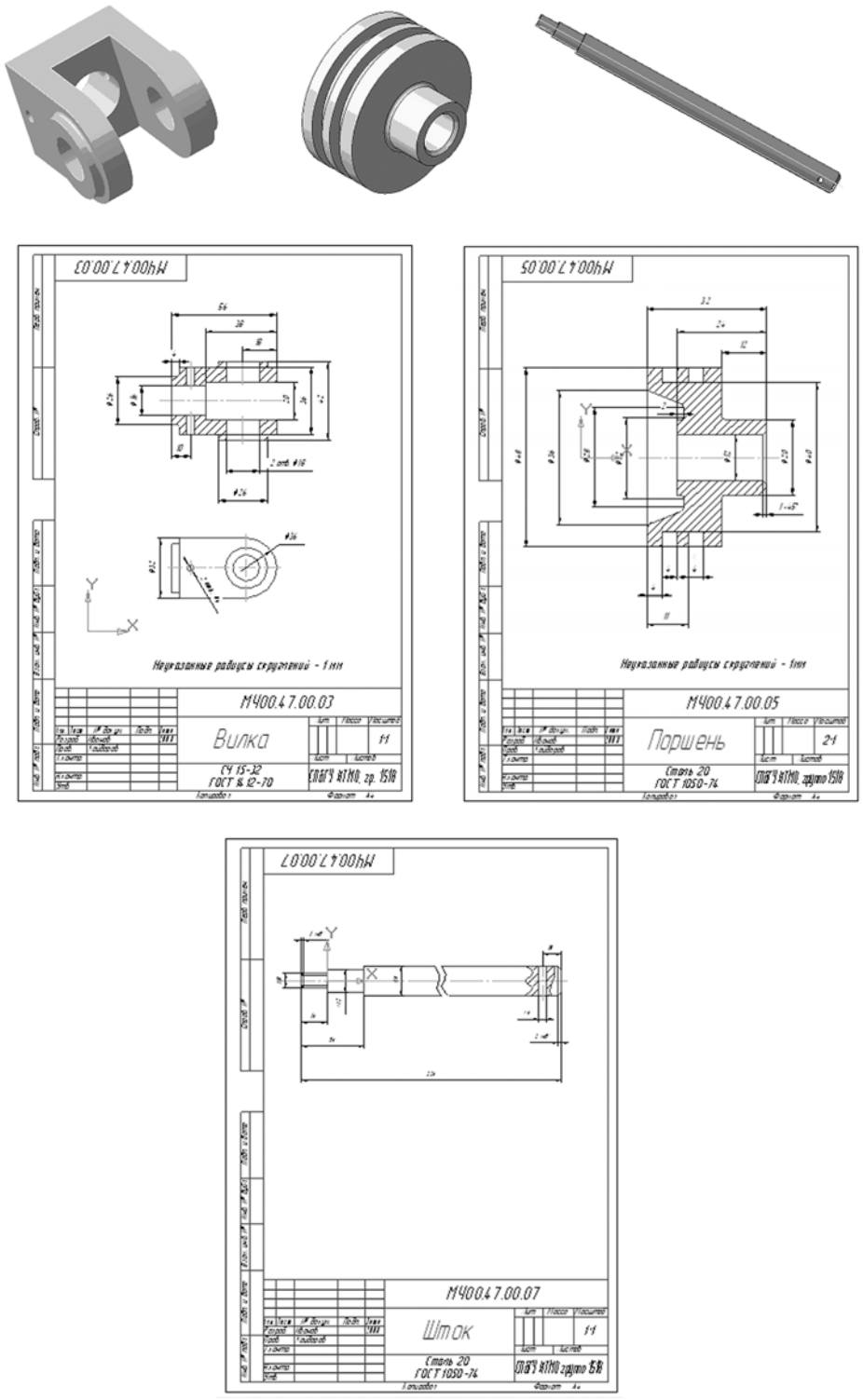


Рисунок 7.6. Пример выполнения самостоятельной работы s14

Самостоятельная работа 1.5. Создание сборки из трехмерных моделей, спецификации и ассоциативного сборочного чертежа в «Компас - 3D»

