МИНИСТЕРСТВА ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ УЗБЕКИСТАН

НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ УЗБЕКИСТАНА

Хайдаров А.

ВИЗУАЛИЗАЦИЯ
(Конспект лекций)

Ташкент 2017

Оглавление

I. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА И ОСНОВНЫЕ ПРИНЦИПЫ ПОСТРОЕНИЯ ПРИЛОЖЕНИЙ ТРЕХМЕРНОЙ КОМПЬЮТЕРНОЙ ГРАФИКИ

Введение

Информационные и коммуникационные технологии в XX веке являлись основанием для научно-технического прогресса. Поэтому очевидно, что они будут играть важнейшую роль во всех сферах жизни человечества и в третьем тысячелетии.

В таких областях как кинематография, издательское и банковское дело, в образовательных учреждениях внедрение этих технологий уже произвело поистине революционный переворот. Объединение компьютерной графики, сетей, систем автоматического управления открыло новые пути использования и отображения информации, «проникновения» в виртуальный мир и организации взаимодействия человека и машины.

Компьютерная графика (computer graphics) – это область информатики (науки о компьютерах – computer sciences), которая изучает все вопросы, связанные с обработкой и синтезом изображения с помощью ЭВМ.

Эта область начала развиваться около 40 лет назад. В те годы удавалось добиться отображения нескольких десятков отрезков на экране электронно-лучевой трубки (ЭЛТ), а современные системы машинной графики позволяют создавать изображения, практически, не отличимые по качеству от фотографических снимков реальных объектов. Общепринятой практикой стало сейчас обучение пилотов с помощью систем моделирования реальной ситуации, как она видится из кабины самолета во время пилотирования, создание изображения виртуального динамического мира во всем его многообразии в реальном масштабе времени. На экран выводят полнометражные кинофильмы, в которых нет ни одного кадра, снятого «на натуре» или в павильоне киностудии, а все действия разворачиваются в памяти компьютера.

Развитие компьютерной графики идет бурно и неравномерно – что-то удивительно быстро устаревает, что-то обретает более отчётливые формы, появляется и очень много нового. Постоянно

расширяющиеся возможности доступных вычислительных средств корректируют набор используемых методов и эффективно применяемых алгоритмов. За последние годы не только возросли функциональные возможности средств компьютерной графики, но и значительно снизилась стоимость графических станций, при чем это характерно для установок всех классов, как простейших, так и профессиональных. В течение нескольких лет стоимость графической станции, способной формировать около одного миллиона трехмерных треугольников в секунду с учетом эффектов освещения и наложения текстуры понизилась со $ 100 000 до нескольких тысяч. Значительно более доступными по цене стали и специализированные графические платы для персональных компьютеров.

В области программного обеспечения также произошли большие изменения, особенно за последние годы. Появилось много различных графических библиотек API, среди которых наиболее популярные Direct3D и OpenGL.

Библиотека OpenGL стала своего рода стандартным интерфейсом для программистов, как при написании прикладных программ, так и при разработке программных продуктов высокого класса – от интерактивных игр до систем визуализации результатов научных исследований. Для решения этих же задач можно применять также и Direct3D, поэтому важно изучить особенности различных графических API, чтобы выбрать стандарт, позволяющий наилучшим образом решить поставленную задачу.

1 Области применения компьютерной графики

Развитие компьютерной графики определяется различными факторами, среди которых наиболее существенные:

1) реальные потребности потенциальных пользователей в системах визуализации;

2) достижения в области аппаратного и программного обеспечения.

Развитие компьютерной графики определяет её степень применения во все сферы жизни современного общества. В последние годы компьютерная графика всё больше и больше используется в различных отраслях промышленности, в науке и

образовании, бизнесе и т.д. Однако, можно выделить четыре главные области ее основного применения:

1. Отображение информации.
2. Проектирование.
3. Моделирование.
4. Пользовательский интерфейс.

Во многих практических приложениях можно обнаружить характерные признаки двух и более перечисленных групп, но развитие каждой группы шло своим путем.

1.1 Отображение информации

Зрительная система человека обладает большими возможностями по восприятию информации, чем, например, слуховой аппарат, поскольку выполняет функции и обработки данных, и распознавания образов. Поэтому отображение информации в удобном и понятном для человека виде – важная задача всех времен.

Древние греки еще в конце первого тысячелетия до н.э. были способны преподносить свои архитектурные идеи в графическом виде, хотя соответствующие математические методы появились только в эпоху Ренессанса. Сегодня подобного рода информация создается архитекторами, конструкторами и чертежниками с помощью компьютерных систем.

На протяжении многих столетий картографы и астрономы вычерчивали карты, чтобы представить информацию о расположении небесных тел. Нет смысла говорить о том, какое значение имеют такие карты сегодня не только для навигации на Земле и в Космосе, но и для решения повседневных задач человечества с помощью геоинформационных систем. Сейчас любую карту можно в считанные минуты получить и обработать с помощью Internet.

За последние 100 лет в статистике использовали самые различные технологии для представления в графическом виде первичных данных и результатов их статистической обработки. Сегодня и в этой области не обойтись без компьютеров, которые не только обрабатывают собранные данные, но и формируют соответствующие графики, используя самые разнообразные

способы их представления, в том числе и с применением цвета. Только такая форма представления позволяет человеку без труда интерпретировать информацию, содержащуюся в гигабайтах собранных первичных данных.

Множество важных и интересных проблем анализа данных ставит и медицина. Новые технологии визуализации состояния человеческого организма, такие как компьютерная томография, магниторезонансное обследование, ультразвуковое зондирование и позитронно-эмиссионная томография, позволяют получать трехмерную информацию, которая может быть в последствии обработана вычислительными методами. Сами первичные данные формируются специальной медицинской аппаратурой, а последующая компьютерная обработка позволяет специалистам достаточно просто интерпретировать полученные цветные изображения.

С появлением суперкомпьютеров стало возможным исследовать проблемы, ранее отнесенные к классу неразрешимых. В области визуализации результатов экспериментов и научных расчетов средства компьютерной графики являются мощным инструментом для правильной интерпретации огромных массивов первичных данных. Исследования в таких областях, как течение жидкостей, молекулярная биология и математика, не обходятся без преобразования первичных данных в зримые геометрические образы, что помогает лучше понять суть происходящих процессов.

1.2 Проектирование

Проектирование является одной из стадий создания изделий и сооружений в технике и строительстве. Согласно заданными спецификацией основными характеристиками разрабатываемого изделия, конструктор ищет решение, оптимальное с точки зрения затрат и технических параметров.

Достоинства парадигмы взаимодействия конструктора с изображением проектируемой конструкции на экране ЭЛТ впервые подметил Айвен Сазерленд (Ivan Satherland) еще лет сорок назад. Сегодня уже ни у кого не возникает сомнения прогрессивность применения средств графического взаимодействия конструктора и компьютера в системах автоматизации проектирования (САПР).

Графическая система САПР позволяет изображать объекты на разных стадиях проектирования, что существенно упрощает этот процесс. Такие системы применяются в самых разнообразных отраслях техники – от проектирования микросхем со сверхвысокой степенью интеграции (СБИС) до автомобилей, самолетов и космических аппаратов. Именно САПР позволили сегодня достичь таких высоких результатов в этих отраслях. Не менее широкое применение такого рода системы нашли и в других сферах деятельности человека.

1.3 Моделирование

Как только графические системы стали обладать достаточной производительностью для создания сложных динамических изображений, возникла идея применить их в качестве средства моделирования реальной обстановки (симулятора) на разного рода тренажерах. Первыми такие системы освоили авиаторы и использовались для обучения пилотов на земле. Это позволило значительно снизить стоимость обучения, гарантируя при этом его высокое качество и безопасность.

В телевидении, кинематографии и рекламном деле в последнее время также широко используются средства компьютерной графики, позволяющие создавать динамические изображения, практически неотличимые от снятых в реальных условиях. Подобного рода статические изображения заполнили страницы ряда периодических изданий.

Стоимость создания полнометражного кинофильма с помощью компьютерной графики на сегодняшний день высока и часто превышает стоимость съёмки такого же фильма с реальными актерами и живой природой, но при этом нельзя создавать такие сцены, которые можно построить с помощью систем компьютерной графики.

В последнее время появилась еще одна область применения средств компьютерной графики, которая получила наименование виртуальной реальности (VR – virtual reality). В VR-системах человек-наблюдатель пользуется специальным шлемом с парой миниатюрных дисплеев, на экранах которых формируется разное изображение для правого и левого глаза. В результате создается стереоэффект, подобный тому, который наблюдается человеком в реальной обстановке. Кроме того, положение и ориентация головы

наблюдателя постоянно анализируется и обрабатывается графической системой виртуальной реальности. Затем по этим данным изменяется изображение на экранах дисплеев. Можно добавить к такой системе средства влияния на виртуальную реальность, например перчатки с силомоментными датчиками, данные от которых также необходимо обрабатывать в реальном времени и учитывать при визуализации. Если добавить к такой системе и звуковое сопровождение, то получится иллюзия полного погружения в виртуальный мир, наблюдатель при этом будет чувствовать себя участником происходящего.

Это уже не просто развлечение, а инструмент для практического применения, например, с помощью такой системы хирург может отработать методику проведения операции, космонавт может подготовиться к выходу в открытый космос и проведению ремонтных работ.

Виртуальные миры, но пока другого рода, приникли и в Internet – появился язык VRML (Virtual Reality Modeling Language), который позволяет создавать в сети виртуальные динамические миры. Технология VRML предназначена для создания с помощью языка VRML в Internet (WWW) управляемых трехмерных пространств с гиперсвязями, называемых Moving Worlds (движущимися мирами). Основной задачей VRML является реалистичное отображение образов окружающего мира и расширение возможностей Internet в сфере человеческого общения.

1.4 Интерфейс пользователя

Человек не может взаимодействовать с вычислительной машиной на «её языке» машинных команд – необходим интерфейс пользователя. В различные времена использовались различные подходы для решения этой проблемы, но в последнее время парадигма графического интерфейса стала доминирующей в сфере взаимодействия пользователя с компьютером. Графический интерфейс предполагает использование различного рода окон, пиктограмм, меню, кнопок, устройств указания, таких как мышь для осуществления взаимодействия человека с машиной. Такой подход наиболее прост и понятен для человека, чем, например, интерфейс командной строки. Поэтому большинство современных операционных систем и прикладных программ используют графический интерфейс. При чем с точки зрения пользователя эти

интерфейсы различаются только деталями, например, графические интерфейсы систем X Window, Microsoft Windows и MacOS в достаточной степени похожи, что позволяет пользователю легко переходить от одной системы к другой.

Сейчас уже миллионы людей пользуются услугами сети Internet, доступ к которой немыслим без графических программ-броузеров, таких как Netscape или Internet Explorer, которые используют одни и те же средства графического интерфейса. Мы настолько к ним привыкли, что часто и не задумываемся, что эти средства также относятся к инструментам компьютерной графики.

Единый стиль графического интерфейса – это своего рода стандартизация процесса взаимодействия человека и вычислительной машины. Однако развитие компьютерной графики вносит коррективы в графический интерфейс операционных систем и прикладных программ. Появляются новые решения оформления дизайна той или иной системы. Часто при проектировании графического интерфейса участвуют психологи, определяя элементы интерфейса наиболее удобные для человека.

2 Графическая система

Система компьютерной графики является, прежде всего, вычислительной системой и, как таковая, включает все компоненты вычислительной системы общего назначения. К основным компонентам графической системы можно отнести:

- процессор;
- память;
- буфер кадра;
- устройства вывода;
- устройства ввода.

Эта модель имеет общий характер и отображает структуру и графической рабочей станции, и персонального компьютера, и графического терминала большой вычислительной системы, работающей в режиме разделения машинного времени, и интеллектуальной системы формирования изображений. Но не смотря на то, что все компоненты присутствуют и в стандартном компьютере, именно специализация каждого компонента в соответствии с требованиями задач компьютерной графики и делает систему графической.

В настоящее время практически все графические системы используют растровый принцип создания изображения. Суть его заключается в том, что изображение рассматривается как массив – растр – простейших элементов, или пикселей (pixels). Каждый пиксель имеет четко заданное положение на экране. Массив кодов, определяющих засветку пикселей на экране, хранится в отдельной области памяти, которая называется буфером кадра (frame buffer). В системах особо высокого качества для буфера кадра используются специальные типы микросхем – видеопамять с произвольным доступом (VRAM), которые позволяют быстро вывести содержимое буфера на экран, а в это время загрузить новую видеостраницу.

Глубина (depth) буфера кадра характеризует количество бит информации, определяющих засветку каждого отдельного пикселя. Современные полноцветные системы характеризуются глубиной 24 бит, они способны создавать по-настоящему фотореалистическое изображение.

В простых системах, как правило, используется единственный процессор, на который возлагается решение как «обычных» задач, так и задач компьютерной графики. Основные графические функции, в принципе, сводятся к преобразованию описания графического примитива (отрезка, многоугольника и т.д.), сформированного прикладной программой в коды засветки определенных пикселей в буфере кадра. Процесс преобразования описания графического примитива в коды засветки пикселей получил наименование растрового преобразования (rasterization) или сканирующего преобразования (scan conversion). В современных высокопроизводительных графических системах для такого преобразования используются специализированные процессоры, причем не один, а несколько, каждый из которых выполняет свой набор графических функций.

В большинстве графических систем в качестве хотя бы одного из возможных устройств ввода используется обычная алфавитно-цифровая клавиатура. Но более специфическими устройствами, предназначенными именно для ввода графической информации, являются мышь, джойстик и планшет. Каждое из этих устройств способно передавать в систему информацию о положении и формировать управляющие сигналы – поэтому их называют устройствами указания.

Доминирующее положение среди устройств вывода изображений занимают электронно-лучевые трубки (ЭЛТ). В цветных ЭЛТ экран покрывается точками трех разных типов люминофора – одни под воздействием электронного луча излучают красный свет, другие – зелёный, третьи – синий. Точки люминофоров разного цвета размещены триадами. Большинство цветных ЭЛТ имеют три электронные пушки (соответственно трем цветам точек люминофора). Между отклоняющей системой и экраном в такой цветной ЭЛТ располагается теневая маска – металлический экран с множеством отверстий соответственно триадам точек люминофора. Теневая маска обеспечивает попадание луча, испускаемого определенной пушкой, только на точки люминофора «своего» цвета в соответствующей триаде.

Несмотря на то, что ЭЛТ являются наиболее распространенным средством отображения изображений в системах компьютерной графики, в последнее время интенсивно разрабатываются приборы, основанные на других физических принципах. Но и в них используется тот же растровый способ создания изображения. Например, применяют жидкокристаллические дисплеи (LEDs – liquid-crystal displays). Такие приборы также требуют регенерации изображения, как и ЭЛТ.

Графическая система должна выполнять функции, специфицированные в графическом прикладном интерфейсе (см. раздел 4), и формировать изображение на устройстве вывода. В течение последних 40 лет было предложено много вариантов структурной организации средств, реализующих функции графической API. В первых графических системах для решения задач компьютерной графики использовались машины общего назначения со стандартной архитектурой, предложенной еще фон Нейманом. Затем стали применять дисплейные процессоры, которые освобождали основной процессор от решения задач компьютерной графики – от него требовалось только сформировать описание изображения, которое затем полностью обрабатывалось на дисплейном процессоре.

Особенностью компьютерной графии есть конвейерный принцип обработки информации, то есть задачи компьютерной графики, как правило, требуют единообразных действий для всех объектов сцены, а это быстрее всего можно реализовать с помощью

специализированного графического процессора с конвейерной архитектурой. В конвейер можно включить такие этапы обработки (см. раздел 4): геометрические преобразования, отсечение, проективное преобразование, растровое преобразование.

3 Принципы создания трехмерного изображения

3.1 Основные понятия компьютерной графики

Изображение, формируемое системой трехмерной компьютерной графики, по своей природе является искусственно синтезируемым, поскольку оно не существует физически. Нам необходимо получить реалистичное изображение, поэтому при синтезе используются те же физические законы, которые действуют в отношении реальных объектов, воспринимаемых зрением человека.

В любом процессе формирования изображения присутствуют две сущности: объект и наблюдатель. Объект существует в пространстве независимо от процесса создания изображения и соответственно от наблюдателя. Синтезируемый графический объект формируется через спецификацию положения в пространстве разнообразных графических примитивов – точек, отрезков прямых или многоугольников. В большинстве графических систем для описания или аппроксимации объектов оказывается достаточно множества описаний точек в пространстве или вершин (vertices). Например, отрезок прямой характеризуется двумя вершинами, многоугольник – упорядоченным списком вершин, а сфера – двумя вершинами, одна из которых соответствует центру, а другая – любой точке на поверхности сферы. Одна из главных функций САПР – обеспечить пользователю возможность формировать такую синтетическую модель проектируемого изделия в окружающей это изделие среде.

Роль наблюдателя может выполнять камера, которая изображение объектов. Если наблюдателем является человек, то изображение формируется на сетчатке глаза, а в фотокамере – на поверхности фотопленки. Наблюдатель и наблюдаемый объект находятся в одном и том же трехмерном пространстве, а созданное изображение на экранной плоскости является двухмерным. Суть процесса формирования изображения и состоит в том, чтобы, зная положение (и характеристики) наблюдателя и объекта, описать получаемое при этом двухмерное изображение.

В упрощенном описании процесса формирования изображения можно опустить многие важные детали, однако, для построения реалистичных изображений их нужно учитывать. Совершенно очевидно, что без источника света объект просто-напросто погрузиться во «тьму», и ни о каком его изображении просто не может быть и речи (при этом сам объект будет существовать). Также на объект влияет цвет и текстура его поверхности.

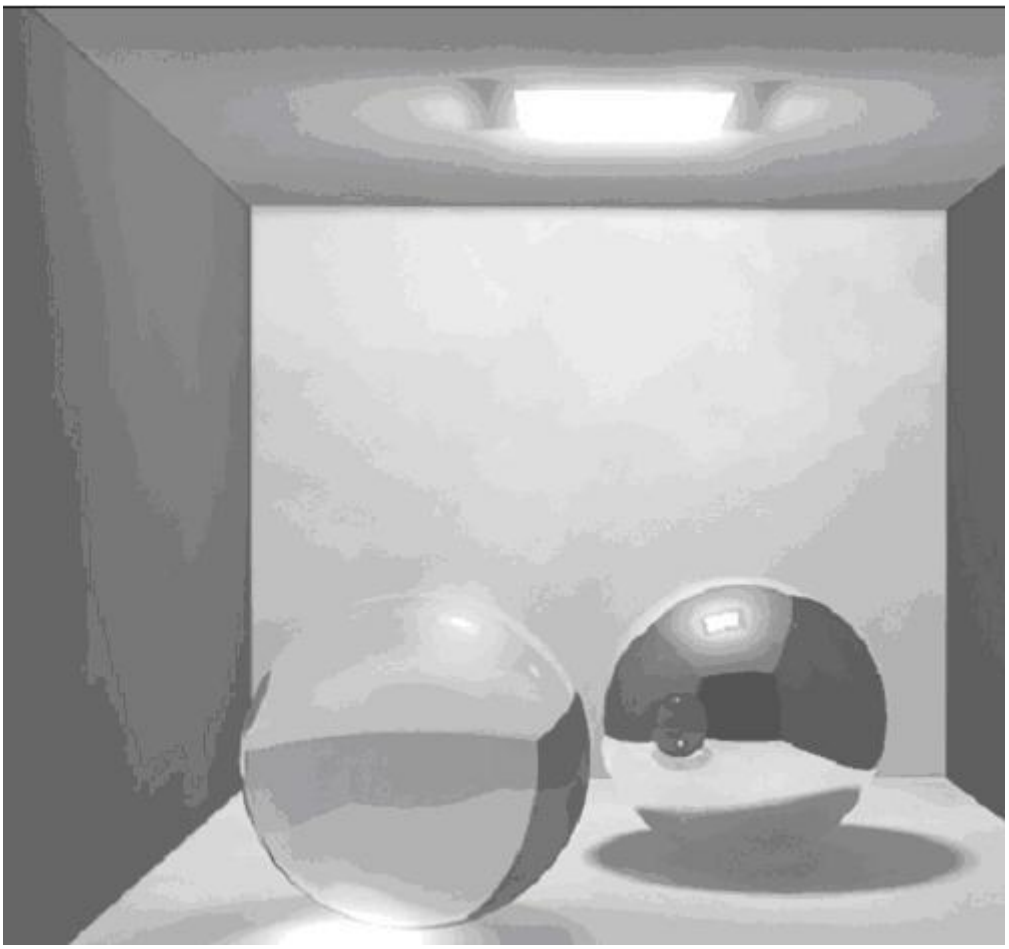


Рисунок 3.1 – Синтезированное реалистичное изображение

Свет, излучаемый источником, по-разному взаимодействует с различными участками поверхности объекта, и часть отраженной световой энергии попадает в объектив фотокамеры. Количество энергии попавшей в фотокамеру зависит от характера взаимодействия света с поверхностью объекта.

Поскольку свет распространяется по прямой линии, то его можно анализировать в терминах геометрических лучей, исходящих из источника света во все направления. Большая часть испускаемых источником лучей уходит в бесконечность, не

попадая прямо ни в камеру, ни на один из объектов сцены. Эти лучи не вносят никакого вклада в создание изображения, хотя их и могут видеть другие наблюдатели. Оставшиеся лучи попадают на объекты сцены и освещают их (см. рис. 3.1). Лучи, падающие на объект, могут взаимодействовать с ним по-разному. Например, если поверхность объекта зеркальна, то отраженные лучи могут (в зависимости от ориентации поверхности) достичь объектива и внести определенный вклад в формируемое изображение. Другие поверхности, которые называются рассеивающими, или диффузными, рассеивают падающий на них свет в различных направлениях. Если поверхность прозрачная, то световой луч от источника может пройти через нее, но при этом возникает его преломление и частичное отражение. Все эти преломленные и отраженные лучи могут затем взаимодействовать с другими объектами и, возможно, попасть в объектив камеры.