Дополнительная работа на хорошую или отличную оценку

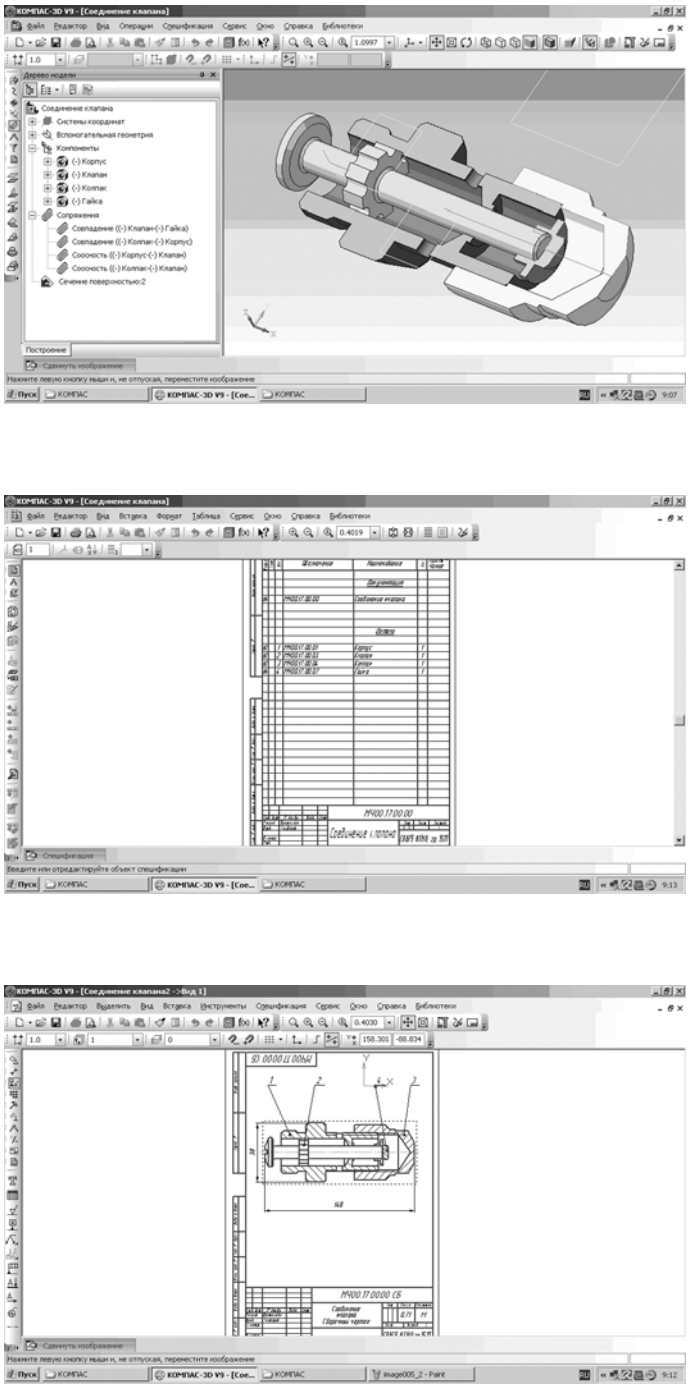


Рисунок 7.7. Пример выполнения самостоятельной работы s15

Модуль 2. Современные технологии программирования трехмерных моделей

Самостоятельная работа 2.1. Создание трехмерной модели на языке VRML

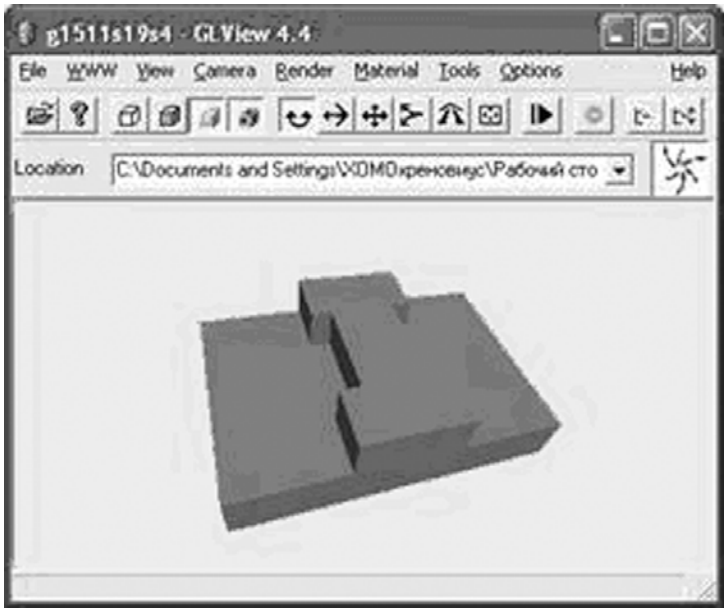


Рисунок 7.8. Пример выполнения самостоятельной работы s21

Самостоятельная работа 2.2. Создание трехмерной модели детали средствами библиотеки OpenGL.

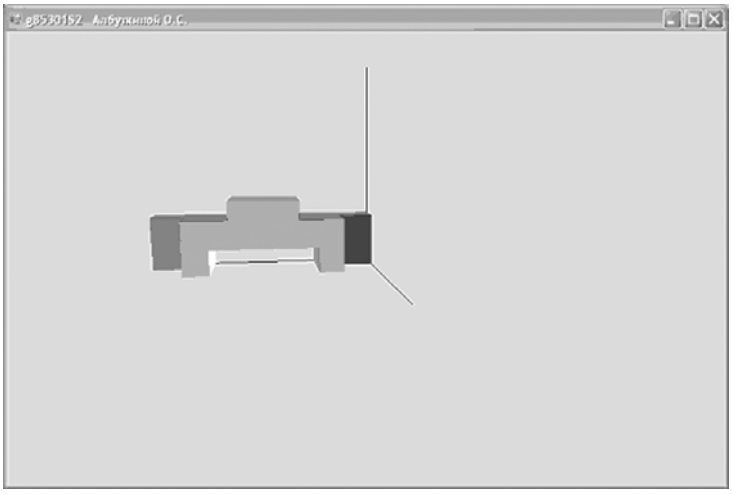


Рисунок 7.9. Пример выполнения самостоятельной работы s22

Самостоятельная работа 2.3. Создание Applet трехмерной модели на языке Java

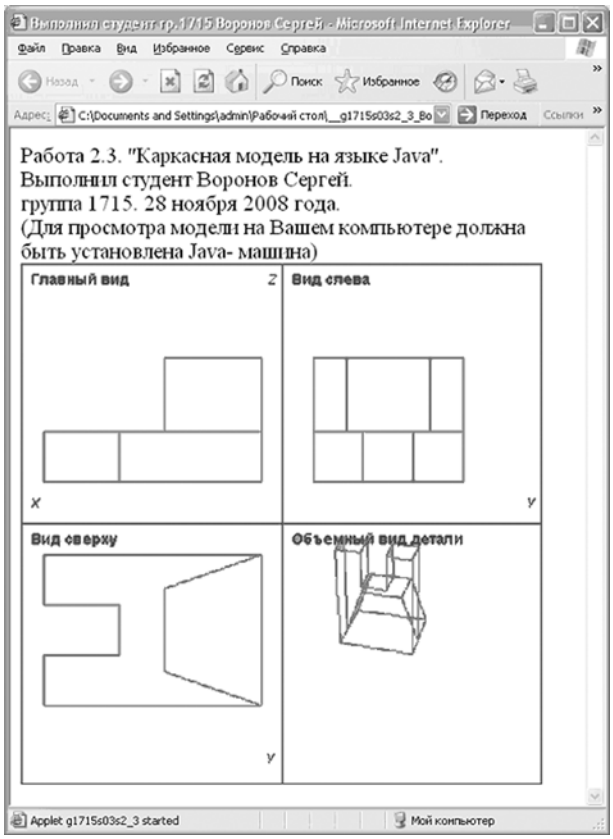


Рисунок 7.10. Пример выполнения самостоятельной работы s23

Самостоятельная работа 2.4. Создание трехмерной модели с применением DirectX

Дополнительная работа на хорошую или отличную оценку

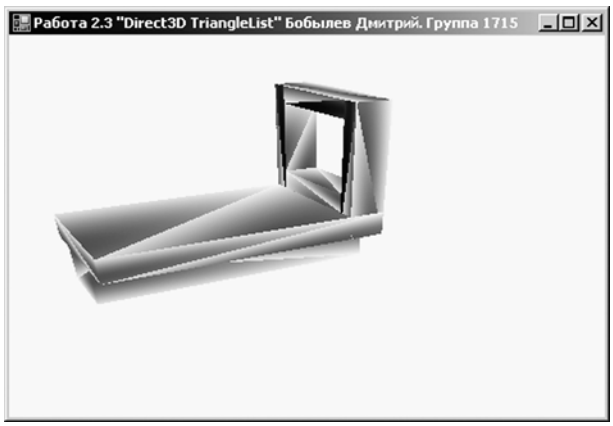


Рисунок 7.11. Пример выполнения самостоятельной работы s24

Самостоятельная работа 2.5. Создание трехмерной модели на управляемой трехмерной сцене на языке C#.

Дополнительная работа на хорошую или отличную оценку

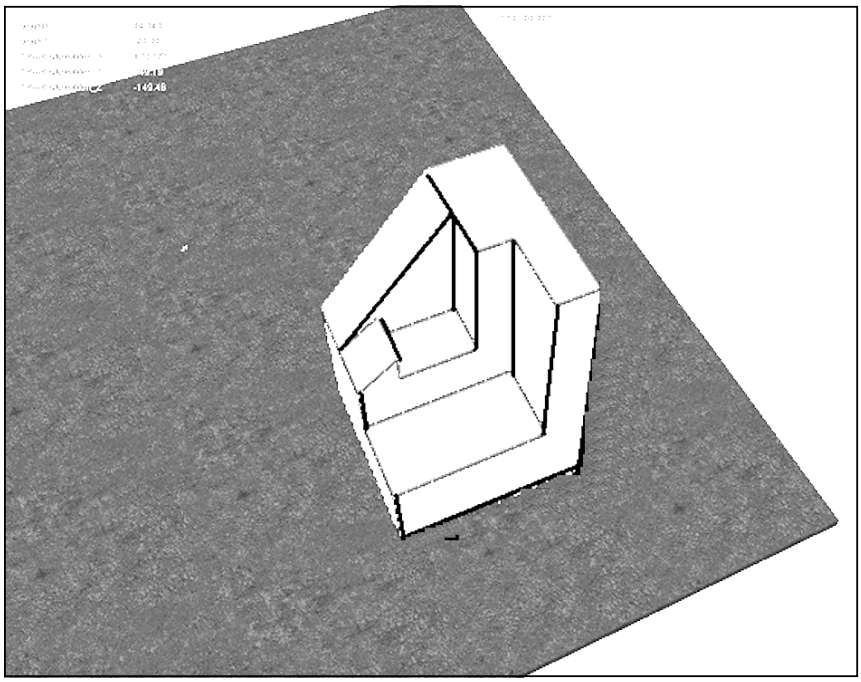


Рисунок 7.12. Пример выполнения самостоятельной работы s25

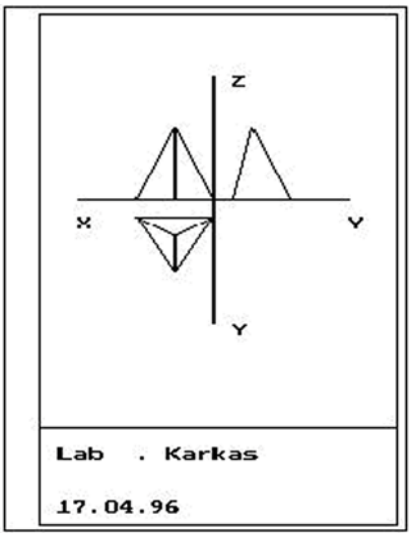
Глава 8. Пример оформления и правила сдачи самостоятельных работ

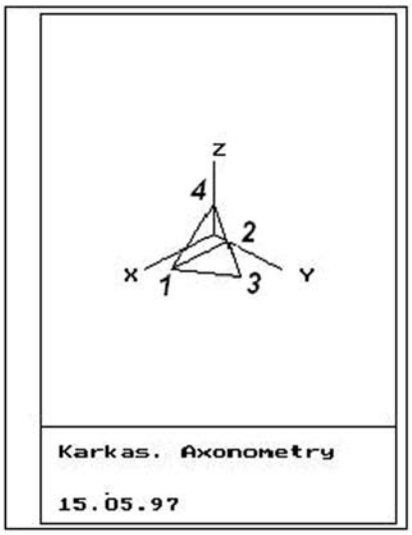
8.1. Пример оформления содержательной части отчета по работе s12

Данная работа связана с программированием каркасной модели. В отчет необходимо включить:

- рисунок задания (рисунок 7.1),
- решение (в данной работе это таблицы точек и линий),
- листинг программы в C# (для работы по программированию),
- экранную копию работающей программы (рисунок 7.4),
- литература и интернет-ссылки.

Для ускорения работы над заданием желательно нарисовать эскизы ортогональной (рис 8.1) и аксонометрической (рис 8.2) проекций детали, а затем составить таблицы для координат вершин (таблица 8.1) и линий (таблица 8.2). Далее в тексте программы по указанному образцу вписываются свои значения координат из таблицы линий.





Рисунки 8.1 и 8.2 Ортогональные и аксонометрическая проекции каркасной модели.