Описанный метод называют трассировкой лучей, он моделирует процесс формирования изображения на основе анализа заданных оптических явлений. Этот метод в последнее время все шире применяется в компьютерной графике. Его можно применять для моделирования оптических эффектов в сколь угодно сложной среде – все зависит только от производительности программы и компьютера, так как соответствующие алгоритмы требуют большого объема вычислений.

Рассматриваемые модели оптических систем формирования изображения непосредственно подводят нас к фундаментальной концепции современной трехмерной компьютерной графики, которая получила в литературе название модель синтезированной камеры (synthetic-camera model).

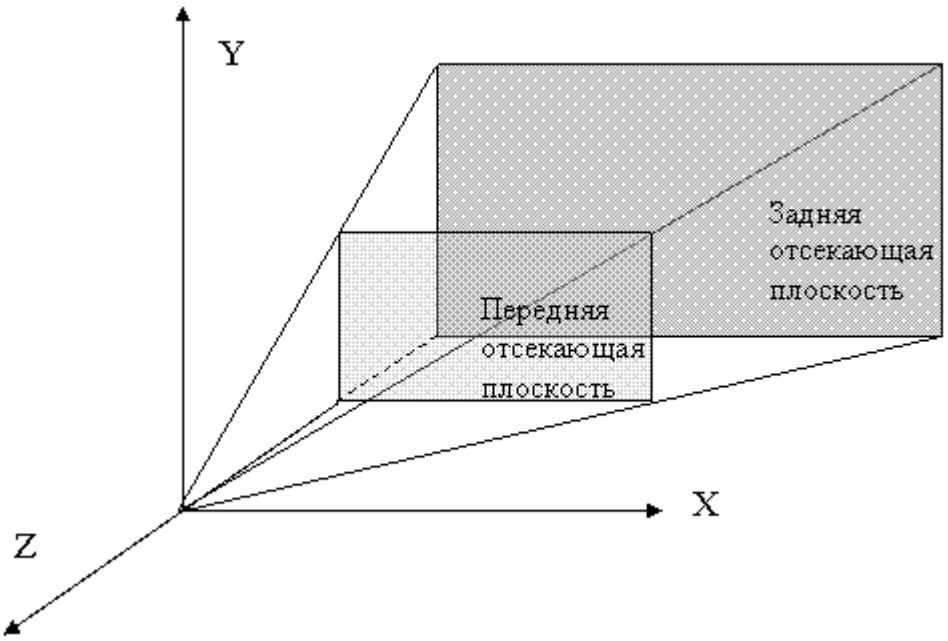


Рисунок 3.2 – Пирамида видимости

На рисунке 3.2 изображена пирамида видимости при центральном проецировании. Пирамида видимости имеет две отсекающие плоскости – переднюю и заднюю, которые удаляют части изображения объектов, находящихся от наблюдателя ближе, чем передняя плоскость и дальше, чем задняя.

При параллельном проецировании пирамида превращается в параллелепипед, так как центр проекций уходит в бесконечность.

3.2 Геометрические объекты

Компьютерная графика имеет дело с множеством геометрических объектов, таких как точки, отрезки, многоугольники и многогранники. Все разнообразие геометрических объектов можно с определенной степенью точности свести к ограниченному множеству простейших сущностей. Нам понадобится для этого три базовых типа – скаляры, точки и векторы. Существует несколько вариантов определения этих сущностей, в зависимости от того, с какой точки зрения их рассматривать – чисто математической (формальной), геометрической или с точки зрения программной реализации.

Точка играет роль фундаментального графического объекта в компьютерной графике. Она задает положение в пространстве и,

как правило, не имеет даже размеров. Точки существуют в пространстве не зависимо от каких-либо систем координат.

Скаляры не имеют геометрических свойств и чаще всего используются в качестве единиц измерения, так как скаляр – это действительное число. Например, длина отрезка или угол вращения объекта есть скаляры.

В компьютерной графике точки часто связываются с направленными отрезками и образуют при этом векторы. Физики используют термин «вектор» для обозначения величины, характеризуемой направлением и значением, например, скорость и сила являются векторами. Однако вектор в этом смысле может не иметь точки приложения.

Скаляры, точки и векторы можно рассматривать как элементы математических множеств. В математике принято использовать различные абстрактные пространства для решения различных прикладных проблем. Для компьютерной графики интерес представляют векторные, аффинные и Евклидовы пространства [1]. Векторное пространство содержит две сущности – векторы и скаляры. Аффинное пространство – это расширение векторного пространства, в которое включен дополнительный тип объектов – точка. В Евклидово пространство добавлена мера для измерения расстояния.

Математики предпочитают иметь дело со скалярами, точками и векторами как членами множеств, в информатике привычнее рассматривать их как абстрактные типы данных. Абстрактный тип данных – есть множество операций над некоторыми данными, причем операции определяются независимо от внутреннего представления данных в машине или от способа их реализации в конкретной системе. Таким образом, необходимо задать геометрические абстрактные типы данных, и с их помощью программист может на основании базовых объектов конструировать новые объекты.

В трехмерном пространстве значительно возрастает разнообразие геометрических объектов доступных компьютерной графике, так как появляются пространственные кривые, поверхности и совершенно новый класс объектов – объемные тела.

При создании системы трехмерной компьютерной графики, способной работать со всем разнообразием трехмерных объектов, программист сталкивается с множеством проблем, из которых наиболее существенны две:

1) пространственные объекты имеют сложное математическое описание;

2) из всего многообразия трехмерных объектов отбирают те, которые могут быть эффективно реализованы методами компьютерной графики: а те типы объектов, которые не попали в это множество, нужно аппроксимировать, то есть представить их приближенно объектами отобранных типов.

Как правило, в множество объектов, которые могут быть эффективно реализованы методами компьютерной графики, включаются объекты обладающие следующими тремя свойствами:

1. Объекты описываются поверхностями и могут рассматриваться как полые.

2. Количественно объекты полностью характеризуются множеством трехмерных вершин. Такие объекты наилучшим образом подходят для реализации в системах с конвейерной архитектурой (см. раздел 2).

3. Объекты состоят из плоских выпуклых многоугольников или могут быть аппроксимированы плоскими выпуклыми многоугольниками. В трехмерном пространстве многоугольник определяется упорядоченным множеством вершин. Если таких вершин более трех, то многоугольник не обязательно плоский, а значит значительно труднее определить внутреннюю область такого объекта. Поэтому в подавляющем большинстве графических систем на прикладную программу возлагается ответственность за то, что отображаемые многоугольники являются плоскими, иначе не гарантируется корректный результат растрового преобразования. Так как треугольник по своей природе всегда является плоским многоугольником, то все поверхности в графической системе желательно представлять множеством таких треугольников. Это может выполняться как на стадии моделирования объектов сцены, так и на стадии подготовки объектов к визуализации, когда графическая система сама производит разбиение плоских многоугольников на треугольники.

Исключением из данного подхода является конструктивная геометрия тел. В системах, использующих конструктивную геометрию тел, объект строится из небольшого множества монолитных (объемных) примитивов с помощью операций теории множеств, таких как объединение и пересечение.

В основном, современная компьютерная графика использует полигональные модели, описываемые множеством вершин и полигонов.

3.3 Геометрические преобразования

Геометрические преобразования – это преобразования геометрических объектов в пространстве. Для полигональных моделей геометрические преобразования сводятся к преобразующим действиям с множество точек и векторов, описывающих геометрический объект.

3.3.1 Аффинные преобразования на плоскости

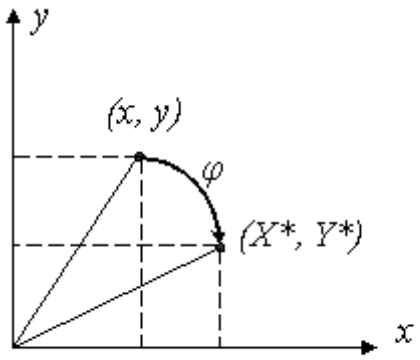


Рисунок 3.3 – Поворот точки вокруг начала координат

Кратко рассмотрим геометрические преобразования на плоскости. Существует три вида основных аффинных преобразований на плоскости.

Любое другое аффинное преобразование может быть представлено суперпозицией этих основных преобразований.

1. Поворот вокруг начала координат на угол φ (см. рисунок 3.3), описывается формулами

$$X^* = x\cos\varphi - y\sin\varphi$$
$$Y^* = x\sin\varphi + y\cos\varphi \qquad (3.1)$$

2. Растяжение (сжатие) вдоль координатных осей (относительно начала координат)

$$X^* = \delta x$$
$$Y^* = \lambda y \qquad (3.2)$$

где $\delta, \lambda > 0$ - коэффициенты растяжения (масштабирования)

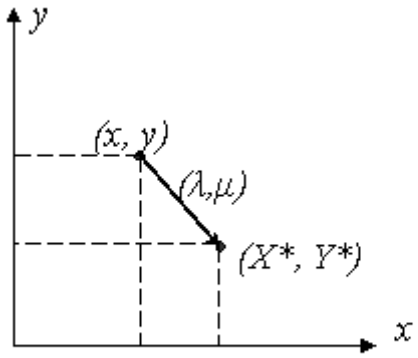


Рисунок 3.4 – Перенос точки

3. Перенос на вектор (λ, μ) изображен на рисунке 3.4.

$$X^* = x + \lambda$$
$$Y^* = y + \mu \qquad (3.4)$$

3.3.2 Однородные координаты

В трехмерном пространстве любой вектор $\vec{w}$ можно представить в виде линейной комбинации трех линейнонезависимых векторов $\vec{w} = a\vec{i} + b\vec{j} + c\vec{k}$, где скаляры a, b и c – компоненты (координаты) вектора $\vec{w}$ в базисе $\vec{i}, \vec{j}, \vec{k}$. Векторы, образующие базис, определяют конкретную систему координат. Для однозначного задания точки необходимо задать точку отсчета P_0 – начало координат. В совокупности вектора базиса и точка

отсчета называют фреймом. В конкретном фрейме любой вектор можно однозначно описать в виде

$$\vec{w} = a\vec{i} + b\vec{j} + c\vec{k}, \qquad (3.5)$$

Для любой точки P в том же фрейме можно записать соотношение

$$P = P_0 + x\vec{i} + y\vec{j} + z\vec{k}. \qquad (3.6)$$

Таким образом, любой вектор описывается во фрейме тремя скалярами (координатами), а описание точки включает три скаляра и данные о точке отсчета.

Пусть имеется два базиса: $\{\nu_1, \nu_2, \nu_3\}$ и $\{u_1, u_2, u_3\}$. Каждый вектор первого базиса можно представить во втором базисе и наоборот. Следовательно, существует девять компонент $\{\gamma_{ij}\}$ таких, что

$$u_1=\gamma_{11}\nu_1+\gamma_{12}\nu_2+\gamma_{13}\nu_3;$$

$$u_2=\gamma_{21}\nu_1+\gamma_{22}\nu_2+\gamma_{23}\nu_3; \qquad (3.7)$$

$$u_3=\gamma_{31}\nu_1+\gamma_{32}\nu_2+\gamma_{33}\nu_3.$$

Из этих компонент составим матрицу M

$$M = \begin{pmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{pmatrix}, \qquad (3.8)$$

тогда соотношение между компонентами примет вид

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = M \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}. \qquad (3.9)$$

Матрица M содержит информацию, необходимую для изменения представления вектора из одного базиса в другой.

Пусть имеется вектор $\vec{w}$, который в базисе $\{\nu_1, \nu_2, \nu_3\}$ имеет представление $\{\alpha_1, \alpha_2, \alpha_3\}$, то есть

$$\vec{w} = \vec{a}\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}, \qquad (3.10)$$

где $\vec{a}$=(α_1, α_2, α_3). Предположим, что вектор $\vec{b}$=(β_1, β_2, β_3) есть представление вектора $\vec{w}$ в базисе {u_1, u_2, u_3}. Таким образом, получили соотношение

$$\vec{w} = \vec{b}\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \vec{b} \cdot M \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \vec{a}\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}. \qquad (3.11)$$

Обратим внимание на то, что изменение базиса не затрагивает положение точки начала координат, поэтому плоскопараллельное смещение фрейма или перенос начала координат таким способом представить нельзя. Для решения этой проблемы вводят однороднее координаты (homogeneous coordinates), в которой для представления точек и векторов в трехмерном пространстве используется четырехмерный вектор. Любую точку P во фрейме, заданном набором параметров (v_1, v_2, v_3, P_0), можно однозначно представить соотношением

$$P = P_0 + \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3. \qquad (3.12)$$

Определим на множестве точек операцию умножения скаляров 0 и 1 на точку следующим образом:

$$0 \cdot P = 0,\ 1 \cdot P = P. \qquad (3.13)$$

Тогда выражение 3.12 можно записать в таком виде

$$P = \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 & 1 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{pmatrix}. \qquad (3.14)$$

Четырехмерная матрица-строка в правой части уравнения является однородными координатами точки P во фрейме, заданном параметрами (v_1, v_2, v_3, P_0). Вектор в этом же фрейме будет выглядеть следующим образом:

$$\vec{w} = \begin{pmatrix} \delta_1 & \delta_2 & \delta_3 & 0 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{pmatrix}. \qquad (3.15)$$

Итак, формальный аппарат однородных координат позволяет выполнять операции над точками и векторами с помощью обычных операций линейной алгебры. В этом случае проблема изменения фреймов (3.11) будет решаться просто.

Пусть имеется два фрейма: $\{v_1, v_2, v_3, P_0\}$ и $\{u_1, u_2, u_3, Q_0\}$. Тогда векторы базиса второго фрейма и его начало координат можно выразить так

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{pmatrix} = M \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{pmatrix}, \quad M = \begin{pmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} & 0 \\ \gamma_{21} & \gamma_{22} & \gamma_{23} & 0 \\ \gamma_{31} & \gamma_{32} & \gamma_{33} & 0 \\ \gamma_{41} & \gamma_{42} & \gamma_{43} & 1 \end{pmatrix}. \qquad (3.16)$$

Важным достоинством использования однородных координат в трехмерной компьютерной графике является то, что все аффинные преобразования (см. п. 3.3.1) при использовании представления в однородных координатах выполняются единообразно – с помощью перемножения матриц. Это даёт возможность просто организовать последовательное выполнение сложных преобразований в конвейере, а также реализовать большинство этих операций аппаратно, что значительно повысит скорость обработки.

3.3.3 Аффинные преобразования в пространстве

Поступая аналогично тому, как это было сделано в двумерном пространстве (см. п. 3.3.1) и применяя аппарат однородных координат опишем базовые аффинные преобразования, к которым можно свести любые другие более сложные преобразования, используемые в компьютерной графике.

1. Поворот вокруг осей координат фрейма. Существует три матрицы, задающие соответственно вращение вокруг оси абсцисс (3.17), ординат (3.18) и аппликат (3.19).

$$Rx = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3.17)$$

$$Ry = \begin{pmatrix} \cos\psi & 0 & \sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3.18)$$

$$Rz = \begin{pmatrix} \cos\chi & -\sin\chi & 0 & 0 \\ \sin\chi & \cos\chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3.19)$$

2. Матрица растяжения (сжатия) приводится в формуле 3.20, коэффициенты α, β, γ выражают растяжения вдоль осей абсцисс, ординат и аппликат соответственно.

$$D = \begin{pmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3.20)$$

3. Матрица переноса на вектор (λ, μ, ν) приводится в 3.21

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \lambda & \mu & \nu & 1 \end{pmatrix} \quad (3.21)$$

Приведенные матрицы позволяют выполнять соответствующие аффинное преобразование в однородных координатах в виде умножения матриц:

$$P_{new} = P_{old} \cdot Matrix, \qquad (3.22)$$

где P_{old} – однородные координаты до преобразования, P_{new} – однородные координаты после преобразования, Matrix – матрица преобразования.

3.4 Визуализация

3.4.1 Основные этапы визуализации

Визуализация заключается в отображении объектов сцены на экране. При этом необходимо использовать описание камеры или наблюдателя. Объекты задаются в мировом фрейме, а камера имеет свой собственный фрейм, который задает её положение и ориентацию. Поэтому для получения изображения на экранной плоскости сначала необходимо выполнить преобразование мировых координат объектов в координаты фрейма камеры, а затем выполнить проецирование на экранную плоскость и полученное изображение представить на растровом экране. На каждом этапе возникают различные трудности, которые необходимо преодолевать для получения реалистичного изображения.

Камера находится в сцене и может быть расположена так, что часть объектов будет не видна через эту камеру. Поэтому необходимо осуществлять отсечение таких объектов для того, чтобы не выполнять лишних вычислительных действий с ними. Как правило, для отсечения используют пирамиду (или параллелепипед) видимости (см. рисунок 3.2). Изображаться на экране будут только те объекты, которые попали внутрь этой пирамиды, а остальные будут либо отсечены полностью, либо частично, если часть объекта находится в пирамиде видимости – это отдельная проблема, которая решается с помощью различных алгоритмов трехмерного отсечения, например, алгоритма трехмерного отсечения Коэна-Сазерленда [1].

После выполнения преобразования координат во фрейм камеры и трехмерного отсечения необходимо спроецировать видимые объекты на экранную плоскость, при этом возникает проблема удаления невидимых частей объектов – необходимо определить какие объекты будут видны наблюдателю, а какие будут закрыты для наблюдателя другими объектами. Такую проблему решают с помощью привлечения алгоритмов удаления

невидимых частей [1,2,3], таких как алгоритм художника, алгоритм Варнака и др. Самым распространенным, пожалуй, является алгоритм Z-буфера. Он хорошо реализуется как программно, так и аппаратно, прекрасно сочетается с конвейерной архитектурой графической системы и может быть реализован в процессе растрового преобразования и закраски. Основная идея алгоритма: создается вспомогательный буфер глубины – это двумерный массив, в котором каждому пикселю экрана ставится в соответствие наименьшая глубина до объектов (граней), которые пересекаются с лучом, проведенным из центра проекции через пиксель экранной плоскости; видимым считается тот объект, расстояние до которого меньше.

3.4.2 Растровое преобразование

Растровое преобразование – это завершающий этап построения изображения, когда необходимо двумерное описание объекта изобразить на растровом экране, то есть нужно заполнить буфер кадра (см. раздел 2) соответствующими значениями кодов засветки пикселя.

Растровое преобразование отрезков можно реализовать по алгоритму Брезенхема (Bresenham) [1]. Этот алгоритм был предложен в 1965 году, и с тех пор используется практически во всех графических системах. Главное его достоинство заключается в том, что для растеризации отрезка нужно выполнять только целочисленные вычисления, что существенно ускоряет и упрощает процесс растеризации по сравнению с алгоритмами, использующими действительные числа. Поэтому алгоритм Брезенхема просто реализовать аппаратно.

Растровое преобразование многоугольников сопряжено с тонированием и закраской.

Рассмотрим один из вариантов растеризации треугольника. Возьмем любой треугольник ABC и растровый экран, как показано на рисунке 3.5. Изображение треугольника на экране - набор горизонтальных отрезков, причем из-за того, что треугольник - фигура выпуклая, каждой строке экрана соответствует не более одного отрезка. Поэтому достаточно пройтись по всем строкам экрана, с которыми пересекается треугольник, то есть, от минимального до максимального значения *y* (вертикальной

координаты) для вершин треугольника, и нарисовать соответствующие горизонтальные отрезки.

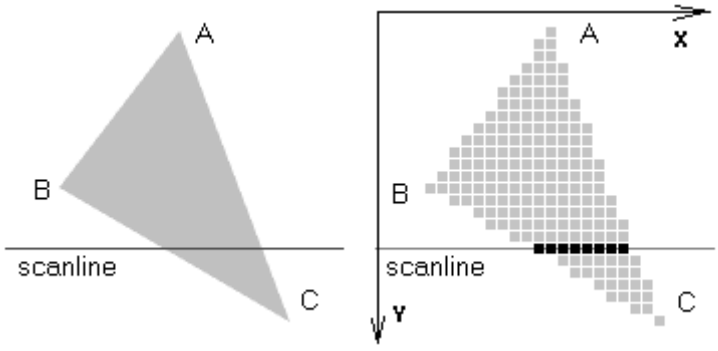


Рисунок 3.5 – Растеризация треугольника

Отсортируем вершины так, чтобы вершина A была верхней, C - нижней, тогда

$$min\_y = A.y, \qquad (3.23)$$

$$max\_y = C.y.$$

Необходимо пройтись по всем линиям от min_y до max_y. Рассмотрим какую-нибудь линию, соответствующую значению sy на экране,

$$A.y \le sy \le C.y\,. \qquad (3.24)$$

Если sy меньше B.y, то она пересекает стороны AB и AC; если sy больше или равно B.y, то стороны BC и AC.

Процедура растеризации будет выглядеть следующим образом. Зная координаты всех вершин, можем написать уравнения сторон и найти пересечение нужной стороны с прямой y = sy. Получим два конца отрезка, затем определим, какой из них левый, а какой правый путем сравнения их координат по горизонтали. Далее необходимо изобразить отрезок и повторить эту процедуру для каждой строки.