Таблица 8.1 Значения координат вершин пирамиды

Номер вершины	X	Y	Z
1	40	10	0
2	0	10	0
3	20	40	0
4	20	20	40

Таблица 8.2 Значения координат начала и конца линии ребер каркасной модели

Номер линии (между вершинами)	Значения координат начала линии			Значения координат конца линии		
	Xn	Yn	Zn	Xk	Yk	Zk
0 (1-2)	40	10	0	0	10	0
1 (2-3)	0	10	0	20	40	0
2 (3-1)	20	40	0	40	10	0
3 (1-4)	40	10	0	20	20	40
4 (2-4)	0	10	0	20	20	40
5 (3-4)	20	40	0	20	20	40

Листинг 8.1. Пример выполнения задания s12.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace g1511XXs1_FIO_WindowsApplication1
{
        /// <summary>
        /// Summary description for Form1.
        /// </summary>
        public class Form1 : System.Windows.Forms.Form
        {
                private System.Windows.Forms.Button button1;
                private System.Windows.Forms.Label label1;
                private System.ComponentModel.IContainer components;
                public Form1()
                {
                //
                // Required for Windows Form Designer support
                //
                        InitializeComponent();
// TODO: Add any constructor code after InitializeComponent call
                }
/// <summary>
/// Clean up any resources being used.
        void line(Pen pen,int x1,int y1,int x2, int y2)
        {
        Graphics canvas= Graphics.FromHwnd(this.Handle);
        canvas.DrawLine(pen,x1,y1,x2,y2);}
```

```
/// </summary>
protected override void Dispose( bool disposing )
{
if( disposing )
{
if (components != null)
{
components.Dispose();
}
}
base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
this.button1 = new System.Windows.Forms.Button();
this.label1 = new System.Windows.Forms.Label();
this.SuspendLayout();
//
// button1
//
this.button1.Cursor = System.Windows.Forms.Cursors.Hand;
this.button1.Location = new System.Drawing.Point(48, 304);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(128, 23);
this.button1.TabIndex = 0;
```

```
        this.button1.Text = "button1 Показать";
this.button1.Click += new System.EventHandler(this.button1_Click);
        //
        // label1
        //
        this.label1.Location = new System.Drawing.Point(224, 304);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(272, 16);
        this.label1.TabIndex = 1;
        this.label1.Text = "label1";
        //
        // Form1
        //
        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
        this.ClientSize = new System.Drawing.Size(512, 334);
        this.Controls.Add(this.label1);
        this.Controls.Add(this.button1);
        this.MaximizeBox = false;
        this.Name = "Form1";
this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
this.Text = "g1511XXs1_Фамилия Самостоятельная работа №1
12.12.2006";
this.Load += new System.EventHandler(this.Form1_Load);
this.Activated += new System.EventHandler(this.Form1_Activated);
this.ResumeLayout(false);

            }
            #endregion
            /// <summary>
            /// The main entry point for the application.
            /// </summary>
```

```
            [STAThread]
            static void Main()
            {
                  Application.Run(new Form1());
            }
private void button1_Click(object sender, System.EventArgs e)
            {
      this.SetStyle(ControlStyles.SupportsTransparentBackColor,true);
      this.SetStyle(ControlStyles.SupportsTransparentBackColor,true);
      this.BackColor=Color.FromArgb(1.0,0,0,10);
//                                          this.Invalidate();

//////////////    this.MaximizeBox = false;
//////////////    this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
      /* Установка параметров */
            int x0,y0,i;  x0=110;y0=110;

      /*    1. Ввод значений исходных данных */
            int n=14;
int [ ] xn = { 40, 0,20,40, 0,20,25,22,22,25,18,18,15,15};
int[]  yn={ 10,10,40,10,10,40,22,22,25,28,28,22,22,28};
int[]  zn={  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int[]  xk={  0,20,40,20,20,20,22,22,25,22,18,15,15,18};
int[]  yk={ 10,40,10,20,20,20,22,25,28,28,22,22,28,28};
int[]  zk={  0, 0, 0,40,40,40, 0, 0, 0, 0, 0, 0, 0, 0};

            /*    2. Инициализация графики  */
Graphics canvas= Graphics.FromHwnd(this.Handle);
            float w=1.0f,w0=2.0f;
            Pen pen=new Pen(Color.Red,w0);
            Pen pen2=new Pen(Color.Green,w);
```

```
SolidBrush b=new SolidBrush(Color.FromArgb(150,0,0,255));
        Font f=new Font("Arial",15);
                Font f2=new Font("Arial",8);
                /*          вычерчивание рамки формата А4*/
                canvas.DrawRectangle(pen2,2,2,210,297);
                canvas.DrawRectangle(pen2,20,5,185,287);
                line(pen2,20,297-55,205,297-55 );

                /* Оси координат и их обозначение */
        line(pen2,x0-70,y0,x0+70,y0 );          line(pen2,x0,y0-70,x0,y0+70);
                canvas.DrawString("X",new Font("Areal",15),b,x0-70,y0+10);
                canvas.DrawString("Y",f,b,x0+70,y0+10);
                canvas.DrawString("Z",f,b,x0+10,y0-70);
                canvas.DrawString("Y",f,b,x0+10,y0+70);
                canvas.DrawString("  Ортогональные проекции"
,f2,b,25,270);

                /* Основные циклы вычислений проекции XOY*/
                for(i=0;i<n;i++)
                { //DrawLine(pen,int,int,int,int)
canvas.DrawLine(pen,x0-xn[i],y0+yn[i],x0-xk[i],y0+yk[i]);
                        };

                /* Основные циклы вычислений проекции XOZ*/
                for(i=0;i<n;i++)
                {  //My function line(pen,int,int,int,int)
                line(pen,x0-xn[i],y0-zn[i],x0-xk[i],y0-zk[i] );
                };

                /* Основные циклы вычислений проекции ZOY*/
                float k=1.0f; // масштаб
                for(i=0;i<n;i++)
```

```
            {  //DrawLine(pen,float,float,float,float)
            canvas.DrawLine(pen,1.0f*(x0+k*yn[i]),
            (y0-k*zn[i]),(x0+k*yk[i]),(y0-k*zk[i]) );
            };


private void Form1_Activated(object sender, System.EventArgs e)
            {
                  button1_Click(null, null);
            }

private void Form1_Load(object sender, System.EventArgs e)
            {
this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
                  label1.Text="";
      this.SetStyle(ControlStyles.SupportsTransparentBackColor,true);