Остановимся более подробно на нахождении пересечения прямой y = sy (текущей строки) и стороны треугольника, например AB. Запишем уравнение прямой AB:

$$x = A.x + (y - A.y)\frac{B.x - A.x}{B.y - A.y} \qquad (3.25)$$

Подставляем сюда известное для текущей прямой значение y = sy:

$$x = A.x + (sy - A.y)\frac{B.x - A.x}{B.y - A.y} \qquad (3.26)$$

Для других сторон пересечение ищется аналогично. Пример реализации на языке Си.

```
// ...
// здесь сортируем вершины (A,B,C)
// ...
for (sy = A.y; sy <= C.y; sy++)
{
x1 = A.x + (sy - A.y) * (C.x - A.x) / (C.y - A.y);
if (sy < B.y)
 x2 = A.x + (sy - A.y) * (B.x - A.x) / (B.y - A.y);
 else {
 if (C.y == B.y)
 x2 = B.x;
 else
 x2 = B.x + (sy - B.y) * (C.x - B.x) / (C.y - B.y);
 }
 if (x1 > x2) { tmp = x1; x1 = x2; x2 = tmp; }
 drawHorizontalLine(sy, x1, x2);
}
```

// ...

3.4.3 Простая модель освещенности

Для получения реалистичного изображения необходимо не просто выполнять растеризацию с заполнением многоугольника цветом объекта, а учитывать освещенность сцены (см. п. 3.1).

При расчете освещенности для любой поверхности должны учитываться следующие факторы:

- интенсивность света источников;
- интенсивность света, отраженного от предметов окружающей обстановки;
- отражающие свойства освещаемой поверхности;
- взаимное расположение источников света, освещаемых поверхностей и наблюдателя.

Главное различие моделей освещенности заключается в учете отражающих свойств освещаемой поверхности, а именно зеркальной составляющей.

Все поверхности по характеру отражения падающего света делятся на три класса. К первому классу относят поверхности с идеальным диффузным отражением (ламбертовские поверхности). Отраженный свет для них распределяется равномерно во всех направлениях. К таким поверхностям приближены асфальт, песок и т.п. Ко второму классу принадлежат поверхности с идеальным зеркальным отражением. Падающий луч, отраженный луч и нормаль к такой поверхности в точке падения лежат в одной плоскости, и угол падения равен углу отражения. К поверхностям второго класса приближены вода, стекло, полированные металлы. Третий класс образуют поверхности с антизеркальным отражением, для которых реально существует телесный угол, в рамках которого распространяется отраженная энергия. К поверхностям третьего класса можно отнести поверхности, покрытые растительностью. Практически для любой поверхности существуют все три составляющие.

Рассмотрим простую модель освещенности [3]. Световая энергия, падающая на поверхность, может быть частично

поглощена, частично отражена и частично пропущена (преломлена). Объект можно увидеть, только если он отражает или пропускает (преломляет) свет. Количество поглощенной, отраженной и пропущенной световой энергии зависит от свойств материала объекта и длины световой волны. При освещении белым цветом, если поглощаются лишь определенные длины волн, то у света, исходящего от объекта, изменяется распределение энергии по длинам волн и объект приобретает определенный цвет.

Свойства отраженного света зависят от строения, направления и формы источника света, от ориентации и свойств освещаемой поверхности.

Свет точечного источника света отражается от идеальной зеркальной поверхности по закону косинусов Ламберта: интенсивность отраженного света пропорциональна косинусу угла между направлением света и нормалью к поверхности, то есть

$$I = I_l k_d \cos\theta, \quad 0 \le \theta \le \frac{\pi}{2}, \qquad (3.27)$$

где I – интенсивность отраженного света, I_l – интенсивность точечного источника, k_d – коэффициент диффузного отражения $(0 \le k_d \le 1)$, θ – угол между направлением света и нормалью к поверхности. Коэффициент k_d зависит от материала и от длины световой волны, в простых моделях его можно считать постоянным.

Формула 3.27 не учитывает рассеянного света. Объекты реального мира подвержены рассеянному свету, отраженному от объектов окружающей обстановки. Поскольку для расчета таких источников нужны большие вычислительные мощности, в компьютерной графике они заменяются на коэффициент рассеяния – константу, которая добавляется к формуле 3.27:

$$I = I_a k_a + I_l k_d \cos\theta, \quad 0 \le \theta \le \frac{\pi}{2}, \qquad (3.28)$$

где I_a – интенсивность рассеянного света, k_a – коэффициент диффузного отражения рассеянного света $(0 \le k_a \le 1)$.

Для учета влияния расстояния на интенсивность освещенности рассматривают модель

$$I = I_a k_a + \frac{I_l k_d \cos\theta}{K + d}, \qquad (3.29)$$

где d – расстояние от объекта до наблюдателя, К – произвольная постоянная.

Интенсивность зеркальной составляющей отраженного света зависит от угла падения, длины волны падающего света и свойств материала. Так как физические свойства зеркального света очень сложны, то в простых моделях используют эмпирическую модель Буи-Туонга Фонга зеркальной составляющей

$$I_s = I_l w(i, \lambda) \cos^n \alpha, \qquad (3.30)$$

где I_s – интенсивность зеркальной составляющей, *w(i, λ)* – кривая отражения, представляющая отношение зеркально отраженного света к падающему свету как функцию угла падения *i* и длины волны λ, n – степень, аппроксимирующая пространственное распределение зеркально отраженного света, α – угол между отраженным лучом и вектором наблюдателя. Объединяя эту модель с формулой 3.29, получаем модель освещенности

$$I = I_a k_a + \frac{I_l}{K + d}(k_d \cos\theta + w(i, \lambda) \cos^n \alpha). \qquad (3.31)$$

Функция *w(i, λ)* довольно сложна, поэтому её обычно заменяют константой k_s, которая подбирается экспериментально. Таким образом, получили такую модель освещенности

$$I = I_a k_a + \frac{I_l}{K + d}(k_d \cos\theta + k_s \cos^n \alpha). \qquad (3.32)$$

Если в сцене присутствует m источников света, то их эффекты суммируются

$$I = I_a k_a + \sum_{j=1}^{m} \frac{I_l}{K + d}(k_d \cos\theta_j + k_s \cos^n_j \alpha_j). \qquad (3.33)$$

Для цветного изображения необходимо получить интенсивности освещенности I по формуле 3.33 для каждого из трех основных цветов.

Формула 3.33 используется для расчета интенсивностей точек объекта при растеризации, когда необходимо определить, каким цветом засвечивать пиксель на экране. Однако выполнять подобный расчет для каждого пикселя во время растеризации – сложный процесс, поэтому применяют различные алгоритмы закраски полигонов.

3.4.4 Методы закраски

Закраска многоугольника (треугольника) производится согласно принятой модели освещенности, но так как существуют объекты, построенные аппроксимацией кривой поверхности треугольниками, то необходимо выполнять сглаживание закраски на линии соединения таких многоугольников. Это необходимо делать потому, что каждый конкретный треугольник имеет свою собственную нормаль, и в результате получаются различные закраски у соседних треугольников.

Поскольку вершина является точкой пересечения, как минимум, двух по-разному ориентированных многоугольников, то в ней происходит разрыв непрерывности функции вектора нормали, поэтому Гуро (Gouraud) предложил нормаль в точке вершины вычислять путем усреднения нормалей многоугольников, пересекающихся в этой вершине. Метод закраски по Гуро тогда сводится к следующему: необходимо определить интенсивность освещенности в вершинах многоугольника, используя при этом усреднённый вектор нормали, а затем с помощью билинейной интерполяции вычислять интенсивности каждого пикселя на сканирующей строке во время растеризации (см. п. 3.4.2).

Фонг предложил интерполировать не освещенность точек от вершины к вершине, а вектор нормалей. Затем его использовать для вычисления интенсивности освещенности. При таком подходе получается лучшая аппроксимация кривизны поверхности и, следовательно, получается более реалистичное изображение, более правдоподобными выглядят зеркальные блики. Однако объем вычислений при этом резко возрастает, так как необходимо

выполнять расчет интенсивности освещения для каждого пикселя во время растеризации.

Существует множество аппаратных реализаций закраски методом Гуро, которые позволяют получать изображения приемлемого качества, практически не увеличивая время закрашивания. Этого нельзя сказать о методе Фонга. В результате в настоящее время метод Фонга используется только в тех системах, где не требуется формировать изображение в реальном масштабе времени.

4 Интерфейс прикладного программирования

4.1 Понятие интерфейса прикладного программирования

Интерфейс между прикладной программой и графической системой (см. раздел 2) – это множество функций, которые в совокупности образуют графическую библиотеку. Спецификация этих функций и есть то, что называют интерфейсом прикладного программирования (API – application programmer's interface).

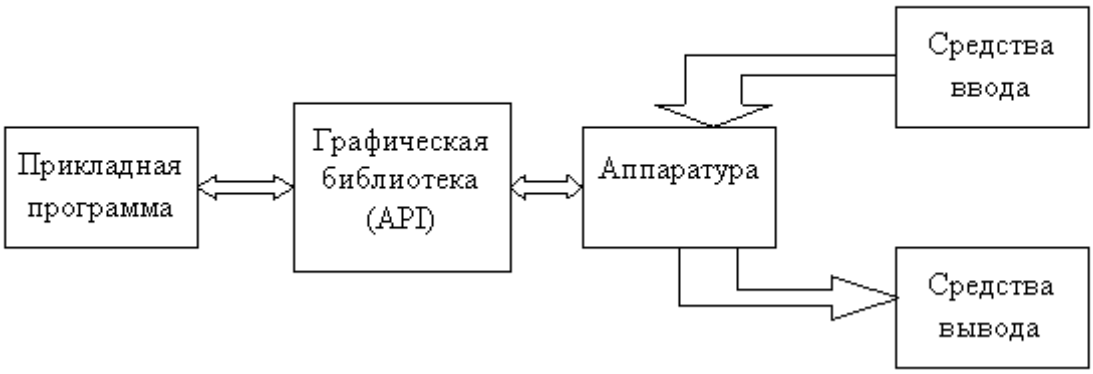


Рисунок 4.1 – Структура прикладной графической системы

Модель системы прикладного программирования показана схематически на рисунке 4.1. Для программиста, занимающегося разработкой прикладной программы, существует только API, и он избавлен, таким образом, от необходимости вникать в подробности работы аппаратуры и программной реализации функций графический библиотеки. С точки зрения прикладного программиста те функции, к которым он обращается, должны соответствовать концептуальной модели описания изображения. Основой для этого часто является описанная выше (см. раздел 3)

модель синтезированной камеры. Она используется во множестве разработанных API, таких как OpenGL, PHIGS, Direct3D, JAVA-3D.

Следуя концепции модели синтезированной камеры, в составе API должны присутствовать функции, которые позволяли бы описывать:

- объекты;
- наблюдателя;
- источники света;
- свойства материалов объектов.

Для описания объектов чаще всего используются массивы вершин. Для простых геометрических объектов – отрезков прямых, прямоугольников и многоугольников – существует достаточно очевидное соответствие между списком вершин и формой объектов, основанное на простых математических соотношениях. Более сложные объекты могут быть по-разному определены с помощью списка вершин. Например, круг можно определить тремя точками на окружности или центром и одной точкой на окружности.

В большинстве API в распоряжение пользователя представляется практически один и тот же набор примитивов. Такие примитивы довольно быстро отображаются аппаратными средствами. Типовой набор включает точки, отрезки, многоугольники и, иногда, текст. Иногда в число примитивов включаются части кривых и участки поверхностей, но часто такие объекты приходится аппроксимировать более простыми примитивами, причем эта задача возлагается на саму прикладную программу.

Некоторые API позволяют пользователю работать напрямую с буфером кадра (см. раздел 2) – считывать и записывать коды засветки отдельных пикселей.

Описать наблюдателя или камеру можно разными способами. Доступные на сегодняшний день графические API отличаются как гибкостью, которую они обеспечивают при выборе параметров камеры, так и количеством имеющихся методов ее описания. Для камеры, например, существуют такие 4 типа параметров, однозначно определяющих характеристики создаваемого ею изображения.

1. Положение камеры задается положением центра проекции.

2. Ориентация. Расположив центр проекции в определенной точке пространства, можно совместить с ним начало локальной системы координат камеры и вращать ее относительно осей этой системы координат, изменяя, таким образом, ориентацию объектива.

3. Фокусное расстояние объектива камеры фактически определяет размер изображения на плоскости проекции.

4. Экранная плоскость. Некоторые API позволяют задавать ориентацию экранной плоскости независимо от ориентации объектива.

Такую спецификацию можно сформировать разными способами. Один из них состоит в том, что положение и ориентация камеры задается набором координатных преобразований (см. п. 3.4.1). Эти координатные преобразования используются для вычисления положения точек объектов, заданных вершинами в мировом фрейме, в системе координат камеры. Такой подход очень удобен и с точки зрения его программной реализации, и с точки зрения гибкости графической системы, поскольку позволяет довольно просто менять виды объекта при перемещении камеры.

Необходимость настройки значений множества параметров также представляет определенную сложность. Частично причины кроются в самой модели синтезированной камеры. В классических методах визуализации, применяемых, в частности, в архитектурном проектировании, акцент делается именно на взаимосвязи наблюдателя и среды, в то время как модель предполагает независимость наблюдателя от наблюдаемых объектов. Ни одна из существующих графических библиотек, построенных на основе синтезированной камеры, не имеет в своем составе функций для задания взаимосвязей между камерой и объектом.

Источник света характеризуется положением, интенсивностью, цветом излучения и его направленностью. В составе большинства API имеются функции для задания такого рода параметров, причем в сцене может присутствовать несколько

источников света с различными характеристиками. Существуют и функции спецификации оптических свойств материалов поверхностей объектов.

4.2 Обзор библиотеки OpenGL

Интерфейс OpenGL [4] является на данный момент одним из самых популярных программных интерфейсов (API) для разработки приложений в области двумерной и трехмерной графики. Стандарт OpenGL был разработан и утвержден в 1992 году девятью ведущими фирмами: Digital Equipment Corporation, Evans & Sutherland, Hewlett-Packard Co., IBM Corp., Intel Corp., Intergraph Corp., Silicon Graphics, Inc., Sun Microsystems и Microsoft Corporation. В основу стандарта была положена библиотека IRIS GL, разработанная Silicon Graphics. Библиотека OpenGL завоевала огромную популярность и была интегрирована со множеством языков и систем разработки приложений (C, C++, Java).

OpenGL – это графический стандарт, который предоставляет широкие возможности несмотря на то, что поддерживает простейшую модель программирования. Её процедурный интерфейс позволяет программистам легко и эффективно описывать как простые, так и комплексные задачи воспроизведения изображения. Более того, OpenGL спроектирована таким образом, чтобы использовать все преимущества любых, даже самых изощренных графических подсистем.

OpenGL – открытая графическая библиотека. На неё имеется спецификация, где все четко документировано и описано, поэтому библиотеку OpenGL может производить любой программист. Главное, чтобы она удовлетворяла спецификации OpenGL и ряду тестов. Как следствие, в библиотеке нет никаких темных мест, секретов, недокументированных возможностей и т.п.

Библиотеку OpenGL выпускают такие корпорации как Microsoft, Silicon Graphics, а также просто группы программистов. Одним из таких примеров служит реализация Mesa, которая распространяется в исходных текстах на языке Си и собирается почти для любой операционной системы. Эту библиотеку написал целый ряд программистов, главным автором является Brian Paul.

Характерными особенностями OpenGL, которые обеспечили распространение и развитие этого графического стандарта, являются:

1. Стабильность. Дополнения и изменения в стандарте реализуются таким образом, чтобы сохранить совместимость с разработанным ранее программным обеспечением.
2. Надежность и переносимость. Приложения, использующие OpenGL, гарантируют одинаковый визуальный результат вне зависимости от типа используемой операционной системы и организации отображения информации. Кроме того, эти приложения могут выполняться как на персональных компьютерах, так и на рабочих станциях и суперкомпьютерах.
3. Легкость применения. Стандарт OpenGL имеет продуманную структуру и интуитивно понятный интерфейс, что позволяет с меньшими затратами создавать эффективные приложения, содержащие меньше строк кода, чем с использованием других графических библиотек. Необходимые функции для обеспечения совместимости с различным оборудованием реализованы на уровне библиотеки и значительно упрощают разработку приложений.

Основные возможности, которые OpenGL предоставляет разработчикам:

1. геометрические примитивы (точки, линии, многоугольники);
2. растровые примитивы (битовые массивы, прямоугольники пикселей);
3. работа с цветом в RGBA и в индексном режимах;
4. видовые и модельные преобразования;
5. удаление невидимых частей изображения;
6. прозрачность;
7. использование B-сплайнов для рисования линий и поверхностей;
8. наложение текстуры;
9. применение освещения;
10. использование плавного сопряжения цветов;
11. устранение ступенчатости;
12. использование «тумана» и других «атмосферных» эффектов;
13. использование списков изображений.

Как уже было сказано, существует реализация OpenGL для разных платформ, для чего было удобно разделить базовые функции графической системы и функции для отображения графической информации и взаимодействия с пользователем. Были созданы библиотеки для отображения информации с помощью оконной подсистемы для операционных систем Windows и Unix (WGL и GLX соответственно), а также библиотеки GLAUX и GLUT, которые используются для создания консольных приложений.

Библиотеки GLAUX и GLUT отвечают за взаимодействие с системой окон и обеспечивают взаимодействие с пользователем. Библиотека (GLUT - GL Utility Toolkit) [5] была разработана для X Window [6, 7], но ее новые версии применимы и в операционных системах Microsoft Windows 98/NT/2000/XP. GALUX является более простым инструментальный пакетом, он используеется, в частности, компанией Microsoft для ОС Windows. Возможно конвертирование программ, написанных с ориентацией на GLAUX в программы, работающие с библиотекой GLUT.

В состав библиотеки (GLU – graphics utility library) 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. Все они реализованы через базовые функции OpenGL.

С точки зрения архитектуры, графическая система OpenGL является конвейером, состоящим из нескольких этапов обработки данных:

1. аппроксимация кривых и поверхностей;
2. обработка вершин и сборка примитивов;
3. растеризация и обработка фрагментов;
4. операции над пикселами;
5. подготовка текстуры;
6. передача данных в буфер кадра.

Вообще, OpenGL можно сравнить с конечным автоматом, состояние которого определяется множеством значений специальных переменных (их имена обычно начинаются с символов GL_) и значениями текущей нормали, цвета и координат текстуры. Вся эта информация будет использована при

поступлении в систему координат вершины для построения фигуры, в которую она входит. Смена состояний происходит с помощью команд, которые оформляются как вызовы функций.

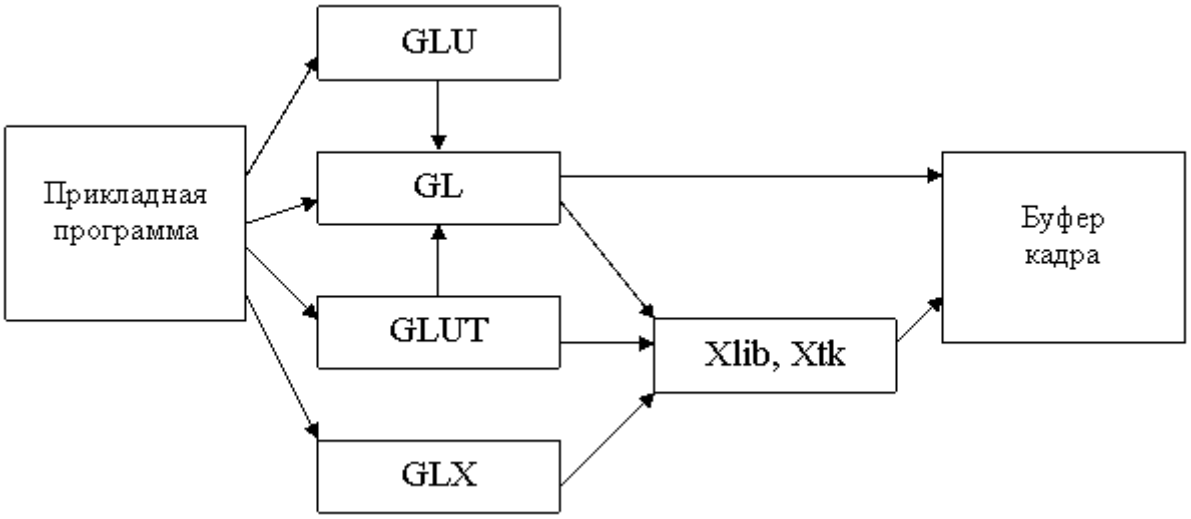


Рисунок 4.2 – Организация библиотеки OpenGL в X Window

Имена функция OpenGL начинаются с букв gl, а сами функции хранятся в основной библиотеке, которая на рисунке 4.2 обозначена аббревиатурой GL.

На рисунке 4.2 схематически представлена организация системы библиотек в той версии OpenGL, которая работает под управлением X Window. Аналогичная организация используется и в версиях OpenGL, работающих под управлением других операционных систем, в частности Microsoft Windows.

4.3 Обзор библиотеки DirectX и Direct3D

Библиотека DirectX обладает широкими возможностями, предоставляемыми программисту графических и игровых приложений для Windows. Первоначально эта библиотека предназначалась для разработки игр, однако сейчас она стала настолько популярной среди разработчиков для Windows, что ее стали использовать и для других целей - системы дистанционного обучения, САПР и др. DirectX включает в себя несколько отдельных библиотек, каждая из которых предназначена для решения отдельных задач:

- DirectDraw - организует взаимодействие приложений с графическим устройством, решает проблемы двумерной графики и одновременно является основой для Direct3D;
- Direct3D - предназначена для построения приложений Windows, использующих трехмерную графику, Direct3D обладает также существенными возможностями: перекрываемая буферизация глубины (Z-буфер), сплошная закраска и закраска Гуро, множественные источники света и различные типы освещения, поддержка материалов и текстур, выполнение преобразований и отсечений и др.;
- DirectSound, DirectMusic позволяют работать со звуком, в том числе с трехмерным окружением;
- DirectPlay используется для программирования сетевых игр;
- DirectInput - доступ к устройствам ввода, таким как клавиатура, джойстик и др.

Следует также указать, что DirectX использует аппаратное ускорение везде, где это возможно, при этом независимо от аппаратных средств (возможностей того или иного видеоадаптера или звуковой платы), если аппаратура не в состоянии выполнить возложенные на неё расчеты, то DirectX эмулирует их программно.

Библиотека Direct3D поддерживает два режима работы (см. рис. 4.3): Immediate Mode (непосредственный режим) - это низкоуровневый программный интерфейс 3D для создания эффективных мультимедиа приложений для Windows;

Retained Mode (абстрактный режим) - это высокоуровневый программный интерфейс для быстрой, легкой разработки 3D графики в Windows.

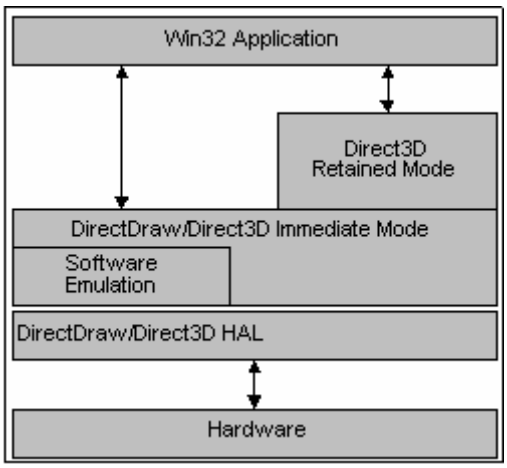


Рисунок 4.3 - Архитектура Direct3D

Так как объекты DirectX взаимодействуют со спецификацией COM (Component Object Model), а обращение к COM объектам происходит только через интерфейсы, то несложно предположить, что имеется определенный набор интерфейсов Direct3D, обеспечивающих совместимость с COM. Интерфейс IDirect3D7 является корневым интерфейсом для доступа к другим интерфейсам DirectX.

Объект DirectDraw обеспечивает функциональность Direct3D, интерфейс для него IDirect3D7. Объект DirectDrawSurface создает поверхности с трехмерными свойствами, этот объект содержит битовые карты текстур, интерфейс для него IDirectDrawSurface7. Объект Direct3DDevice хранит состояние приложения и определяет какими возможностями обладает компьютер, интерфейс для него IDirect3DDevice7.

Устройства Direct3D инкапсулируют и хранят состояния визуализации (рендеринг), они выполняют преобразования и операции освещения, поддерживаются интерфейсами IDirect3D7 и IDirect3DDevice7. В Direct3D существует 4 типа устройств, приложение выбирает то устройство, которое более всего подходит к имеющемуся аппаратному обеспечению (устройство можно назначить и принудительно):

- Устройство HAL (Hardware Abstract Layer). Приложение даже в случае наличия аппаратного ускорения не получает прямого доступа к видеоплате, а вызывает функции Direct3D. При использовании этого устройства достигается максимальное быстродействие.
- Устройство MMX. MMX - это набор специальных инструкций микропроцессора, предназначенных для ускорения мультимедиа операций. Устройство MMX не является устройством с аппаратным ускорителем, все операции полностью моделируются программно, однако, это устройство самое быстрое среди всех моделируемых устройств.
- Устройство RGB полностью программно моделирует работу с цветами, работает медленнее MMX.

– Устройство Ramp программно эмулирует работу с монохромным освещением, не поддерживается в DirectX 6 и выше.

Для создания устройства необходимо воспользоваться методом IDirect3D7::CreateDevice().

Direct3D позволяет задавать свойства материалов, источники света и налагать текстуры на поверхности.

4.4 Некоторые высокоуровневые надстройки OpenGL

Анонсировав в 1992 году базовый стандарт OpenGL, SGI продолжала развитие библиотеки путем создания высокоуровневых надстроек, основной задачей которых было предоставление разработчику средств более быстрого и надежного создания приложений. Со временем они завоевали заслуженное место в корпоративном секторе. Рассмотрим некоторые из таких надстроек

4.4.1 GLUI

GLUI [8] – это свободно распространяемая библиотека, предназначенная для построения привычного графического интерфейса с пользователем для консольных приложений на основе библиотеки GLUT. Автором GLUI является Паул Радемахер (Paul Rademacher).

GLUI обеспечивает функционирование таких элементов управления как кнопки, переключатели, радиокнопки, окна списка и т.д. (см. рисунок 4.4) в приложениях OpenGL независимо от системы окном. Она обрабатывает все зависимые от системы проблемы.

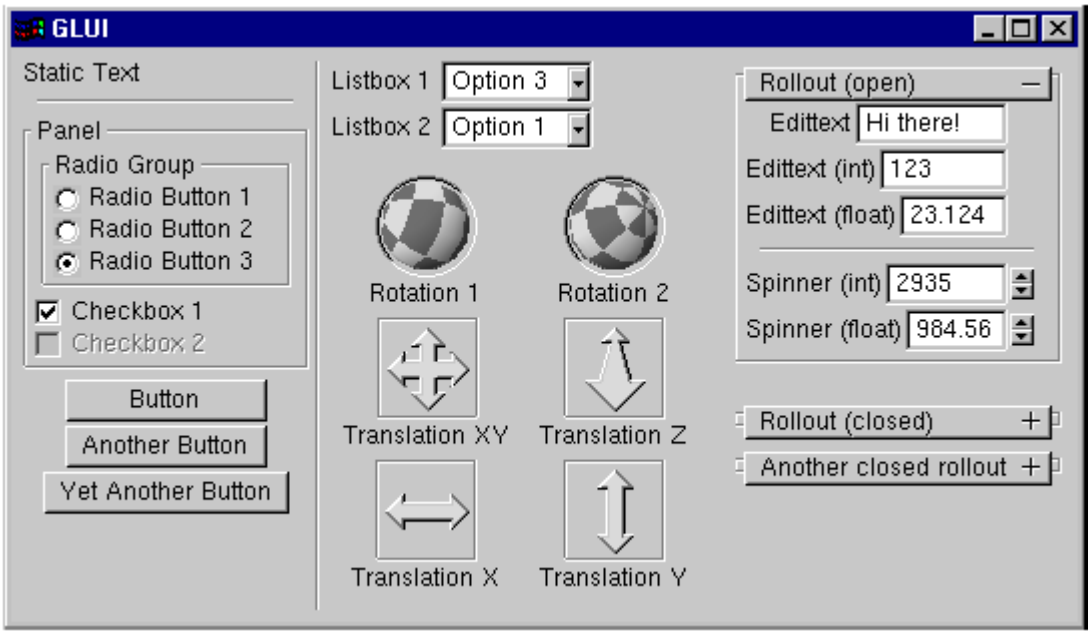


Рисунок 4.4 – Различные элементы управления библиотеки GLUI

Библиотека GLUI обладает такими особенностями:

- полная интеграция с инструментарием GLUT;
- простое создание нового окна интерфейса пользователя, практически одной командой;
- поддержка многооконности;
- поддержка стандартных элементов управления таких как:
  - кнопки;
  - переключатели для булевых переменных;
  - радиокнопки для выбора взаимоисключающих вариантов;
  - текстовые поля для того, чтобы вводить текст и числа;
  - списки;
  - поля статического текста;
  - панели для группировки наборов элементов управления;
  - другие;
- элементы управления генерируют функции обратного вызова при изменении значений;
- переменные прикладной программы могут быть связаны с элементами управления и автоматически обновляться, когда изменяется значение этого элемента управления;

- автоматическое вычисление размеров элементов управления и т.п.

4.4.2 The Fast Light Tool Kit (FLTK)

The Fast Light Tool Kit (FLTK) [9] – это библиотека для создания графического пользовательского интерфейса в кроссплатформенных проектах OpenGL.

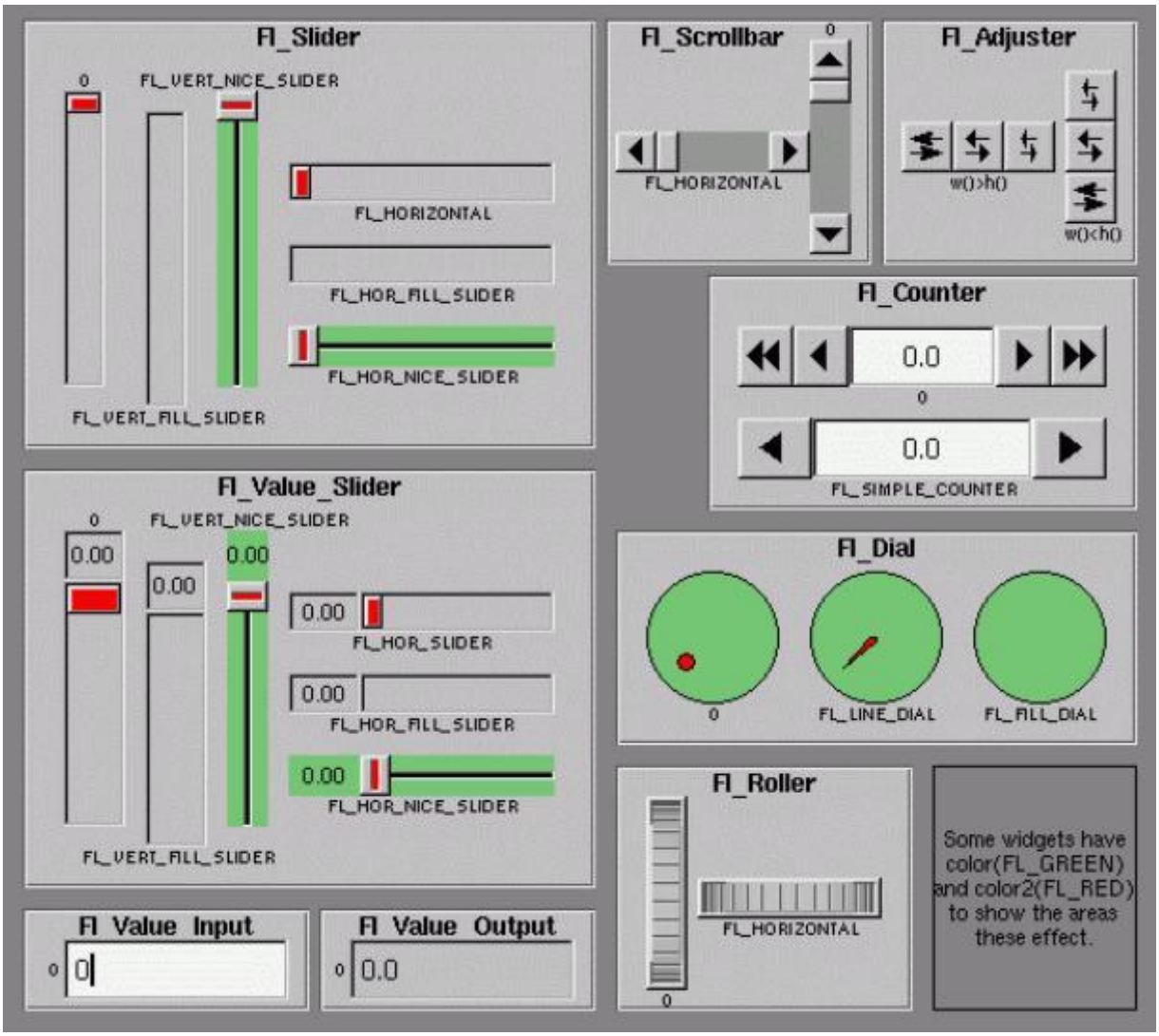


Рисунок 4.5 – Некоторые элементы управления библиотеки FLTK

Библиотека распространяется согласно лицензии LGPL и может функционировать с платформой X (UNIX) и Microsoft Windows 95/98/NT/XP. Автором этой библиотеки считается Билл Спайтзак (Bill Spitzak).

На рисунке 4.5 приводятся некоторые элементы управления, предоставляемые библиотекой FLTK. Она обладает большими возможностями построения пользовательского интерфейса по сравнению с GLUI, однако, FLTK также сложнее в применении.

4.4.3 Interactive Visualisation Framework - Ivf++

Interactive Visualisation Framework Ivf++ [7] - библиотека C++, предназначенная для удобного создания приложений трехмерной графики. Появилась эта библиотека благодаря Джонасу Линдеману (Jonas Lindemann) в конце 1999 для Microsoft Windows. В начале 2000 года библиотека была перенесена на Linux и IRIX 6.x. В то же самое время библиотека Ivf++ была сделана доступной как открытый проект, используя лицензию LGPL.

Как говорилось выше (см. п. 4.2), OpenGL использует процедурный подход к созданию приложений, что может серьёзно усложнить написание больших программных комплексов. Поэтому сегодня существует достаточно много объектно-ориентированных библиотек, таких как Open Inventor, OpenGL Performer и OpenGL Optimizer, инкапсулирующих вызовы функций OpenGL и предназначенных для создания приложений трехмерной графики. Однако, эти библиотеки сложны и запутаны, кроме того, не все из них являются кроссплатформенными или имеют открытый код. Основной целью Ivf++ является упрощение использования OpenGL при создании приложений компьютерной графики. Ivf++ включает порядка 140 классов и обладает большинством функциональных возможностей OpenGL. Недостающие возможности, могут быть легко расширены, путем получения производных классов от базовых классов Ivf++. Программный интерфейс OpenGL не скрыт полностью в Ivf++, команды OpenGL могут быть непосредственно вызваны из методов классов Ivf++.

Для работы Ivf++ кроме OpenGL необходимы также такие библиотеки

- The GLE Tubing and Extrusion Library 3.0.x (http://linas.org/gle/)

- Fast Light Toolkit 1.0.x (FLTK) (http://www.fltk.org)
- GLTT 2.x (http://www.moonlight3D.org/gltt)
- Freetype 1.3.x (http://www.freetype.org)

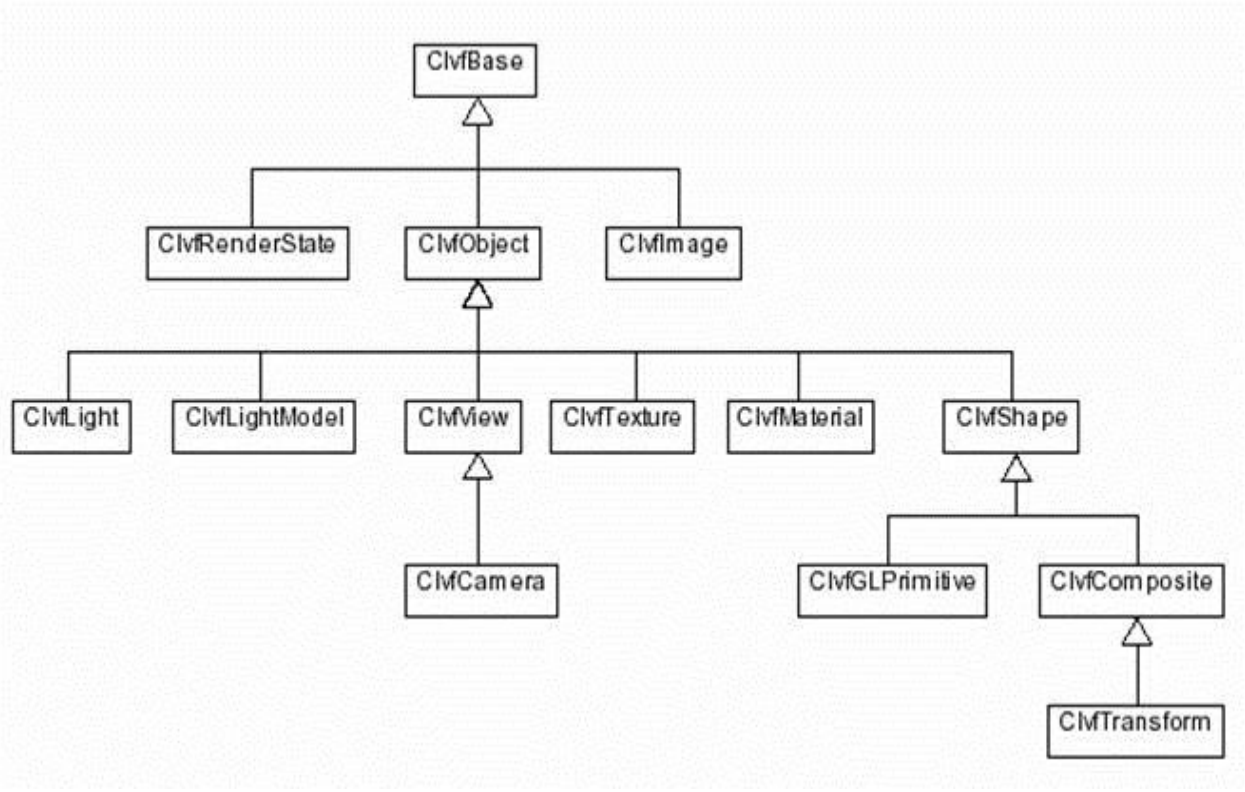


Рисунок 4.6 – Иерархия классов библиотеки Ivf++

На рисунке 4.6 схематично показана иерархия классов библиотеки Ivf++. Большинство классов в Ivf ++ получено из трех основных абстрактных классов, а именно: CIvfBase, CIvfObject и CIvfShape. Класс CIvfBase является корневым для всех других классов в библиотеке и, в основном, выполняет подсчет ссылок на объекты. От класса CIvfObject наследуются все классы, использующие работу с OpenGL различными способами. Класс CIvfShape используется всеми визуальными классами в библиотеке. Класс CIvfComposite может содержать набор дочерних объектов, полученных из CIvfShape, что позволяет определить иерархический граф сцены. CIvfTransform используется для создания и работы с иерархическими моделями, например, такими как руки робота.

4.5 OpenGL 2.0

Спроектированный для решения «больших» производственных задач, стандарт OpenGL благодаря своей универсальности, стабильности и кроссплатформенности снискал уважение у различных разработчиков приложений компьютерной графики. Совершенствование графических акселераторов и алгоритмов обработки графических данных привело к необходимости развития базового стандарта OpenGL. Разработчики компьютерной графики получили в свое распоряжение отличную библиотеку и несколько универсальных надстроек наподобие Open Inventor и Open Optimiser [10]. Поставщики графических микросхем для ПК самостоятельно создавали расширения OpenGL, которые стали вдвое больше самого стандарта. Противоречия между производителями при этом не идут OpenGL на пользу. Например, компания nVidia предоставляет разработчикам специальные драйверы и инструментарий, которые расширяют OpenGL, но совместимы только с ее графическими микросхемами. Еще запутанней ситуация с аппаратными технологиями сглаживания каркасных сеток: ATI и nVidia предлагают несовместимые между собой алгоритмы. Аналогичная история произошла и с шейдерами (см. п. 4.6); впрочем, в последнем случае все обошлось простым переименованием версий таким образом, что по версии шейдера теперь можно определять, для чьих микросхем он создавался.

Между тем, Microsoft разработала свой собственный вариант программирования мультимедиа-приложений DirectX (см. п.4.3). К основному преимуществу этой библиотеки можно отнести более высокую производительность для платформы Windows, чем у OpenGL. Однако DirectX обладает и серьезными недостатками: это отсутствие версий библиотеки для других платформ; не устоявшаяся 3D-часть библиотеки Direct3D; нежелание корпоративных пользователей переходить на стандарт, который никак не может обрести законченный вид.

Сторонникам OpenGL не остается ничего другого, как под давлением рынка совершенствовать стандарт.

Комитет разработчиков стандарта во главе с компанией 3Dlabs предпринял в 2001 году попытку подогнать OpenGL под нужды современного рынка компьютерной графики – появился

стандарт OpenGL 2.0, который на данный момент существует в виде предварительной спецификации [11]

Недавно компания nVidia, крупный производитель графических процессоров, в содружестве с Microsoft обнародовала свой новый язык Cg (от Computer Graphics), компилятор и несколько инструментальных средств к нему, документацию и множество примеров [12]. Этот бесплатный инструментарий призван помочь в создании эффективных 3D-приложений с использованием языка шейдеров (см. п. 4.6) нового поколения в средах Windows и Linux, OpenGL и DirectX. Стоит отметить, что спецификация языка шейдеров от nVidia полностью совместима с аналогом в DirectX 9.

Редакция OpenGL 2.0 является во многом концептуальным шагом. Разработчики стандарта пытаются подогнать платформу не только к современным технологиям, но и создать гибкую платформу для будущих усовершенствований. Поэтому значительные перемены ориентированны на модернизацию внутренней архитектуры API. Но для того чтобы переход на новый стандарт был как можно мягче, разрабатывается сразу два его варианта. Первый описывает интерфейс, позволяющий без модификаций запускать приложения, написанные для OpenGL 1.3, но пользователей заранее уведомляют, что в будущем он поддерживаться не будет. Второй вариант, «чистый» OpenGL 2.0 является основным.

Разработчики спецификации преследуют такие цели:

1. модернизация внутренней архитектуры библиотеки;

2. перевод основной части функциональности на программируемые шейдеры (см. п. 4.6);

3. реализация методов управления памятью и синхронизацией.