                  Rectangle r=new Rectangle(0,0,200,300);

      r=System.Windows.Forms.Screen.PrimaryScreen.Bounds;
                        w=r.Width;h=r.Height;
                        tw=this.Width;th=this.Height;
                        this.Left=(w-tw)/2;this.Top=(h-th)/2;
            }


      }
}
```

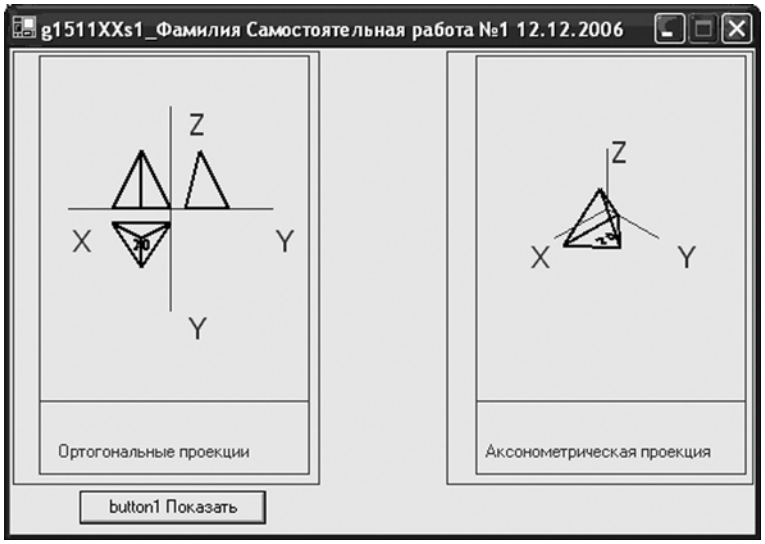


Рисунок 8.3. Экранная копия готовой программы s12.

8.2 Правила сдачи самостоятельной работы

Результаты самостоятельной работы по выдаваемому в электронном виде образцу. Сдается отчет по работе. К отчету прилагается исходные и исполняемые файлы программы. Нумерация файлов и папки осуществляется по личному коду, выдаваемому преподавателем. Все материалы сдаются в электронном виде. Если самостоятельная работа сдается после окончания учебного семестра, то кроме электронных материалов сдается на проверку распечатка отчета.

Например, имя папки: g151101s12_ФамилияСтудента, где g- студенческая группа, 1511- номер группы, 01- номер студента по списку, s- самостоятельная работа, 12- порядковый номер работы. В папке находятся следующие файлы: g151101s12.doc- файл отчета по работе. Кроме того в папке находятся все файлы чертежей: *.cdw, *.m3d (для работы в «Компас-3D») или все исходные фалы проекта, включая исходный файл *.cs и исполняемым файл *.exe в папке bin (для работы по программированию трехмерной модели).

8.3 Дополнительные индивидуальные задания

Кроме типового задания по согласованию с преподавателем можно выполнить дополнительное индивидуальное задание, в котором студент может показать свои знания и умения работы. Например, дополнить графику анимацией или матрицей трехмерных преобразований. Или выполнить полную сборку:

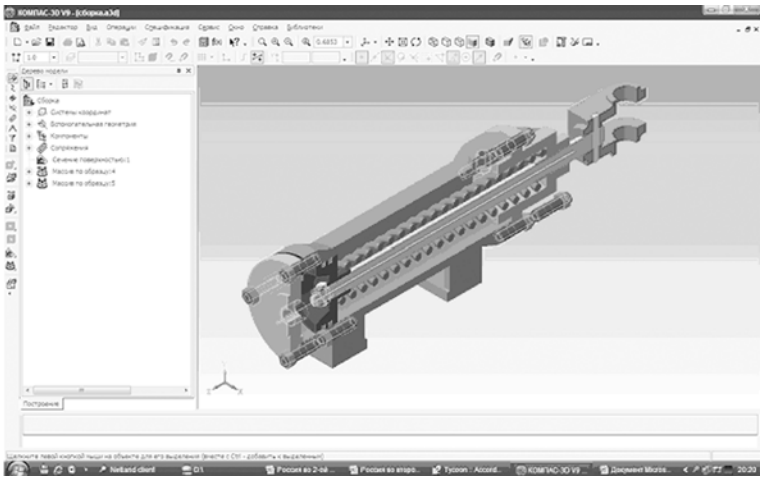


Рисунок 8.4. Полная сборка в «Компас -3D»

Глава 9. Примеры применение методов компьютерной графики

Комплексное использование методов компьютерной геометрии и графики и их модификаций в современном программировании позволяет получить интересные результаты. Рассмотрим ряд программ выполненных с применением данных методов.

9.1 Конструктор помещений

В данной программе создан редактор для конструирования помещений. В результате работы данной программы создается файл трехмерного текстового формата, который можно загрузить в трехмерную управляемую сцену.

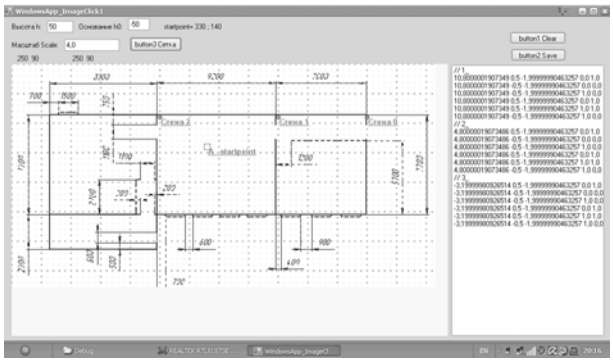


Рисунок 9.1. Экранная копия работы программы

9.2 Создание трехмерных объектов для сцены

Объекты можно создавать в любом профессиональном трехмерном редакторе, поддерживающем конвертирование трехмерных форматов. Например, в качестве такого редактора можно взять полную (профессиональную) версию «Компас -3D».

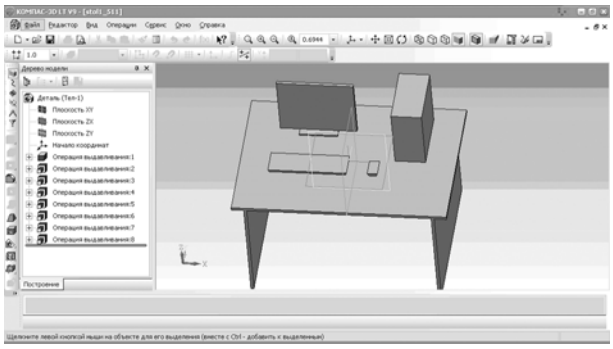


Рисунок 9.2 Экранная копия работы редактора объектов

9.3 Управляемая трехмерная сцена

Данная программа позволяет подгружать в управляемую трехмерную сцену с помещением трехмерные объекты. На рисунках 9.3 и 9.4 приведен пример создания виртуального помещения (аудитории 130, 131, 132) центра компьютерных технологий отдела техники Санкт-Петербургского городского дворца творчества юных.

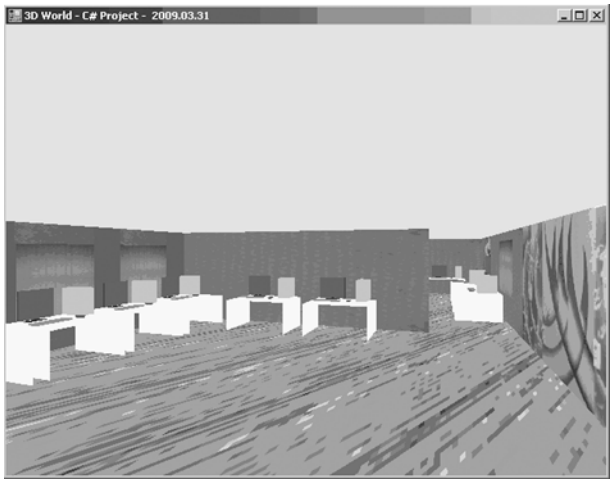


Рисунок 9.3 Экранная копия работы программы

Рисунок 9.4 Экранная копия работы программы

10. Контрольные вопросы

Вопросы по теме 1 «Методы проецирования и системы координат в компьютерной графике»

1. Области применения компьютерной геометрии и графики.
2. Растровая и векторная графика. Какие форматы графических файлов вы знаете?
3. Назовите наименьший элемент растрового изображения. Назовите наименьший элемент векторного изображения.
4. Форматы графических данных (файлов). Сжатие графических данных.
5. Графические форматы BMP и WMF: области применения, преимущества и недостатки, особенности.
6. Графические форматы GIF и PNG: области применения, преимущества и недостатки, особенности.
7. Графические форматы PSD и CDR: области применения, преимущества и недостатки, особенности.
8. Графические форматы JPEG и TIFF: области применения, преимущества и недостатки, особенности.
9. Графические форматы CWD, DWG и DXF: области применения, преимущества и недостатки, особенности