Первый и второй пункты непосредственно связаны между собой. По мнению разработчиков, переход от фиксированной функциональности к программируемости является приоритетным. Этим шагом как минимум решается проблема чрезмерного избытка расширений OpenGL, которые уже сейчас имеют вдвое больший объем описаний, чем спецификация базового стандарта OpenGL;

внедрив же эффективный механизм программируемости, большинство расширений без труда можно реализовать посредством шейдеров (см. п. 4.6). Этот шаг позволит уже в ближайшем будущем значительно обогатить возможности API без создания новых специализированных расширений и драйверов.

Реализация методов управления памятью и синхронизацией служат для решения другой назревшей проблемы — отсутствие средств взаимодействия с мультимедиа данными других типов.

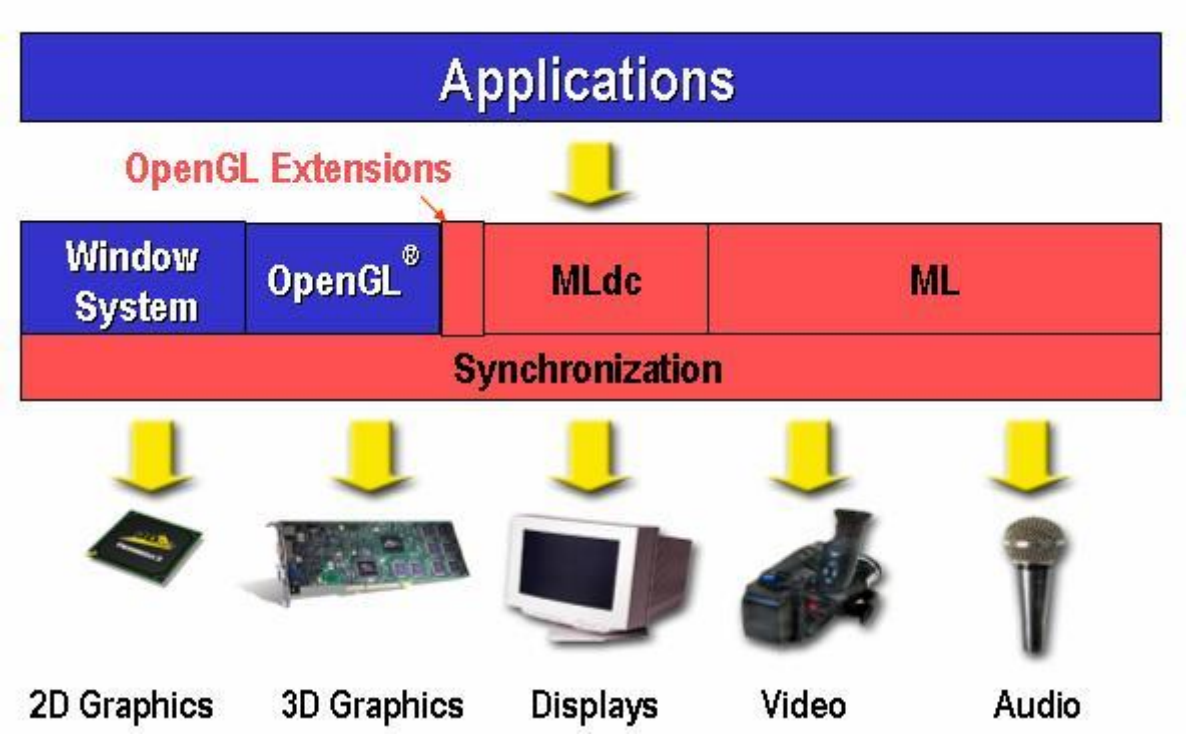


Рисунок 4.7 - OpenML

Поддержка звука и видео координируется с рабочей группой Khronos, развивающей интереснейший проект OpenML (см. рисунок 4.7). Главной задачей этой библиотеки является создание интегрированной среды для разработки приложений, использующих мультимедиа данные, а от OpenGL 2.0 в целях совместимости требуется соответствие ей по механизмам управления памятью, синхронизации и контроля времени.

Есть в OpenGL 2.0 еще одно заметное нововведение — объекты, что, в основном, связанно с реструктуризацией механизма управления памяти, в которой объекты играют важную роль. Такой подход облегчит разработчикам создание приложений трехмерной компьютерной графики.

4.6 Технология шейдеров

Шейдер — программируемый эффект, программа-функция, которая выполняется аппаратным ускорителем наравне со стандартными («зашитыми») функциями. Шейдеры пишутся при помощи набора микрокоманд, т.е. на языке подобном ассемблеру. В ранних программах рендеринга шейдерами назывались части, которые отвечали за закраску (shading) поверхностей. До появления шейдеров, в ускорителях реализовывался стандартный набор эффектов, возможности которого жестко ограничивали качество графики. Шейдеры разделяются на пиксельные (обрабатывающие пиксели) и вершинные (обрабатывающие вершины треугольников или других примитивов).

В таких известных компаниях как Pixar и SideFX написание шейдеров является неотъемлемой частью подготовки сцен для фотореалистичного рендеринга. Небольшие программы, написанные на Си-подобном языке, выполняют генерацию текстур и формируют поверхности объектов. Благодаря шейдерам многие процессы создания поверхности объектов и их текстурирования процедурными методами выполняются гораздо технологичней.

Такой подход во многом идет вразрез с последними тенденциями в области моделирования сцен. Рынок уже насыщен мощными интерактивными визуально-контролируемыми средствами построения сцен и моделей, и те, кто привык пользоваться мощными интерактивными средствами проектирования, больше не желают описывать модели и сцены в текстовых файлах. Но, несмотря на это, технология шейдеров находит себе применение в новых программных продуктах. Для интерактивного применения шейдеров эффективным является путь создания визуальных интерфейсов — принципиальная функциональность задается в виде алгоритма, а настройка параметров осуществляется визуально, с быстрым рендерингом и визуальной оценкой результата. Поэтому вполне естественным является переход аппаратных и программных платформ рендеринга в реальном времени на подобную технологию. Ведь именно реалистичность виртуальных сцен — главное достижение первых графических процессоров с поддержкой шейдеров для ПК от ATI и nVidia.

В OpenGL 2.0 (см. п. 4.5) шейдеры фактически являются дополнением — графические процессоры имеют те же конвейеры и блоки обработки графической информации, но в дополнительных узлах теперь ко всему прочему будут размещены блоки обработки шейдеров:

- Vertex Processor - управление параметрами вершин и их трансформацией;
- Fragment Processor - растровая обработка;
- Unpack Processor - распаковка текстур в память графического процессора;
- Pack Processor - архивирование битовых карт из памяти процессора в память приложения.

Таким образом, разработчик получает дополнительный инструмент и может использовать его совместно с традиционным программированием. Но все же существует несколько ограничений. Так, в случае запуска шейдеров, становятся недоступными некоторые функции из традиционного блока обработки, отвечающего за аналогичные преобразования. Например, после активации блока Vertex приложение не может использовать такие части стандартного конвейера, как трансформация, нормализация и масштабирование нормалей, управление источниками света и материалами, определение новых текстурных координат, методы отсечения, определенные пользователем.

Несмотря на то, что разработчики OpenGL 2.0 [13] определяют четкую границу между программируемостью при помощи шейдеров и фиксированной функциональностью (традиционный подход), наилучших результатов пока можно достичь, используя смешанное программирование 3D-приложений. Язык шейдеров является узкоспециализированным и в OpenGL 2.0 не может полностью заменить собой традиционный подход. Основной причиной этого является принцип работы шейдера, в соответствии с которым все элементы обрабатываются последовательно, по одному.

Шейдеры в OpenGL 2.0 [11] представляют собой код на Си-подобном языке, а в программе они описываются в виде массива байт, идентификатор которого является указателем на этот массив:

const GLubyte *sampleVertexShader = «тело шейдера».

Используя идентификатор и функции OpenGL 2.0, в исходном коде приложения следует выполнить определенную последовательность операций для подготовки шейдера к запуску:

1. создать объект типа шейдер:
Glhandle sampleVS = glCreateShaderObject
(GL_VERTEX_SHADER);

2. выполнить связывание новой ссылки с массивом описания шейдера:
glAppendShader (sampleVS, sampleVertexShader);

3. скомпилировать шейдер: glCompileShader (sampleVS);

4. создать специальный объект Programm Object, который служит контейнером шейдеров: Glhandle ProgObj = glCreateProgramObject();

5. связать шейдер с ProgrammObject:
glAttachShaderObject(ProgObj, sampleVS);

6. выполнить связывание внутри Programm Object, который может содержать и шейдер Fragment:glLinkProgram (ProgObj);

7. сделать Programm Object текущим:glUseProgramObject (ProgObj);

С этого момента шейдером является ProgObj; он и будет выполняться внутри блока glBegin/glEnd.

В программировании трехмерной графики широко используются алгоритмы аффинных преобразований. В связи с этим наряду с типами integer, float, bool при программировании шейдеров в OpenGL 2.0 можно использовать векторы vec(2-4) и квадратные матрицы mat(2-4). Их компонентами являются значения типа float. И к компонентам векторов, и к компонентам матриц разработчик имеет прямой доступ. Большинство операций (умножение, деление, сложение, и т.п.) могут выполняться между данными различного типа. Так, можно трехкомпонентный вектор умножить на матрицу с тремя столбцами, а от матрицы — отнять скалярное значение. Помимо элементарных операций для

оперирования матрицами и векторами предусмотрено множество встроенных функций: тригонометрические, логарифмические, геометрические (вычисление расстояний, векторное и скалярное произведения и пр.). Следует отметить, что встроенные функции реализуются аппаратно, поэтому их использование не влечет за собой значительных временных расходов.

В распоряжении разработчика шейдеров имеются привычные возможности языка Си: работа с массивами, условные операторы (if-else), циклы (for, while, do-while) возможность определения собственных функций и т.п.

Выводы

Компьютерная графика занимает важное место во многих сферах жизнедеятельности человека. Она находит применение в промышленности, в бизнесе, науке и образовании, в медицине, в индустрии развлечений и кино (см. раздел 1). Особенное место занимает компьютерная графика в системах автоматизированного проектирования (САПР) и моделирования (см. п. 1.2, п. 1.3). Не представляется возможным также без компьютерной графики создавать пользовательские интерфейсы (см. п. 1.4), способствующие эффективному взаимодействию человека и вычислительной машины.

Компьютерная графика обладает важными особенностями – во-первых, обработка графических данных может быть реализована по конвейерному принципу (см. раздел 2), что существенно ускоряет процесс получения изображения, во-вторых, простота и единообразие многих этапов визуализации (см. п. 3.4) позволяет создавать аппаратные реализации этих этапов, что ещё больше повышает скорость обработки графических данных. Поэтому на сегодняшний день существует много графических процессоров (графических акселераторов), которые берут на себя решение всех задач, связанных с графикой – это, в свою очередь, освобождает центральный процессор вычислительной машины от решения таких задач и позволяет ему в это же самое время заниматься решением других проблем. Всё это способствует повышению производительности вычислительной машины вцелом.

Для создания компьютерной графики необходимо использовать программные средства управления графическими устройствами. Это могут быть как непосредственные вызовы

соответствующих машинных команд или команд ассемблера, так и стандартизированные, независимые от конкретной аппаратуры прикладные программные интерфейсы (API), например, такие как OpenGL (см. п.4.2) или Direct3D (см. п.4.3). Они позволяют создавать прикладные приложения трехмерной компьютерной графики независимо от графической подсистемы ЭВМ.

Библиотека OpenGL считается одной из лучших библиотек для профессионального применения. Одним из главных её конкурентов является Direct3D из пакета DirectX, разработанный фирмой Microsoft. Стандарт Direct3D создавался исключительно для игровых приложений. Если сравнивать эти две библиотеки, то нельзя сказать, что одна из них лучше, а другая хуже, у каждой библиотеки имеются свои особенности. Например, если сравнивать их в плане переносимости программ с одной платформы на другую, то Direct3D будет работать только на платформах Intel под управлением операционной системы Windows, в то время программы, написанные с помощью OpenGL можно успешно перенести на такие платформы как Unix, Linux, SunOS, IRIX, Windows, MacOS и многие другие. Это означает, что при помощи OpenGL можно создавать кроссплатформенные графические приложения, которые одинаково функционируют не только на системах с различной графической подсистемой, но и на вычислительных машинах с различной аппаратной архитектурой, а также на различных платформах операционных систем.

В плане объектно-ориентированного подхода OpenGL уступает Direct3D. Стандарт OpenGL работает по принципу конечного автомата, переходя из одного состояния в другое и совершая при этом какие-либо преобразования. Однако OpenGL имеет множество высокоуровневых надстроек, таких как Open Inventor или Interactive Visualisation Framework (см. п.4.4.3), которые полностью поддерживают объектно-ориентированный подход к построению приложений компьютерной графики. Кроме этого, OpenGL развивается – создана спецификация OpenGL 2.0 (см. п. 4.5) с поддержкой технологии шейдеров (см. п. 4.6).

Итак, OpenGL представляет собой единый стандарт для разработки трёхмерных приложений, сочетает в себе такие качества как мощь и в то же время простоту. Кроссплатформенность позволяет без труда переносить программное обеспечение с одной операционной системы на

другую. Библиотека OpenGL предоставляет в распоряжение всю мощь аппаратных возможностей, которые имеются на данной вычислительной машине, что избавляет от беспокойства о конкретных деталях используемого оборудования.

Таким образом, OpenGL прекрасно подходит как для профессиональных систем компьютерной графики, так и для простых программ.

Литература

1. Эйнджел Эдвард Интерактивная компьютерная графика. Вводный курс на базе OpenGL, 2 изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 592 с.: ил. – парал. тит. англ.
2. Шикин А. В., Боресков А. В. Компьютерная графика. Полигональные модели. – М.: ДИАЛОГ-МИФИ, 2000. – 464 с.
3. Роджерс Д. Алгоритмические основы машинной графики: пер. с англ. – М.: Мир, 1989. – 512 с., ил.
4. http://www.opengl.org
5. Kilgard M.J. An OpenGL Toolkit. The X Journal, SIGS Publications, November/December 1994.
6. Schiefler R.W., Gettys J., Newman R. X-Window System. – Digital Press, 1988.
7. Kilgard M. OpenGL Programming for the X Windows System. – Addeson-Wesley, Reading, MA, 1996.
8. http://www.cs.unc.edu/~rademach/glui/
9. http://www.fltk.org
10. Евгений Валентинов Open Inventor как средство разработки интерактивных графических приложений. // Открытые системы, 1997, № 6
11. http://www.3dlabs.com/support/developer/ogl2/specs/index.htm
12. Питер Коуэн nVidia представляет графический язык Cg. // Computerworld Россия, 2002, № 28-29
13. http://www6.tomshardware.com/graphic/02q1/020222

II. КОНТЕКСТНАЯ ВИЗУАЛИЗАЦИЯ ПРОСТРАНСТВЕННЫХ ДАННЫХ

1. Введение

Визуализация пространственных данных используется в основном в задачах научной визуализации. Научная визуализация – это создание графических образов, в максимально информативной форме воспроизводящих значимые аспекты исследуемого процесса или явления. При этом большой объем результатов моделирования представляется в компактной и легко воспринимаемой форме. Преставление в виде графических образов позволяет исследователю увидеть изучаемую систему или процесс изнутри, что было бы невозможно без визуализации данных. И, иногда, именно визуализация приводит к полному пониманию явления.

Например, в системе FlowVision [1], предназначенной для визуализации сложных трехмерных течений жидкости или газа, представление объемных характеристик исследуемых распределений основано на применении интерактивной анимации. Кадр анимационной последовательности изображает распределение физической величины в некоторой плоскости сечения посредством линий уровня, тоновой заливки, векторов на плоскости или в виде изоповерхности. Пользователь имеет возможность просматривать в реальном режиме времени движущееся изображение, соответствующее движению плоскости сечения или (для изоповерхностей) изменению величины, и управлять просмотром при помощи манипулятора, что позволяет ему видеть объемные характеристики исследуемого течения.

В [2] также исследуется проблема визуализации трехмерных течений жидкости, однако здесь предметом изучения являются течения в природных объектах (результаты математического моделирования гидродинамики Азовского моря). Практически полная информация о двумерных течениях может быть передана при помощи текстур, ориентированных вдоль направления течения совместно с их анимацией. Для отображения третьей компоненты вектора скорости применяется цвет; ведутся также эксперименты по визуализации трехмерных течений при помощи трехмерных текстур и объемного рендеринга средствами библиотеки OpenGL.

Как и предыдущие два примера, работа [3] связана с визуализаций характеристик течения жидкости в трехмерном пространстве. В данном случае предметом исследований является топология рециркуляционной зоны, возникающей при

стационарном ламинарном движении жидкости в контейнере с локальным теплообменом на горизонтальных поверхностях. Частицы, находящиеся внутри рециркуляционной зоны, перемещаются по траекториям, лежащим на поверхности тора; вне этой зоны движение частиц хаотично. Для выявления топологии рециркуляционной зоны строятся траектории движения некоторого числа маркеров, а затем выполняется триангуляция.

В работе [4] предлагаются методы изучения гидродинамики с помощью визуализации турбулентности потоков.

На сайте [5] можно найти представительную галерею методов научной визуализации в применении к ряду задач исследовательского и прикладного характера. Одна из задач, относящаяся к проблемам экологии, посвящена визуализации распространения загрязняющих веществ на местности со сложным рельефом. Для представления динамики фронта распространения загрязнений по земле сначала строятся изолинии концентрации веществ на нулевой высоте; затем вычисляются проекции изолиний на рельеф поверхности земли.

В рассмотренных примерах усилия разработчиков направлены на поиск выразительных средств графики и анимации, позволяющих в максимально наглядной форме отобразить существенные для исследователя аспекты изучаемого процесса или явления. Что в конечном итоге приводит к более глубокому осмыслению физического процесса. Несколько иначе ставится задача визуализации, составляющая предмет настоящей статьи. Ее цель можно сформулировать как отображение трехмерных скалярных и векторных полей в контексте среды обитания человека - например, визуализация поля температуры или скорости движения воздуха в жилом помещении. Более точно, суть задачи можно охарактеризовать как отображение среды обитания человека с учетом факторов, определяющих степень ее привлекательности для человека, комфортности или пригодности для того или иного функционального применения. К числу таких факторов могут относиться эстетичность среды (ее внешний вид), характеристики микроклимата (температура, скорость движения воздуха, влажность), экологии (концентрация загрязняющих веществ, шумоизоляция), акустика, освещенность и др. В данном случае важны не столько нюансы визуализируемых распределений, сколько создание обобщенного образа среды, включающего реалистичное изображение ее внешнего вида и пространственные распределения тех или иных характеристик. Поэтому выбор

визуальной концепции должен основываться на иных критериях, нежели принятые в классической научной визуализации,

Соответствующий подход, называемый далее *контекстной визуализацией пространственных данных*, реализован в системе Visualizer. Visualizer поддерживает визуализацию трехмерных скалярных и векторных полей (распределение температуры, скорости движения воздуха, влажности и т.п.) вместе с визуализацией физических объектов, представляющих окружение изучаемого распределения – интерьер помещения, городской микрорайон, салон транспортного средства или "внутренность" портативного компьютера.

Дальнейшее содержание статьи построено по следующему плану. В главе 2 приведены примеры, иллюстрирующие возможные приложения Visualizer. В главе 3 описаны реализованные в настоящее время в Visualizer средства контекстной визуализации. Глава 4 посвящена вопросам реализации. В главе 5 рассмотрены возможные направления развития функциональности Visualizer.

2. Примеры контекстной визуализации

Примеры, представленные в этой главе, иллюстрируют возможности системы Visualizer. Распределения скалярных и векторных величин, показанные на рисунках, вычислены при помощи системы моделирования процессов теплообмена и движения воздушных потоков Flow, созданной по заказу фирмы Integra [6]. Изображения получены средствами разработанной системы физически аккуратного моделирования освещенности и построения фотореалистичных изображений [7], которая включает Visualizer и Flow в качестве своих модулей (подробнее о структуре и взаимодействии компонент этой системы см. гл. 3 и 4).

Пример 1. При планировании городских микрорайонов и построении новых жилых домов учитываются многие факторы, в том числе доминирующие направления ветра. На рис. 1 изображено векторное поле скорости движения воздуха в окрестности здания. Направление движения воздуха отображено направлениями стрелок, а абсолютная величина представлена цветом стрелок. Презентации такого рода могут быть полезны, например, для того чтобы выбрать максимально защищенное от ветра место.