10. Цвет в КГ. Аддитивные и субтрактивные цвета. Системы RGB, CMYK.
11. Почему цветовую модель RGB называют аддитивной? Почему цветовую модель CMYK называют субтрактивной?
12. Центральное и ортогональное проецирование.
13. Точка общего положения на эпюре и в косоугольной фронтальной диметрической проекции.
14. Точка общего положения на эпюре и в прямоугольной изометрической проекции.
15. Прямая общего положения на эпюре и в косоугольной фронтальной диметрической проекции.
16. Прямая общего положения на эпюре и в прямоугольной изометрической проекции.
17. Виды аксонометрических проекций по Российскому стандарту.

Вопросы по теме 2 «Основные функции базовой графики и разработка трехмерной каркасной модели»

18. История языков программирования и компьютерной графики. Базовые графические функции.
19. Графические примитивы рисования точки, линии и прямоугольника.

20. Графические примитивы рисования закрашенного прямоугольника и установка стандартных стилей закраски.
21. Установка цвета и толщины графических примитивов, функции вывода графического текста.
22. Написать и объяснить формулы для перевода координат ортогональных проекций точки в экранные координаты и фрагмент программы.
23. Написать и объяснить формулы для перевода координат аксонометрических проекций точки в экранные координаты и фрагмент программы (на примере изометрии).
24. Написать и объяснить формулы для перевода координат аксонометрических проекций точки в экранные координаты и фрагмент программы (на примере фронтальной диметрии).
25. Перспективы и направления развития С++ язычных сред программирования.
26. Структура программы на C#.
27. Последовательность работы на языке C# в средах MS C# Express и Visual Studio.NET.
28. Виды и последовательность компиляция исходного файла в C#
29. Виды и последовательность компиляция исходного файла в Java

Вопросы по теме 3 «Российские стандарты единой системы конструкторской документации и создание чертежа детали в «Компас-3D»

30. Для чего предназначены растровые графические редакторы? Приведите примеры редакторов, опишите достоинства каждого из редакторов.
31. Для чего предназначены векторные графические редакторы? Приведите примеры редакторов, назовите форматы файлов, опишите достоинства каждого из редакторов.
32. Выбор и установка основных параметров для выполнения графической документации в графическом редакторе «Компас 3D». Форматы. Масштабы. Линии. Шрифты. Виды. Нанесение размеров.
33. Форматы листов чертежей установленных ГОСТ 2.301-68. Обозначение и размеры основных форматов.
34. Масштабы - изображений (уменьшения, натуральная величина, увеличение) и их обозначение на чертежах.
35. Наименование, начертание, основные назначения линий. Толщина их по отношению к толщине основной линии чертежа.
36. Размеры шрифта, установленные стандартом.
37. Дать определение вида. Перечислить названия основных видов. Дополнительные виды. Случаи их применения и правила обозначения на чертежах.

38. Назначение разрезов при выполнении чертежей изделий. Определение разреза.
39. Определение. Отличие сечения от разреза (в общем случае).
40. Выбор и установка привязок в графическом редакторе «Компас 3D». Установка точных расстояний и углов. Применение компактной панели: геометрия. Основные геометрические примитивы. Применение соответствующей панели свойств.
41. Типы размеров. Применение компактной панели: размеры.
42. Общее количество размеров на чертеже. В каких единицах измерения указывают линейные, а также угловые размеры?
43. Покажите на примерах нанесения размеров диаметра (радиуса) сферы квадрата, фасок под углом 45° и под другими углами.
44. Типы обозначений на чертежах: текст, разрезы и сечения, вид, выноски. Применение компактной панели: обозначения. Применение соответствующей панели свойств.
45. Измерения на чертежах. Применение компактной панели: измерение. Применение соответствующей панели свойств.
46. Редактирование на чертежах: сдвиг, поворот, масштабирование, симметрия и другие. Применение компактной панели: редактирование. Применение соответствующей панели свойств.
47. Изобразить резьбу на стержне с фаской на видах, полученных проецированием на плоскости, параллельную и перпендикулярную к оси стержня.
48. Изобразить резьбу в отверстии с фаской на разрезе, параллельном оси отверстия и на виде на плоскость, перпендикулярную к оси.
49. Как следует указывать на чертеже границу резьбы и наносить штриховку в разрезах и сечениях металлических стержней и отверстий с резьбой?
50. Перечислить пять параметров, характеризующих резьбу. На примере метрической или трапецеидальной резьбы (привести примеры обозначения резьбы).
51. Последовательность выполнения двухмерного чертежа детали по требованиям ЕСКД.
52. Выполнить двухмерный чертеж детали по выданному двухмерному эскизу и по требованиям ЕСКД.

Вопросы по теме 4 «Последовательность создания трехмерной модели детали в «Компас 3D»

53. Последовательность создания трехмерной модели детали. Типы операций для создания 3D моделй.
54. Вырезать выдавливанием часть детали. Применение соответствующей панели свойств.

55. Приклеить выдавливанием часть детали. Применение соответствующей панели свойств.
56. Выполнить трехмерную модель детали по выданному двухмерному эскизу.
57. Требования ЕСКД по выполнению графической документации в графическом редакторе «Компас 3D». Изображения- виды, сечения, разрезы. Нанесение размеров. Аксонометрические проекции.

Вопросы по теме 5 «Геометрически преобразования трехмерных объектов и обзор алгоритмов компьютерной графики**»**

58. Линейное преобразование фигур на плоскости. Матрица линейного преобразования. Назначение коэффициентов матрицы линейного преобразования на плоскости (изменение масштаба, вращение). Написать фрагмент программы для модификации изображения квадрата средствами базовой графики.
59. Линейное преобразование фигур на плоскости. Матрица линейного преобразования. Назначение коэффициентов матрицы линейного преобразования на плоскости (например, повороты на углы, кратные 90°). Написать фрагмент программы для модификации изображения квадрата средствами базовой графики.
60. Однородные координаты в пространстве. Назначение элементов расширенной матрицы линейного преобразования (изменение масштаба, перемещение и вращение вокруг оси).
61. Ортогональные и аксонометрические проекции. Диметрические проекции. Изометрическая проекция. Получение наглядных перспективных проекций.
62. Алгоритмы, использующие z-буфер. Основная идея.

Вопросы по теме 6 «Графическая библиотека OpenGL и создание трехмерных моделей**»**

63. Применение OpenGL. Описание общей структуры команд о OpenGL.
64. Применение OpenGL. Описание примитивов вывода отрезков.
65. Применение OpenGL. Описание примитивов вывода треугольников.
66. Применение OpenGL. Описание примитивов вывода четырехугольников.
67. Применение OpenGL. Описание примитивов вывода многоугольников.
68. Применение OpenGL. Описание команд перемещения, вращения и масштабирования объектов.
69. Применение OpenGL. Описание команд для отображения ортогональных и перспективных проекций.
70. Применение OpenGL. Описание команд для инициализации, перерисовки и рисования.

71. Применение OpenGL. Описание, назначение и пример команд для сброса (очиски) glClear….

Вопросы по теме 7 «Создание трехмерной модели для интернета на языках VRML и Java. Базовая графика на языке Java**»**

72. Структура программы на языке VRML.
73. Написать фрагмент программы с массивом исходных координат и объяснить последовательность действий для вывода изображения равносторонней пирамиды средствами VRML.
74. Структура программы на Java.
75. Последовательность компиляции исходного файла.
76. Последовательность создания апплета
77. Среды программирования на Java
78. Графические примитивы рисования точки, линии и прямоугольника.
79. Графические примитивы рисования закрашенного прямоугольника и установка стандартных стилей закраски.
80. Установка цвета и толщины графических примитивов, функции вывода графического текста.

Вопросы по теме 8 «Технические средства компьютерной графики».

81. Типы и характеристики мониторов В каких единицах измеряют: разрешение экрана, разрешение принтера, разрешение изображения?
82. Как задается цвет пикселя в режиме True Color? Сколько байтов оперативной памяти для этого нужно?