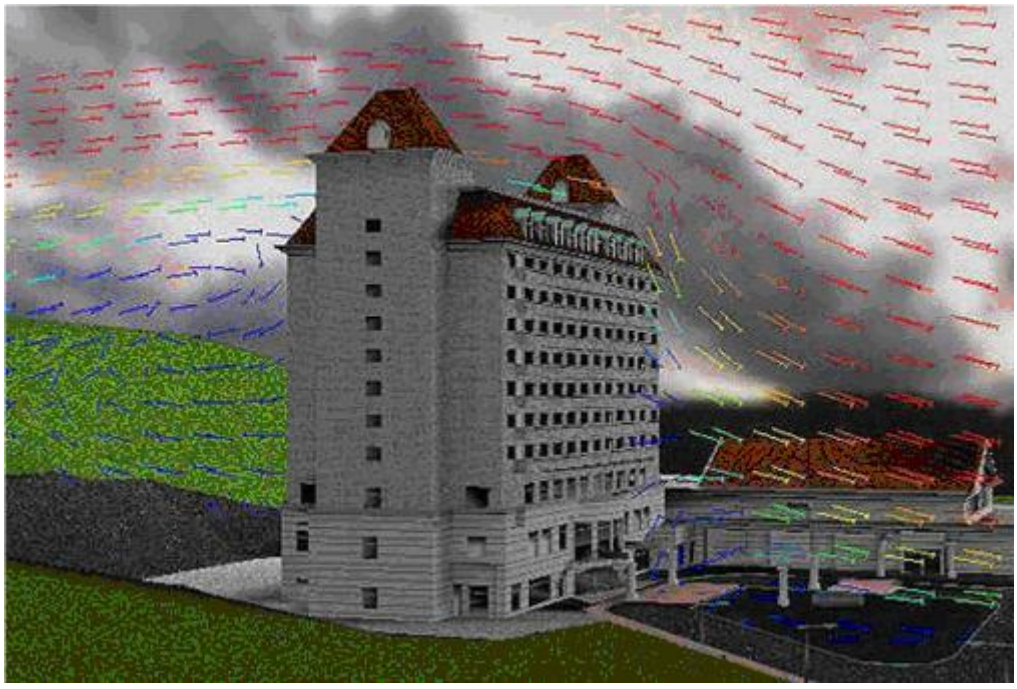


Рис. 1. Векторное поле скорости движения воздуха в окрестности здания

Пример 2. Комфортные условия для водителя и пассажиров в салоне автомобиля давно стало неотъемлемой частью создания новых моделей. Кондиционирование воздуха должно быть эффективным для людей, находящихся в салоне, и в то же время оно должно быть экономичным потому, что работа кондиционера снижает разгонные характеристики автомобиля и повышает расход топлива.

Рис. 2. Распределение температуры воздуха в салоне автомобиля

Цель презентации на рис. 2 – продемонстрировать эффективность системы кондиционирования воздуха в салоне автомобиля. Распределение температуры воздуха представлено полупрозрачной тоновой заливкой в вертикальной плоскости сечения, что позволяет видеть интерьер салона. Поместив модель человека на сиденье водителя, можно увидеть насколько использование кондиционера является эффективным для него.

Пример 3. При планировании интерьеров важно не только функциональное размещение мебели, но и правильное освещение помещения и создание оптимального микроклимата.

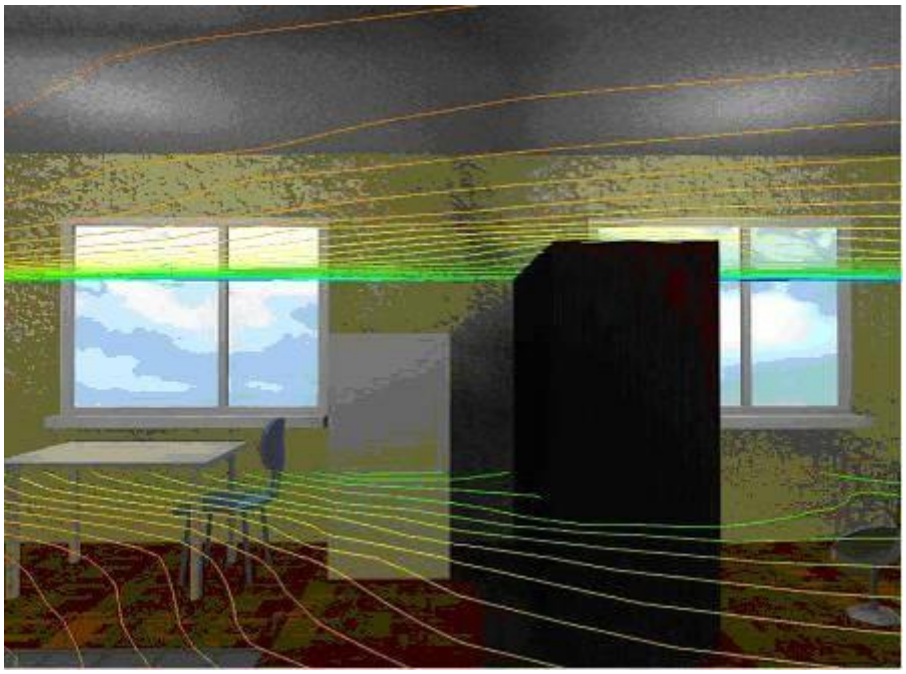


Рис. 3. Распределение температуры воздуха в жилом помещении

На рис. 3 показано распределение температуры воздуха в жилом помещении. Распределение отображено в виде изолиний на двух горизонтальных плоскостях, что позволяет лучше представить изменение температуры во всем пространстве помещения. В результате возможно представить не только внешний вид будущей комнаты, но и выбрать способы контроля внутреннего микроклимата, такие как кондиционер или нагревательный элемент. Совместное изображение предметов мебели и распределения температур позволяет правильно спланировать размещение таких устройств. Например, температурный режим для человека, сидящего за столом, представлен максимально наглядно.

Пример 4. Рис. 4 иллюстрирует применение системы Visualizer в области технического проектирования, в данном случае

– для проектирования систем охлаждения компьютеров. Одной из важных проблем проектирования портативных компьютеров является создание эффективной системы u1086 охлаждения. По сравнению с обычными персональными компьютерами проблема охлаждения для портативных компьютеров осложняется такими факторами, как очень плотное размещение составляющих компонент в компактном корпусе и необходимостью устанавливать минимально энергоемкий вентилятор, так как все производители стараются добиться максимального времени работы от автономного питания.

В данном примере презентация показывает распределение температуры в горизонтальной плоскости внутри корпуса портативного компьютера. При генерации изображения верхняя поверхность корпуса была удалена (хотя моделирование проводилось для закрытого корпуса). Потенциально опасные зоны перегрева выделены красным цветом. Как видно из приведенных примеров, контекстная визуализация пространственных данных оказывается весьма полезной при решении задач по планированию и проектированию различных объектов, связанных с жизнедеятельностью человека. Контекстная визуализация дает возможность быстро увидеть проблемные области в той или иной задаче и сконцентрировать усилия на их устранение.

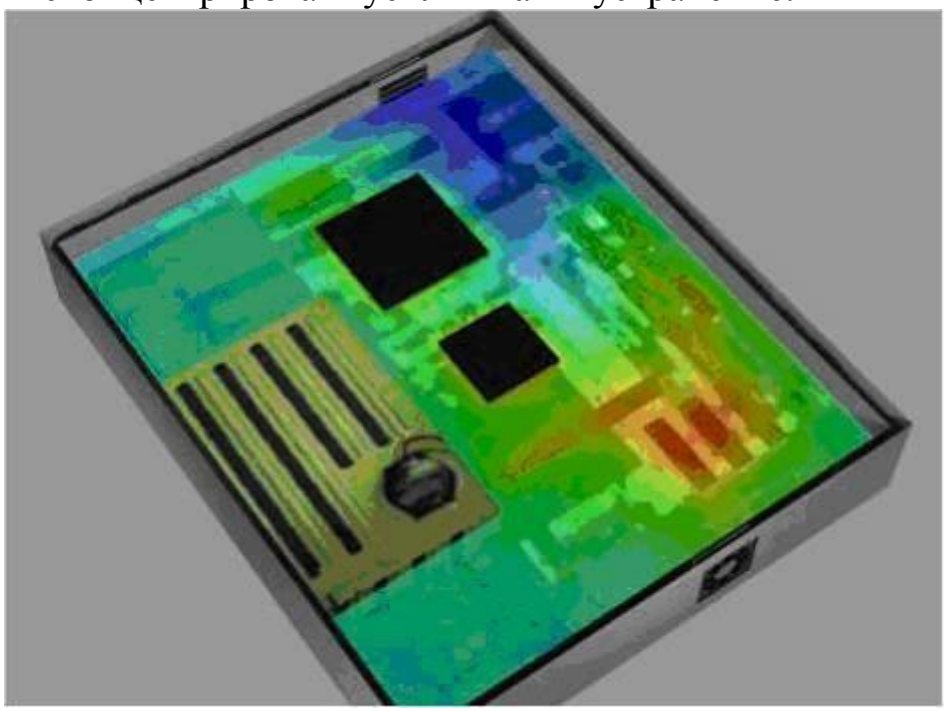


Рис. 4. Распределение температуры в корпусе портативного компьютера

3. Обзор возможностей системы Visualizer
3.1 Формы визуального представления пространственных данных

Visualizer поддерживает визуализацию скалярных полей, а также трех и четырехмерных векторных полей, заданных на равномерной или неравномерной сетке в трехмерном пространстве. Пространственные данные могут быть представлены различными способами. При выборе способов представления учитывалась главная цель контекстной визуализации: представление на одном изображении информации о пространственных данных, сохраняя при этом общий вид среды, для которой эти данные были рассчитаны. Для скалярных полей поддерживаются представления изолиниями, тоновой заливкой, изолиниями с тоновой заливкой. Эти представления могут быть изображены на одной заданной плоскости, на нескольких параллельных плоскостях (см. рис. 3) или на трех ортогональных плоскостях. Последний способ может быть полезен, например, для отображения распределения температуры воздуха вблизи угла комнаты. Поддерживается также представление скалярных полей в виде изоповерхностей. Для тоновой заливки и изоповерхностей имеется опция полупрозрачного изображения (см. рис. 2 и рис. 4), что позволяет видеть и высококачественное изображение среды, и распределение скалярной величины в пространстве. Изображение скалярных полей имеет, пожалуй, наиболее широкий спектр возможностей представления.

Для трехмерных векторных полей поддерживаются представления в виде одноцветных стрелок переменной длины или в виде разноцветных стрелок одинаковой длины. При представлении одноцветными стрелками направление и длина вектора соответствуют трехмерным данным. При представлении стрелками одинаковой длины направление стрелки соответствует направлению вектора, а цвет служит для представления его длины. Такое представление является информативным, как это проиллюстрировано на рис. 1. Четырехмерные поля (например, скорость движения воздуха и его температура) представляются стрелками переменной длины, направление и длина которых соответствуют трем первым компонентам, а цвет служит для отображения четвертого компонента.

3.2 Презентационные наборы данных

Система поддерживает одновременное отображение произвольных наборов данных, полученных в результате моделирования или измерения каких-либо пространственных характеристик. Исходными данными для визуализации служат файлы с массивами числовых данных, представленных в текстовом виде. Входные массивы могут содержать произвольное множество компонентов данных для каждой точки пространства. Visualizer позволяет загружать одновременно несколько входных массивов, которые, вообще говоря, могли быть получены из разных источников (например, при помощи разных программ моделирования). Пользователь на основе загруженных в данный момент входных файлов определяет наборы, состоящие из одного, трех или четырех компонентов, для визуализации их как скалярных полей, трех или четырехмерных векторных полей. Входные пространственные данные могут быть расположены на различных пространственных сетках.

temperature (C)
humidity.h (%)
air speed.x (m/s)
air speed.y (m/s)
pressure (Pa)
temperature (C)
humidity.h (%)
air speed.x (m/s)
air speed.y (m/s)
air speed.z (m/s)
temperature (C)
air speed.x (m/s)
air speed.y (m/s)
air speed.z (m/s)

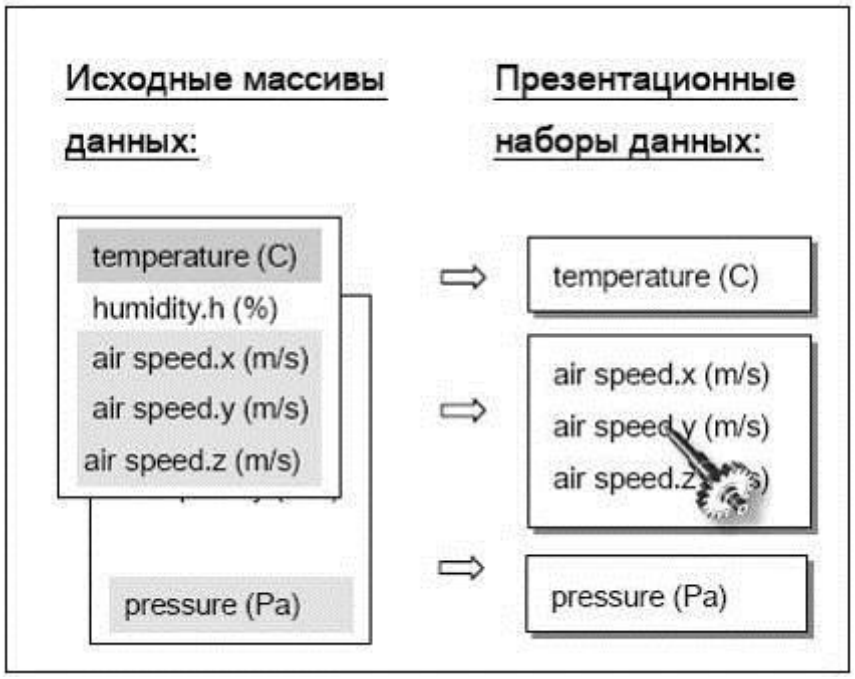


Рис. 5. Определение презентационных наборов данных

На рис. 5 показан пример определения трех презентационных наборов из двух массивов исходных данных: однокомпонентный набор данных temperature (температура), трехкомпонентный набор данных air speed (скорость движения воздуха) и однокомпонентный набор pressure (давление). Все презентационные наборы данных будут в дальнейшем обрабатываться независимо.

Возможность загружать независимые наборы пространственных данных позволяют, например, визуально сравнивать результаты компьютерного моделирования с реально измеренными данными. Такие данные можно рассматривать как две различные презентации, одновременно изображенные в среде.

3.3 Элементы презентации

Для каждого презентационного набора данных пользователь задает один из допустимых способов представления (см. п. 3.1). Пара (презентационный набор данных, способ представления) называется элементом презентации. Можно задать несколько элементов презентации, которые будут отображены одновременно в окружении контекстной сцены, представляющей среду.

Visualizer предоставляет развитый графический интерфейс для определения элементов презентаций, включая задание презентационных наборов данных, выбор визуальных форм представления и опций, таких как цветовая палитра, степень прозрачности для тоновой заливки и изоповерхностей и др.

Имеются также средства для визуального задания множества плоскостей, на которых требуется отобразить данные. Определив элементы презентации, пользователь может сохранить их для использования в последующих сеансах.
Система поддерживает также интерактивные u1079 запросы числовых значений для визуализируемых в данный момент наборов данных – пользователю достаточно указать мышью точку в каком-либо элементе презентации, чтобы получить информацию о значении скалярного или векторного поля в ней.

3.4 Средства анимации

Для более наглядного изучения данных Visualizer позволяет определять и воспроизводить анимационные последовательности, показывающие изменение исследуемого распределения данных в пространстве или времени. При пространственной анимации плоскость с изображением элемента анимации движется вдоль заданной пользователем траектории; при этом также могут выполняться вращения плоскости вокруг координатных осей. В простейшем случае плоскость (например, с изображением скалярного поля тоновой заливкой или изолиниями) просто движется поступательно по прямой линии. В этом отношении возможности анимации в Visualizer схожи со средствами, описанными в [1]. Такой подход позволяет проанализировать все пространство данных. Хотя набор данных здесь статический, но анимация покажет его изменение вдоль определенного направления или направлений.

Анимация по времени возможна для наборов данных, которые содержат информацию об изменении поля данных с течением времени. В этом случае Visualizer может сгенерировать и воспроизвести анимационную последовательность изображений, показывающих изменение распределения в течение заданного пользователем промежутка времени. Это уже позволит визуально анализировать динамические процессы.

3.5 Встроенная система моделирования Flow

Как уже отмечалось в гл. 2, в рамках работ по моделированию и отображению среды обитания человека была также разработана система Flow (Г.В.Кирейко, В.А.Глухов, В.В.Видякин), моделирующая движение воздушных масс и распределение температуры с учетом процессов тепловой радиации, а также естественной и принудительной конвекции. Flow поддерживает моделирование атмосферных процессов как с ламинарным, так и с

турбулентным режимом движения воздуха. Система предоставляет развитый графический интерфейс для задания граничных условий, параметров моделирования и анализа результатов. В качестве исходных данных используется сцена, для объектов которой заданы термодинамические свойства. В сцене также могут быть заданы погодные условия, включающие скорость и направление ветра, температуру и другие свойства наружного воздуха, время года, время суток и проч.

3.6 Инструмент редактирования сцен и моделирования освещенности

Visualizer и Flow реализованы как дополнительные модули системы построения фотореалистичных изображений [7-8], предназначенной для приложений в области архитектуры и светотехники. Эта система содержит средства физически аккуратного моделирования глобальной освещенности сцены с использованием метода Монте-Карло или метода излучательности и позволяет получать качественные высокореалистичные изображения сцен. Это является важной предпосылкой для получения убедительных и достоверных презентаций в Visualizer.

Система построения фотореалистичных изображений представляет собой интегрированную среду, из которой пользователь может вызывать графические инструменты для визуального редактирования сцен и моделирования освещенности, а также вспомогательные модули, такие как Flow и Visualizer. Предполагается, что первоначально геометрическая модель сцены создается в какой-либо из внешних систем трехмерного моделирования (3D Studio, AutoCad и т.п.) и затем импортируется в формат графической системы. Там пользователь может дополнить модель определениями источников света в физических величинах, параметрами естественного освещения, атрибутами, задающими оптические и термодинамические свойства объектов. Поддерживаются также простые операции редактирования геометрии, текстурирование, установка камеры и ряд других возможностей.

После того как сцена, представляющая окружение (контекст) для визуализации пространственных характеристик, подготовлена, пользователь может при помощи модуля Flow выполнить моделирование температуры u1080 и движения воздуха. Результаты моделирования сохраняются в виде файлов с массивами данных, которые могут быть использованы в Visualizer. Отметим, что данные для визуализации в принципе могут быть

получены и другими средствами, например, в результате измерений или при помощи других программ моделирования. В этом случае они должны быть предварительно преобразованы в формат, поддерживаемый Visualizer.

Далее при помощи Visualizer пользователь определяет презентационные наборы данных и способы их отображения, после чего выполняется контекстная визуализация данных.

При необходимости можно «упрятывать» отдельные элементы геометрии сцены. Таким образом, при отображении данных в Visualizer можно убирать элементы сцены, мешающие видеть представление данных. Например, на рис. 4 при визуализации распределения температуры была убрана верхняя часть корпуса портативного компьютера, хотя она присутствовала при моделировании тепло-массопереноса.

Можно также менять положение камеры, для того чтобы рассмотреть все интересующие области визуализированных данных. Причем благодаря контекстной визуализации пользователь легко определяет такие области, исходя сразу из нескольких критериев. Например, при проектировании портативных компьютеров инженеру может быть интересна область с запредельной температурой, а также пространство непосредственно около главного процессора.

4. Вопросы реализации

Схема реализации контекстной визуализации пространственных данных изображена на рис. 6.

Как уже упоминалось Visualizer работает с произвольными пространственными данными. Поэтому подготовка презентационных наборов данных реализована как отдельный компонент – адаптер пространственных данных (VDA – Volumetric Data Adapter). Исходный массив данных представлен в виде текстовых файлов, так как текстовой формат является наиболее универсальным при переносе между различными вычислительными, а также операционными системами. Данные состоят из заголовка и непосредственно тела. Заголовок содержит информацию о u1082 количестве компонентов, их именах, единицах измерения, а также сведения о пространственной сетке, на которой определены компоненты данных. Тело содержит собственно данные в виде последовательности чисел, которые интерпретируются в соответствии с заголовком.

VDA предоставляет интерактивный интерфейс для загрузки исходных массивов и выбора компонентов для презентационных

наборов. Он позволяет извлекать из массивов требуемые компоненты и делает их доступными для модуля визуализации.

Выделенные презентационные наборы данных передаются затем в генератор презентационной геометрии (PGG – Presentation Geometry Generator). PGG генерирует *презентационную геометрию* в форме плоскостей с изолиниями или с тоновой заливкой, изоповерхностей или стрелок, в зависимости от формы представления, выбранной пользователем. Для каждой формы представления имеется специальный PGG. PGG для изоповерхностей основан на алгоритме из [9].

Рис. 6. Реализация контекстной визуализации пространственных данных

Сформированная презентационная геометрия добавляется к геометрии текущей сцены и отображается средствами графической библиотеки (GL – Graphics Library) графической системы. Таким образом, сама сцена, которая часто состоит из большого количества объектов, требующих специальной предварительной обработки для эффективной визуализации, не видоизменяется. Добавленная презентационная геометрия содержит относительно небольшое число объектов, и она практически не влияет на общую эффективность визуализации. Это позволят включать и выключать визуализацию пространственных данных в интерактивном режиме.

Если определена анимация, то генерируется соответствующая последовательность презентационных геометрий, которая воспроизводится по запросу пользователя. Следует отметить, что геометрия каждого элемента презентации отображается посредством индивидуального канала, что позволяет включать и выключать визуализацию отдельных элементов независимо.

5. Возможные направления развития Visualizer

Архитектурные решения, принятые в реализации Visualizer, обеспечивают широкие возможности для наращивания его функциональности. Очевидный способ развития средств визуализации - добавление новых форм представления данных, например, в виде движущихся частиц, траекторий маркеров или текстур. Для этого достаточно реализовать соответствующие PGG.

Интересной представляется также идея введения в изображение сенсорных элементов (скажем, в форме человеческой фигуры), возможно, анимированных. Пример применения этой идеи - изображение движущейся в помещении фигуры человека с отображением на ее поверхности распределения температуры (тоновой заливкой или в виде изолиний) Подобный способ визуализации позволяет в максимально наглядной форме показать, насколько комфортно будет себя чувствовать в данном помещении взрослый человек, ребенок, или, возможно, домашнее животное.