83. Видеопамять, ее необходимое количество для различных режимов работы.
84. Классификация устройств вывода. Классификация дисплеев. Векторные дисплеи. Растровые дисплеи. Плазменная панель. Жидкокристаллические индикаторы.
85. Печатающие устройства. Разрешение устройств. Классификация и принцип действия принтеров.
86. Графопостроители. Классификация. Планшетные графопостроители. Графопостроители с переметающимся носителем.
87. Классификация и принцип действия устройств ввода. Клавиатуры, кнопки, световое перо. Мышь, трекбол, джойстик. Планшеты.

Литература

Основная литература

1. Хайдаров Г.Г. Примеры выполнения самостоятельных работ по компьютерной геометрии и графике,: Метод. Указания. СПб, СПбГУ ИТМО, 2006, -52с.
2. Большаков В.П. Инженерная и компьютерная графика. Практикум – СПб.: БХВ, 2004. -592с.
3. Инженерная графика: учеб. пособие для студ. высш. учеб. заведений /В.В. Елкин, В.Т. Тозик. –М.: Издательский центр «Академия», 2008. – 304 с.
4. Бочков А.Л. Трехмерное моделирование в системе «Компас-3D» (практическое руководство). –СПб.: СПбГУ ИТМО, 2007. -84с.
5. Павловская Т. А. C#. Программирование на языке высокого уровня. Учебник для вузов. –СПб.: Питер, 2007. -432 с.
6. Троелсен Э. C# и платформа .NET. Библиотека программиста. -СПб.: Питер, 2006. – 796с.

Дополнительная литература

7. Герман О.В., Герман Ю.О. Программирование на Java и C# для студентов. -СПб.: БХВ-Петербург, 2005. – 512с.
8. Боресков А.В. Графика трехмерной компьютерной игры на основе OpenGL. - М.: ДИАЛОГ-МИФИ, 2004.-384с
9. Хайдаров Г.Г., Алексеев С.Ю. Примеры выполнения лабораторных работ по алгоритмам компьютерной графики.: Метод. указания. СПб, СПбГТИ (ТУ), 2005, -30с
10. Порев В. Н. Компьютерная графика. -СПб.: БХВ-Петербург, 2002. – 432с.
11. Шикин А.В., Боресков А.В., Компьютерная графика. Полигональные модели. –М.: ДИАЛОГ-МИФИ, 2001. -464с.
12. Агуров В.П. C#. Сборник рецептов. -СПб.: БХВ-Петербург,2007. -432 с.

Содержание

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы.

КАФЕДРА ИНЖЕНЕРНОЙ И КОМПЬЮТЕРНОЙ ГРАФИКИ

Кафедра осуществляет подготовку студентов по трем новым специальностям: 050501.04 ПРОФЕССИОНАЛЬНОЕ ОБУЧЕНИЕ (дизайн) , 050501.06 ПРОФЕССИОНАЛЬНОЕ ОБУЧЕНИЕ (информатика, вычислительная техника и компьютерные технологии) и 230203 "ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ДИЗАЙНЕ".

В настоящее время кафедрой по учебным планам этих специальностей читаются более 80 различных дисциплин. Кафедра обладает развитой материально-технической базой, имеет изостудию, чертежный зал, восемь учебно-научных лабораторий компьютерной графики и мультимедиа, оснащенных локальной сетью, в которой более 130 компьютеров. Оборудование компьютерных классов кафедры удовлетворяет самым современным требованиям, предъявляемым к системам компьютерной графики. С каждого рабочего места можно выйти в глобальную сеть Интернет. Лаборатории имеют более 10 различных учебно-научных Web-серверов, электронную почту, оснащены аппаратурой записи и цифровой обработки видеоизображения; аппаратурой синтеза и обработки звука (в т.ч. midi-клавиатуры) и т. п. В 2002 году создается первый в России электронный учебник по НГ, обладающий возможностями трехмерной анимации и интерактивного взаимодействия с читателем. Начиная с 2005 года, регулярно издаются монографии и учебники в центральных издательствах РФ.

Геннадий Гасимович Хайдаров,

Вячеслав Трофимович Тозик

Компьютерные технологии трехмерного моделирования

Учебное пособие

к самостоятельным работам

по дисциплине «компьютерная геометрия и графика»

В авторской редакции

Дизайн

Верстка

Редакционно-издательский отдел Санкт-Петербургского государственного университета информационных технологий, механики и оптики

Зав. РИО Н.Ф. Гусарова

Лицензия ИД № 00408 от 05.11.99

Подписано к печати

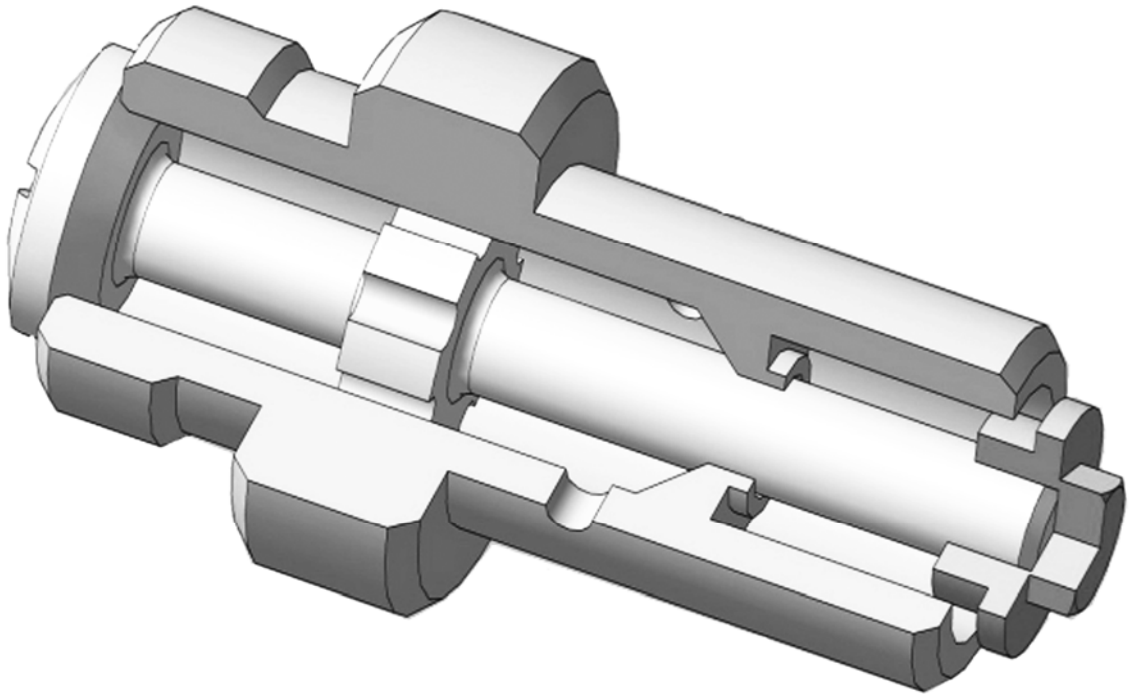
Заказ №

Тираж 100 экз.

Отпечатано на ризографе

Г. Г. Хайдаров , В.Т. Тозик

Компьютерные технологии трехмерного моделирования



Санкт-Петербург
2010

Редакционно-издательский отдел

Санкт-Петербургского государственного университета информационных технологий, механики и оптики

197101, Санкт-Петербург, Кронверкский пр., 49