На рис. 7 представлена схема реализации идеи контекстной визуализации пространственных данных на поверхности сенсорного объекта[1]. Для каждой формы представления данных (изолинии, тоновая заливка, стрелки) должен присутствовать PGG, на вход которого подается, с одной стороны, презентационный набор данных, с другой – геометрия сенсорного объекта, выбранного пользователем. На основе этих данных PGG генерирует презентационную геометрию. В остальном визуализация выполняется так же, как описано в гл. 4.

Другое возможное направление развития – реализация дополнительных программ моделирования, а также поддержка внешних программ моделирования, которая сводится к реализации соответствующих конвертеров данных.

[1] Подход не предполагает перевычисление распределения температуры с учетом тепла, выделяемого человеком.

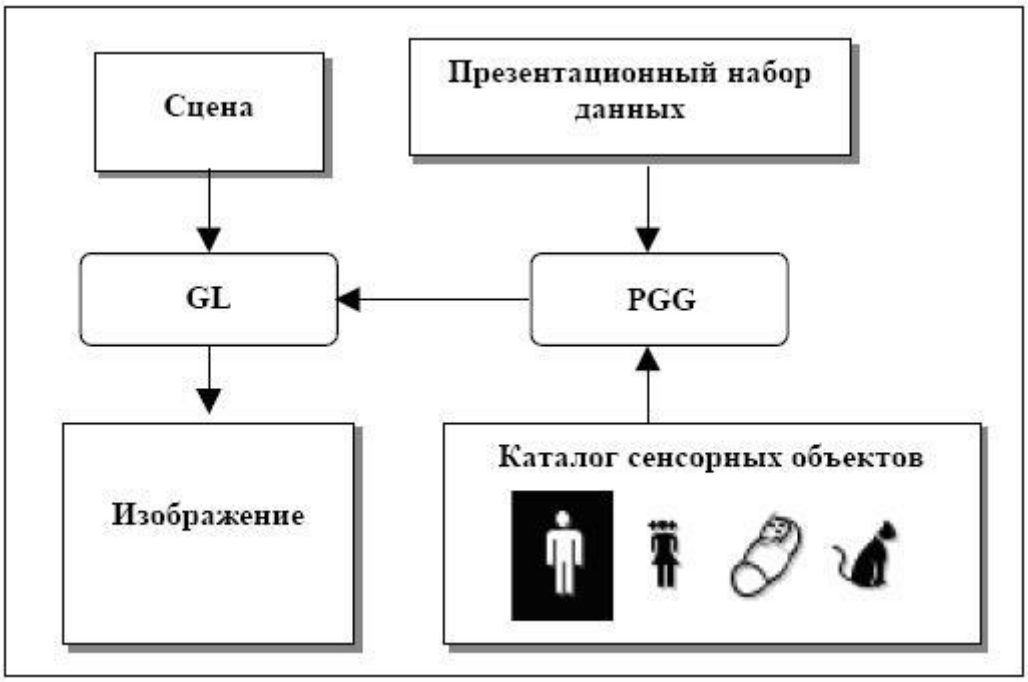


Рис. 7. Контекстная визуализация пространственных данных на поверхности сенсорного объекта

6. Заключение

Visualizer был реализован в отделе машинной графики Института прикладной математики им. М.В. Келдыша РАН как один из дополнительных модулей для системы физически аккуратного моделирования освещенности и построения фотореалистичных изображений для архитектурных приложений. Visualizer представляет собой инструмент для визуализации пространственных данных (распределения температуры, скорости движения воздуха и т.п.) в контексте среды обитания человека – в жилом помещении, салоне автомобиля, городском микрорайоне. Контекстная визуализация позволяет получать изображения, представляющие обобщенный образ среды с учетом факторов, определяющих
ее привлекательность для человека (в том числе, внешней привлекательности), комфортность (микроклимат, экология) или функциональную пригодность для какого-либо целевого использования. Созданные методы могут представлять интерес для использования в архитектуре, проектировании систем отопления и кондиционирования, градостроительстве, ландшафтном дизайне.

Высокоинформативные презентации, генерируемые при помощи Visualizer, могут служить инструментом принятия решений в разнообразных областях человеческой деятельности. Существенное достоинство контекстной визуализации – «понятность» изображений для лиц, не являющихся

профессиональными специалистами в какой-либо из перечисленных выше областей – чиновников, бизнесменов, потенциальных покупателей квартиры или загородного дома. Можно предположить, что рост требований, предъявляемых человеком к среде своего обитания, приведет к растущему спросу на средства визуализации такого рода.

Подобная направленность системы не умаляет ее ценности как инструмента визуализации для профессионального применения. Visualizer предоставляет достаточно богатый набор форм визуального представления данных, развитые возможности управления параметрами изображения и режимом просмотра, средства анимации.

В заключение хотелось бы отметить, что применимость идеи контекстной визуализации не ограничивается указанными выше областями. Visualizer с успехом использовался, например, для проектирования систем охлаждения в компьютерах, в строительном проектировании для предсказания деструктивного воздействия погодных факторов на элементы конструкции здания, в проектировании осветительных устройств. Мы полагаем, что идея контекстной визуализации может найти применения и в других отраслях человеческой деятельности, например, в сфере образования.

Версию статьи с цветными рисунками можно найти по адресу http://www.keldysh.ru/pages/cgraph/publications/cgd_publ.htm

Литература

[1] Аксенов А.А., Сельвачев А.Ю., Клименко С.В. Интерактивная анимация для визуализации
движения жидкости. Труды Межд. конф. по компьютерной
графике и машинному зрению –
Графикон-99, Москва, 1999.
[2] Аниканов А.А., Потий О.А. Проблемы и подходы к решению задачи визуализации данных о
течениях в природных объектах. Труды Межд. конф. по компьютерной графике и машинному
зрению – Графикон-2003, Москва, 2003.
[3] Гудзовский А.В., Клименко С.В. Визуализация рециркуляционных зон в трехмерном течении.
Труды Межд. конф. по компьютерной графике и машинному зрению – Графикон-99, Москва, 1999.
[4] Belotserkovskii O.M., Chechetkin V.M., Oparin A.M. Visualization of hydrodynamic calculations -

Межд. конф. по компьютерной графике и машинному зрению – Графикон-2003, секц. «Научная
визуализация в прикладных задачах», Москва, 2003.
[5] The Scientific Visualization Group, Institute for System Programming of Russian Academy of Sciences.
http://www.ispras.ru/~3D/eng; gallery:
http://www.ispras.ru/~3D/eng/problems/gallery.htm.
[6] http://www.integra.jp
[7] A.Khodulev, E.Kopylov Physically accurate lighting simulation in computer graphics software. Proc.
GraphiCon’96 - The 6-th International conference on Computer Graphics and Visualization, t.Petersburg,
1996.
[8] А.Г.Волобой, В.А.Галактионов, К.А.Дмитриев, Э.А.Копылов. Двунаправленная трассировка
лучей для интегрирования освещенности методом квази- Монте Карло. "Программирование", № 5,
2004.
[9] Lorensen W.E., Cline H.E. Marching Cubes: A High Resolution 3D Surface Construction Algorithm,
Proceedings of SIGGRAPH'87, ACM Computer Graphics, Vol.21, No. 4, July 1987.

III. ВИЗУАЛИЗАЦИЯ В СРЕДЕ DELPHI

1. Построение графиков. Компонент TDBChart

Для построения графиков используется компонент TDBChart, расположенный на странице *Data Controls* палитры компонентов Delphi.

1.1. Создание графика

Для того чтобы создать график, поместите в форму компонент TDBChart в форме будет создана заготовка. Затем щелкните мышью по этой заготовке два раза. Будет произведен переход в *редактор графика* В среде этого редактора можно установить свойства графика и его серий. Содержимое *редактора графика* представляет собой табулированный блокнот Для нового графика первой всегда показывается закладка Chart и для страницы *Chart* - закладка Series.

Каждая из закладок на странице Chart предназначена для установки параметров того или иного компонента графика

Series - содержит серии графика Серией называется набор точек графика На графике серии соответствует отдельная линия или ряд столбцов Если в графике несколько серий, будет визуализировано несколько линий или рядов столбцов.

General - устанавливает общие параметры графика, такие как объемность графика, отступы от краев, возможность увеличения (Zoom) и др.

Axis - устанавливает свойства осей.

В области ShowAxis определяется, для какой оси устанавливаются параметры - левой, правой, верхней или нижней. На странице, определяемой закладкой *Scales,* устанавливаются свойства масштаба значений по оси. *Automatic* устанавливает автоматическое масштабирование данных по оси - минимум и максимум вычисляются динамически, исходя из текущих значений серии. При отмене автоматического масштабирования можно установить автоматическое масштабирование минимального *{Minimum}* или максимального *(Maximum}* значения (отметка Auto). Для установки значения максимума и (или) минимума вручную следует нажать соответствующую кнопку *Change.* Шаг масштаба по оси выбирается автоматически, если в *Desired Increment* установлено значение 0. Установить фиксированное

значение шага можно, нажав кнопку *Change*. Закладка Title позволяет установить текст заголовка по оси, угол расположения заголовка и шрифт, которым заголовок выводится. Закладка **Labels** задает параметры меток для оси. Закладка **Tiks** устанавливает параметры самой линии оси.

Titles - определяет заголовок графика, шрифт, выравнивание и др. **Legend** - задает параметры легенды. Легенда - область графика, где приводится информация о графике.

Panel - определяет параметры панели, на которой располагается график.

Paging - устанавливает параметры многостраничного графика.

Walls - задает "стенку" графика.

1.2. Добавление серии в график и установка свойств серии в редакторе графика

1.2.1. Добавление серии в график

На графике одновременно может располагаться несколько серий. В большинстве случаев их значения строятся по одинаковому закону и две и более серий одновременно показываются в графике для сравнения. Чтобы добавить в график серию, следует на странице Chart, (закладка Series) нажать кнопку Add. После этого появится окно выбора типа серии (рис I.1).

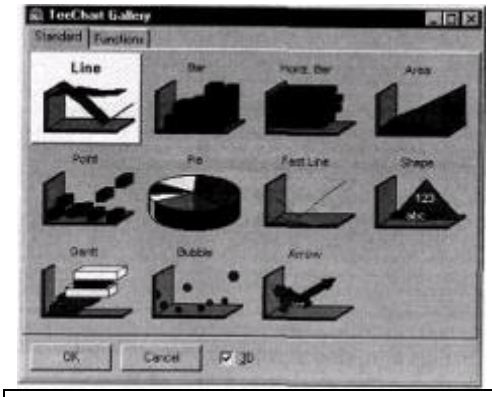


Рис I.1 Редактор графика - окно выбора типа серии

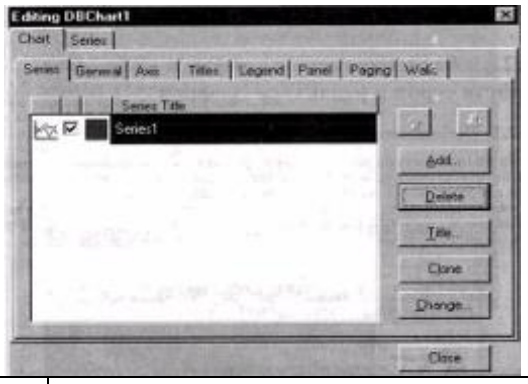


Рис I.2 Редактор графика - список серий графика

После выбора типа серии в график добавляется компонент, дочерний от базового типа TChartSeries - TLineSeries, TBarSeries, TPieSeries и т.д. Выберем серию типа Line и нажмем *Ok*. В окне страницы Chart (закладка Series) будет показана серия (рис. I.2).

Кнопка Add может использоваться для добавления других серий, кнопка *Delete* - для удаления текущей серии. После нажатия кнопки Title можно определить заголовок серии, кнопки *Clone* - создать новый экземпляр такой же серии в этом же графике, кнопки *Change* - изменить тип текущей серии.

Перейдем с закладки *Chart* на закладку Series. На этой странице представлен блокнот с закладками *Format, General, Marks, DataSource.* Рассмотрим свойства серии, которые можно установить на страницах, соответствующих этим закладкам.

1.2.2. Выбор источника данных

Несомненно, главные свойства серии можно определить на странице *DataSource.* На ней определяется источник данных для серии. Выпадающий список ниже закладки позволяет определить тип источника данных для серии:

No Data - серии не назначается источник данных. Далее мы собираемся сделать это программно. Кроме того, заготовленный шаблон серии может в разное время использоваться для показа данных из разных источников, которые мы также собираемся переключать программно во время выполнения.

Random Values - набор случайных чисел. Бывает полезен при формировании заготовки серии, источник данных которой мы собираемся установить позднее.

Function - функция (Copy, Average, Low, High, Divide, Multiply, Subtract, Add) - служит для построения графиков на основании данных в двух или более сериях.

DataSet - позволяет указать НД, значения полей (столбцов) которого будут использоваться для формирования точек серии. В качестве НД могут выступать компоненты TTable, TQuery, TClientDataSet.

Выберем *DataSet* и из выпадающего списка выберем компонент Table 1, ранее расположенный на нашей форме. Table1 (тип TTable) - набор данных, связанный с таблицей Kap_pryb.DB, где хранятся данные о зависимости между размером капитала некоторой фирмы (поле Kapital) и приростом дохода для каждого факта увеличения капитала (поле ProcKPred). Укажем, что поле Kapital содержит значения по оси X, а поле ProcKPred - значения по оси Y.

Замечание. Не все типы серии требуют значений по осям Y или X Для серий типа Pie, Bar можно указывать значения по одной

из осей и значения меток *Labels* В качестве меток могут использоваться символьные поля и поля типа даты и времени.

Серия типа Bar может содержать точки, сформированные как по осям X, Y, так и по оси Y (Bar) и меткам Labels

После того, как мы указали источник данных и поля для формирования значений по осям X и Y, нажмем кнопку Close и выйдем *изредактора графика* На ранее пустой панели будет построен график.

Если вернуться в *редактор графика,* на странице Series можно, помимо *DataSource,* увидеть закладки *Format, General, Marks* Их назначение-

Format - определяет свойства палитры, линий графика и т д

Gene/a/ - задает форматы данных

Marks - устанавливает марки - значения в рамке над точками серии.

1.2.3. Определение функций

На странице Series (закладка Data *Source)* в качестве источника данных можно определить функцию. Функция обычно используется для показа отношений между другими сериями.

Пусть для графика определены две серии, показывающие общую сумму продаж товаров по месяцам и сумму продаж для конкретного покупателя. Источниками данных этих серий выступают два компонента TQuery. Определим третью серию, которая показывала бы разницу между первыми двумя. Для этого добавим в график новую серию и на странице Series в закладке Data *Source* выберем *Function.* В диалоговом окне определения функций выберем тип функции - Subtract (вычитание). В области *Source Series \ Available* показаны серии, присутствующие в графике. Используя кнопку ”, переместим две упомянутые выше серии (источником которых служат TQuery) в область Selected.

Тогда на графике получим три серии, причем белым цветом показана серия, источником которой служит функция разности значений первых двух серий.

1.3. Добавление серии во время выполнения

Число серий, присутствующих в текущий момент в графике, во время выполнения можно определить при помощи метода компонента TDBChart **function SeriesCount: Longint;**

Список серий графика содержится в его свойстве **property Series[Index:Longint]:TChartSeries;** где Index лежит в диапазоне 0.. *SeriesCount*-1, поскольку отсчет серий идет с нуля.

Метод **procedure RefreshData;** обновляет данные в серии из наборов данных, которые служат их источником. Например, пусть график показывает отпуск товара конкретному покупателю. Источником данных для серии графика служит Query 1, которому в качестве параметра передается имя покупателя. При всякой смене имени покупателя происходит обновление Query 1, сопровождающееся изменением данных. Эти изменения отражаются в графике так:

```
Queryl.Open;
DBChartl. RefreshData;
```

Покажем, как можно добавить серию в график во время выполнения:

```
var
MySeries : TBarSeries;
MySeries := TBarSeries.Create(Self);
MySeries.ParentChart := DBChartl;
MySeries.SeriesColor := cIGreen;
MySeries.DataSource := Queryl;
MySeries.XLabelsSource := 'MES';
MySeries.YValues.ValueSource:='S';
MySeries.Active := True;
MySeries.Title := Table1POKUP.Value;
```

В приведенном примере создается серия типа Bar (вертикальная столбчатая диаграмма), в качестве родительского графика ей назначается DBChartl. В качестве источника данных назначается Queryl. Для формирования значений серии используются поля компонента Queryl: S (по оси Y) и MES (в качестве Labels). Затем серия активизируется (после этого она становится видна в графике) и ее заголовку присваивается наименование покупателя.

1.4. Работа с сериями. Компонент TChartSeries

Компонент *TChartSeries* является родительским типом для серий графика (для TLineSeries, TAreaSeries, TPointSeries,

TBarSeries, THorizBarSeries, TPieSeries, TChartShape, TFastLineSeries, TArrowSeries, TGanttSeries, TBubbleSeries).

1.4.1.Свойства компонента TChartSeries

property Active : Boolean; - активизирует (показывает) серию в графике (значение True) и дезактивизирует (скрывает) серию (False). Например: DBChartl.Series[0].Active :=True;

property DataSource : TComponent; - ссылается на компонент типа НД (TTable, TQuery, TClientDataSet) или на другую серию, откуда берутся данные для показа в серии. Например: DBChartl.Series[0].DataSource := Query2;

property HorizAxis: THorizAxis; - указывает, какая горизонтальная ось будет использована для серии. Значения: *aTopAxis* • верхняя горизонтальная ось; *aBottomAxis* - нижняя горизонтальная ось.

property Marks : TSeriesMarks; - описывает свойства марок серии, т.е. значений в прямоугольниках, рисуемых для каждого значения серии. Свойства объекта *Marks:*

property Arrow : TChartPen; - задает свойства пера, рисующего марку. Свойства объекта Arrow:

property Color: TColor; - цвет линий;

property Mode: TPenMode; - способ рисования линий;

property Style: TPenStyle; - стиль линий;

property Visible: Boolean; - видимость линий;

property Width: Integer; - задает ширину линий;

property ArrowLength : Integer; - длина в пикселях линии, соединяющей марку с соответствующим изображением элемента серии. По умолчанию 16;

property BackColor: TColor; - определяет цвет фона марки. По умолчанию $80FFFF (желтый);

property Clip: Boolean; - если содержит True, марки не могут накладываться на другие элементы графика (на легенду, метки осей и т.д.);

property Font : TFont; - определяет шрифт, которым выводится информация внутри марки;

property ParentSeries : TChartSeries; - содержит указатель на серию, к которой принадлежат марки;

property Style : TSeriesMarksStyle; - определяет содержимое марки. По умолчанию *smsLabel.* В обработчике события *TChartSeries. OnGetMarkText* можно переопределить значения,

принятые по умолчанию. Например: DBChartl.Series[0].Style := smsLabelValue;

Возможные значения свойства Style:

• ***smsValue*** **-** значения по осиУ (*YValue),* за исключением THorizBarSeries *(XValue).* Например, "9087";

• smsPercent - процентное значение, например "44%"; для форматирования процентного значения также используется свойство *TChartSeries. PercentFormat;*

• ***smsLabel*** **-** показывает метку, ассоциированную с точкой графика, например "Сахарный песок" (при построении графика продаж по товарам); в том случае, если метки со значениями не ассоциированы, в марках выводятся сами значения;

• ***smsLabelPercent*** **-** показывает метку и процентное значение, например "Сахарный песок 44%";

• ***smsLabelValue*** **-** показывает метку и значение, например "Сахарный песок 9087";

• ***smsLegend*** **-** показывает один из элементов легенды графика, список возможных значений доступен через свойство *TChartLegend. TextStyle;*

• ***smsPercentTotal*** **-** показывает процентное значение и общую сумму, от которой оно взято, например "44% от 20563".

• ***smsLabelPercent Total*** **-** показывает метку, процентное число и общую сумму, например "Сахарный песок 44% от 20563";

• ***smsXValue*** • показывает значение по оси X (XValue), например "01.02.2006";

• **property Transparent: Boolean; -** значение True определяет, что цвет фона марки не используется (в качестве фона используется "прозрачный цвет"); по умолчанию False;

property Visible : Boolean; -определяет, видимы ли (True) или нет (False) марки на графике.

property ParentChart : TCustomChart; - указывает компонент *TDBChart, к* которому принадлежит серия. Изменение этого свойства позволяет во время выполнения добавлять в график новые серии, показывать серии в других графиках. Например:

```
var
MySeries : TBarSeries;
MySeries := TBarSeries.Create(Self);
MySeries.ParentChart := DBChartI;
```

property PercentFormat : String; - определяет формат показа процентных значений;

property RecalcOptions: TSeriesRecalcOptions; - указывает перечень событий, приводящих к пересчету значений серии (

учитывается только для серий, свойство DataSource которых указывает на другую серию) по умолчанию *[rOnDelete, rOnModify, rOnInsert, rOnClear],*

property SeriesColor: TColor; - определяет цвет, которым выводятся значения серии в графике. Например:
DBChartI.Series[0].SeriesColor := clBlue;

property ShowInLegend: Boolean; - определяет, показывать ли (True) легенду или нет (False). по умолчанию True;

property Title: String; - определяет заголовок серии; по умолчанию заголовок отсутствует, но он может быть назначен в редакторе графика (кнопка Title в окне *Series).* Например: DBChart1.Series[0].Title := Edit1.Text;

property ValueColor[Index:LongInt]:TColor; - массив, определяет цвет элемента серии с номером Index, например, DBChart1.Series[0].ValueColor[2] := clAqua;

property ValueFormat: String; - определяет формат показа значений серии; при прорисовке осей используется для форматирования меток, при прорисовке серии используется для форматирования значений, показываемых в марках;

property ValueMarkTextlIndex:Longint]:String; - массив значений, выводимых в марках серии;

property VertAxis : TVertAxis; - определяет местоположение вертикальной оси - слева на графике *(aLeftAxis)* или справа *(aRightAxis);*

property XLabel[Index:LongInt]: String; - массив, хранящий метки серии по оси X; Index должен находиться в диапазоне *0. Count* -1;
DBChartI.Series[0].XLabel[2] := Edit2.Text; property **XLabelsSource: String; -** имя поля НД (или иного источника значений для серии), определяемого в свойстве *DataSource.* Содержимое этого поля служит для отображения значений по оси X. Поле должно быть типа, к которому применяется метод AsString. Если значение свойства опущено, значения по оси X не выводятся. Например: DBChartI.Series [0].XLabelsSource := 'MES';

property XValue[Index:LongInt] : Double; - возвращает значение в списке *XValues* (см. ниже) с индексом Index (значение в диапазоне *0. Count* -1).

property XValues:TChartValueList; - хранит значения серии по оси X. Значения из этого списка НЕЛЬЗЯ удалять, добавлять и т.д. напрямую. Для этого следует воспользоваться соответствующими методами компонента TChartSeries. Могут быть полезны следующие свойства TCliart ValueList:

• **property Valuel Index:LongInt |: Double;** - обеспечивает доступ к элементу серии с индексом Index (значение в диапазоне *0. Count -1).* Например:

```
DBChart1.Series[0].YValues.Value[2] := StrToFloat(Edit3.Text) ;
DBChart1.Series[0].Repaint;
```

• **property ValueSource : String;** - указывает источник данных для формирования значений по оси X. В зависимости от того, каков источник данных для серии (свойство *DataSource* компонента TChartSeries), может содержать:

1) имя поля - числового типа, типа даты, времени, даты и времени; в этом случае свойство серии *DataSource* должно ссылаться на НД (TTable, TQuery, TClientDataSet), например:

```
DBChart1.Series[0].DataSource := Query2;
DBChart1.Series[0].XValues.ValueSource := 'Pole1';
```

при этом необходимо помнить, что данные будут взяты в серию только из открытого НД; если НД закрыт, то получение данных будет отложено до открытия НД;

2) имя существующего TChart *ValueList* из другой серии; в этом случае свойство *DataSource* серии должно ссылаться на другую серию, например:

```
DBChart1.Series[0].DataSource := DBChart2.Series [4] ;
DBChart1.Series[0].XValues.ValueSource := 'X';
```

Свойства **property YValuetIndex:LongInt]: Double; property YValues: TChartValueList;** аналогичны свойствам XValue и XValues и используются для вертикальной оси.

1.4.2. Методы компонента TchartSeries

function AddXY(Const AXValue, AYValue: Double; Const AXLabel: String; ACoIor: TColor): Longint;

Добавляет новую точку в серию. Параметры AXValue и AYValue содержат соответственно значения по осям X и Y. Параметр AXLabel содержит метку для добавляемой точки серии. Параметр ACoIor определяет цвет. Функция возвращает позицию новой точки в серии. Например:

```
DbChartl.Series[0].AddXY(TmpX,TmpY,TmpLabel,clAqua);
```

function AddY(Const AYValue: Double; Const AXLabel: String; AColor: TColor): Longint; добавляет в серию новое значение по оси X. Применяется для тех серий, в которых график

строится по X и меткам значений по X (например, Pie, Bar). Назначение параметров такое же, как у метода *AddXY*.

procedure Assign Values(Source: TChartSeries); - копирует все точки из серии *Source* в текущую серию.

procedure CheckDataSource; - обновляет точки в серии, независимо от того, какой компонент является источником данных - набор данных или другая серия. Обновление производится по текущим данным источника. Метод рекомендуется вызывать в случае изменений данных в источнике.

procedure Clear; - удаляет все значения из серии; если вслед за этим не занести новых точек, будет показываться пустой график.

procedure ColorRange(AValueList: TChartValueList ;Const From Value, To Value: Double; AColor: TColor);
Изменяет цвет указанного диапазона точек серии. AValueList - либо XValues, либо YValues. From Value указывает начальное, а To Value конечное значение в списке AValueList. AColor - новый цвет. Например:

```
WITH DbChartl.Series[0] do begin
ColorRange(XValues,XValues.Value[2],
XValues.Value[2],clAqua) ;
END;//with
```

function Count : Longint; - возвращает число точек в серии. Например, поместить все значения по X и Y точек серии в ListBoxl:

```
ListBoxl. Items . Clear;
WtTH DbChartl.Series [0] do begin for i := 0 TO Count - 1 do
ListBoxl.Items.Add(
FloatToStr(XValues.Value[i]) + ' ' +
FloatToStr(YValues.Value[i])) ;
END;//with
```

procedure Delete(ValueIndex : Longint); - удаляет из серии точку с номером ValueIndex. График, к которому принадлежит серия, автоматически перерисовывается. Например:
DBChartl.Series[0].Delete(4);

procedure DoSeriesClick(ValueIndex:LongInt; Button:TMouseButton; Shift:

TShiftState; X, Y: Integer); virtual; - инициирует наступление события *OnClick.*

function GetCursorValueIndex : Longint; - возвращает индекс точки серии в *TChart ValueList,* ближе всего к которой расположен курсор мыши. Если такую точку определить не удается, возвращается - 1. Например, в следующем фрагменте Label7.Caption будет содержать индекс ближайшей точки к курсору мыши или '???', если такая точка не определена:

```
procedure TForm1.DBChartlDblClick(Sender: TObject);
var Tmp : Integer;
begin
Tmp := DBChartl.Series[0].GetCursorValueIndex;
IF Tmp >= 0 THEN Label7.Caption := IntToStr(Tmp)
ELSE Label7.Caption := '???';
end;
```

procedure GetCursorValues(Var x, y: Double); **-** возвращает значения по X и Y точки графика (а не только серии), ближе всего к которой расположен курсор мыши. Например, Label lO.Caption и Labell2.Caption в следующем фрагменте содержат соответственно значения координат X и Y графика, соответствующие точке, на которой находится курсор мыши:

```
var TmpX, TmpY : Double;
DBChartl.Series[0].GetCursorVaiues(TmpX,TmpY);
LabellO.Caption := Format('%10.2f,[TmpX]);
Labell2.Caption := Format('%10.2f,[TmpY]);
```

function GetHorizAxis: TChartAxis; - возвращает указатель на назначенную серии горизонтальную ось. Используя данный указатель, можно вызывать методы оси, обращаться к ее свойствам.

function GetVertAxis:TChartAxis; - возвращает указатель на вертикальную ось.

function MaxXValue: Double; virtual; - возвращает максимальное значение по **X.**

function MinXValue: Double; virtual; - возвращает минимальное значение по **X.**

function MaxYValue: Double; virtual; - возвращает максимальное значение по Y.

function MinYValue: Double; virtual; - возвращает минимальное значение по Y.

procedure RefreshSeries; - обновляет значения серии из источника данных, указанного в свойстве DataSource.

procedure Repaint; - приводит к полной перерисовке всего графика. Рекомендуется вызывать этот метод в случае изменения хотя бы одного из основополагающих свойств серии (например, при изменении значения в *DataSource* и др.).

function ValuesListCount:LongInt; - возвращает число списков значений точки, используемых в серии. Обычно это 2 (XValues и YValues), но некоторые серии используют 3 (BubbleSeries - XValues, YValues, Radius; GanttSeries -Y, Start,End).

function VisibleCount: Longint; - возвращает число точек серии, видимых на графике.

1.4.3. События компонента TChartSeries

property OnBeforeAdd: TSeriesOnBeforeAdd;
TSeriesOnBeforeAdd = Function(Sender: TChartSeries): Boolean of object;
Наступает перед добавлением точки в серию. В обработчике данного события может производиться анализ корректности добавляемых в серию точек. Наступает также при соединении серии с источником данных (TTable или TQuery).

property OnAfterAdd: TSeriesOnAfterAdd;
TSeriesOnAfterAdd = procedure(Sender:TChartSeries; ValueIndex:Longint) of object;
Происходит после добавления точки в серию.

property OnClear Values: TSeriesOnClear;
TSeriesOnClear = procedure(Sender: TChartSeries) of object;
Происходит при очистке серии от точек.

property OnClick: TSeriesClick;

TSeriesClick = procedure(Sender:TChartSeries; ValueIndex: Longint; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
Происходит при щелчке мышью на серии.

property OnGetMarkText: TSeriesOnGetMarkText;
TSeriesOnGetMarkText = procedure (Sender : TChartSeries ; ValueIndex :
Longint; Var MarkText: String)
Происходит при формировании марки для точки в серии. Обработчик может использоваться для изменения содержимого марки.

2. Графические возможности Delphi

Delphi позволяет программисту разрабатывать программы, которые могут выводить графику: схемы, чертежи, иллюстрации.

Программа выводит графику на поверхность объекта (формы или компонента Image). Поверхности объекта соответствует свойство canvas. Для того чтобы вывести на поверхность объекта графический элемент (прямую линию, окружность, прямоугольник и т. д.), необходимо применить к свойству canvas этого объекта соответствующий метод. Например, инструкция Form1.Canvas.Rectangle (10,10,100,100) вычерчивает в окне программы прямоугольник.

2.1. Холст

Как было сказано ранее, поверхности, на которую программа может выводить графику, соответствует свойство Canvas. В свою очередь, свойство canvas — это объект типа TCanvas. Методы этого типа обеспечивают вывод графических примитивов (точек, линий, окружностей, прямоугольников и т. д.), а свойства позволяют задать характеристики выводимых графических примитивов: цвет, толщину и стиль линий; цвет и вид заполнения областей; характеристики шрифта при выводе текстовой информации.

Методы вывода графических примитивов рассматривают свойство Canvas как некоторый абстрактный холст, на котором они могут рисовать (canvas переводится как "поверхность", "холст для рисования"). Холст состоит из отдельных точек — пикселов. Положение пиксела характеризуется его горизонтальной (X) и

вертикальной (Y) координатами. Левый верхний пиксел имеет координаты (0, 0). Координаты возрастают сверху вниз и слева направо (рис. II.1). Значения координат правой нижней точки холста зависят от размера холста.

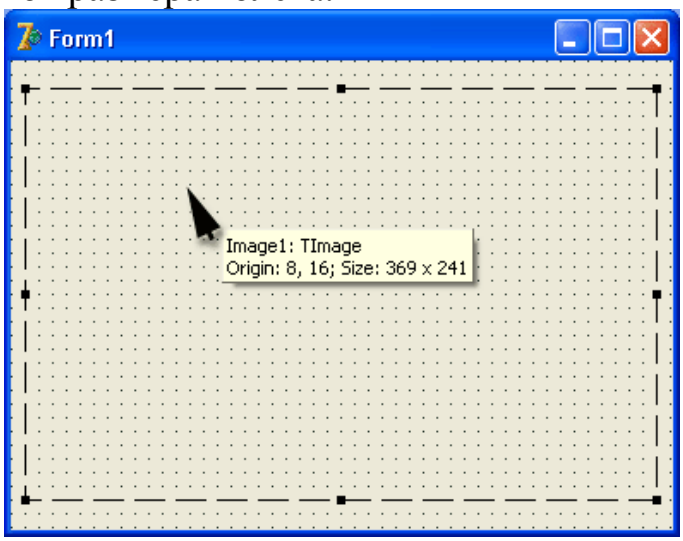


Рис. II.1. Координаты точек холста

Размер холста можно получить, обратившись к свойствам Height и width **области иллюстрации** (image) или к **свойствам формы:** ClientHeight и Clientwidth.

2.2. Карандаш и кисть

Художник в своей работе использует карандаши и кисти. Методы, обеспечивающие вычерчивание на поверхности холста графических примитивов, тоже используют карандаш и кисть. Карандаш применяется для вычерчивания линий и контуров, а кисть — для закрашивания областей, ограниченных контурами.

Карандашу и кисти, используемым для вывода графики на холсте, соответствуют свойства Pen (карандаш) и Brush (кисть), которые представляют собой объекты типа треп и TBrush, соответственно. Значения свойств этих объектов определяют вид выводимых графических элементов.

2.3. Карандаш

Карандаш используется для вычерчивания точек, линий, контуров геометрических фигур: прямоугольников, окружностей, эллипсов, дуг и др. Вид линии, которую оставляет карандаш на поверхности холста, определяют свойства объекта треп, которые перечислены в табл. II.1.

Таблица II.1. Свойства объекта треп (карандаш)

Свойство	**Определяет**
Color	Цвет линии
Width	Толщину линии
Style	Вид линии
Mode	Режим отображения

Свойство Color задает цвет линии, вычерчиваемой карандашом. В табл. II.2 перечислены именованные константы (тип TCoior), которые можно использовать в качестве значения свойства color.

Таблица II.2. Значение свойства Color определяет цвет линии

Константа	**Цвет**	**Константа**	**Цвет**
clBlack	Черный	clSilver	Серебристый
clMaroon	Каштановый	clRed	Красный
clGreen	Зеленый	clLime	Салатный
clOlive	Оливковый	clBlue	Синий
clNavy	Темно-синий	clFuchsia	Ярко-розовый
clPurple	Розовый	clAqua	Бирюзовый
clTeal	Зелено-голубой	clWhite	Белый
clGray	Серый		

Свойство width задает толщину линии (в пикселах). Например, инструкция Canvas. Pen. width: =2 устанавливает толщину линии в 2 пиксела.

Свойство style определяет вид (стиль) линии, которая может быть непрерывной или прерывистой, состоящей из штрихов различной длины. В табл. II.3 перечислены именованные константы, позволяющие задать стиль линии. Толщина пунктирной линии не может быть больше 1. Если значение свойства Pen.width больше единицы, то пунктирная линия будет выведена как сплошная.

Таблица II.3. Значение свойства Pen. type определяет вид линии

Константа	**Вид линии**
psSolid	Сплошная линия
psDash	Пунктирная линия, длинные штрихи

psDot	Пунктирная линия, короткие штрихи
psDashDot	Пунктирная линия, чередование длинного и короткого штрихов
psDashDotDot	Пунктирная линия, чередование одного длинного и двух коротких штрихов
psClear	Линия не отображается (используется, если не надо изображать границу области, например, прямоугольника)

Свойство Mode определяет, как будет формироваться цвет точек линии в зависимости от цвета точек холста, через которые эта линия прочерчивается. По умолчанию вся линия вычерчивается цветом, определяемым значением свойства Pen.Color.

Однако программист может задать инверсный цвет линии по отношению к цвету фона. Это гарантирует, что независимо от цвета фона все участки линии будут видны, даже в том случае, если цвет линии и цвет фона совпадают.

В табл. II.4 перечислены некоторые константы, которые можно использовать в качестве значения свойства Pen.Mode.

Таблица II.4. Значение свойства Pen. Mode влияет на цвет линии

Константа	**Цвет линии**
pmBlack	Черный, не зависит от значения свойства Pen. Color
pmWhite	Белый, не зависит от значения свойства Pen. Color
pmCopy	Цвет линии определяется значением свойства Pen . Color
pmNotCopy	Цвет линии является инверсным по отношению к значению свойства Pen. Color
pmNot	Цвет точки линии определяется как инверсный по отношению к цвету точки холста, в которую выводится точка линии

2.4. Кисть

Кисть (canvas.Brush) используется методами, обеспечивающими вычерчивание замкнутых областей, например геометрических фигур, для заливки (закрашивания) этих областей. Кисть, как объект, обладает двумя свойствами, перечисленными в табл. II.5.

Таблица II.5. Свойства объекта TBrush (кисть)

Свойство	**Определяет**

Color Цвет закрашивания замкнутой области
Style Стиль (тип) заполнения области

Область внутри контура может быть закрашена или заштрихована. В первом случае область полностью перекрывает фон, а во втором — сквозь незаштрихованные участки области будет виден фон.

В качестве значения свойства Color можно использовать любую из констант типа TColor (см. список констант для свойства Pen.color в табл. II.2).

Константы, позволяющие задать стиль заполнения области, приведены в табл. II.6.

Таблица II.6. Значения свойства Brush, style определяют тип закрашивания

Константа	**Тип заполнения (заливки) области**
bsSolid	Сплошная заливка
bsClear	Область не закрашивается
bsHorizontal	Горизонтальная штриховка
bsVertical	Вертикальная штриховка
bsFDiagonal	Диагональная штриховка с наклоном линий вперед
bsBDiagonal	Диагональная штриховка с наклоном линий назад
bsCross	Горизонтально-вертикальная штриховка, в клетку
bsDiagCross	Диагональная штриховка, в клетку

В качестве примера в листинге II.1 приведена программа **Стили заполнения областей,** которая в окно (рис. II.2) выводит восемь прямоугольников, закрашенных черным цветом с использованием разных стилей.

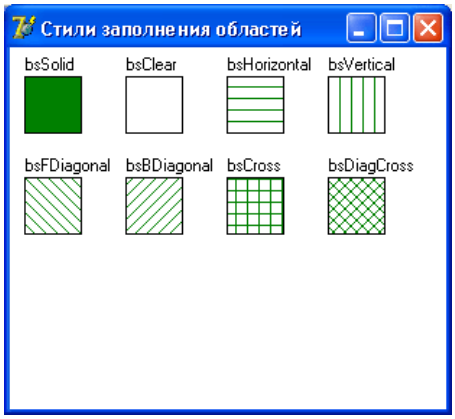


Рис. II.2. Окно программы **Стили заполнения областей**

Листинг II.1. Стили заполнения областей

unit brustyle_; **interface**

```
uses
Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs, ExtCtrls;
type
TForm1 = class(TForm)
procedure FormPaint(Sender: TObject);
private
{ Private declarations}
public
{ Public declarations )
end;
var
Form1: TForm1;
implementation
{$R *.DFM}
// перерисовка формы
procedure TForm1.FormPaint(Sender: TObject);
const
bsName: array[1..8] of string =
('bsSolid','bsClear','bsHorizontal',
'bsVertical','bsFDiagonal','bsBDiagonal',
'bsCross','bsDiagCross');
var
x,y: integer; // координаты левого верхнего угла
прямоугольника
w,h: integer; // ширина и высота прямоугольника
bs: TBrushStyle;// стиль заполнения области
k: integer; // номер стиля заполнения
i,j: integer;
begin
w:=40; h:=40; // размер области(прямоугольника)
y:=20;
for i:=l to 2 do
begin
x:=10;
for j:=1 to 4 do
begin
k:=j+(i-1)*4; // номер стиля заполнения
case k of
1: bs = bsSolid;
2: bs = bsClear;
3: bs = bsHorizontal;
```

```
4: bs = bsVertical;
5: bs = bsFDiagonal;
6: bs = bsBDiagonal;
7: bs = bsCross;
8: bs = bsDiagCross; end;
// вывод прямоугольника
Canvas.Brush.Color := clGreen;
// цвет закрашивания — зеленый
Canvas.Brush.Style := bs;
// стиль закрашивания
Canvas . Rectangle (x, y, x+w, y-t-h) ;
// вывод названия стиля
Canvas.Brush.Style := bsClear;
Canvas.TextOut(x, y-15, bsName[k]);
// вывод названия стиля
x := x+w+30;
end;
y := y+h+30;
end;
end.
```

2.5. Вывод текста

Для вывода текста на поверхность графического объекта используется метод TextOut. Инструкция вызова метода TextOut в общем виде выглядит следующим образом:

Объект.Canvas.TextOut(x, y, Текст)

где:

- объект — имя объекта, на поверхность которого выводится текст;
- x, y — координаты точки графической поверхности, от которой выполняется вывод текста (рис. II.3);
- Текст — переменная или константа символьного типа, значение которой определяет выводимый методом текст.

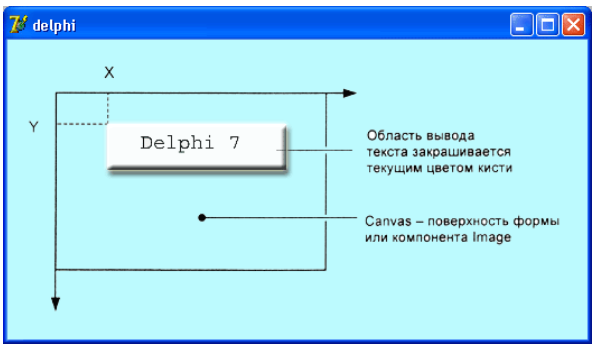


Рис. II.3. Координаты области вывода текста

Шрифт, который используется для вывода текста, определяется значением свойства Font соответствующего объекта canvas. Свойство Font представляет собой объект типа TFont. В табл. II.7 перечислены свойства объекта TFont, позволяющие задать характеристики шрифта, используемого методами TextOut и TextRect для вывода текста.

Таблица II.7. Свойства объекта TFont

Свойство	**Определяет**
Name	Используемый шрифт. В качестве значения следует использовать название шрифта, например Arial
Size	Размер шрифта в пунктах (points). Пункт— это единица измерения размера шрифта, используемая в полиграфии. Один пункт равен 1/72 дюйма
Style	Стиль начертания символов. Может быть: нормальным, полужирным, курсивным, подчеркнутым, перечеркнутым. Стиль задается при помощи следующих констант: fsBold (полужирный), fsltalic (курсив), f sUnderline (подчеркнутый), f sStrikeOut (перечеркнутый).

Свойство	**Определяет**
style	Свойство style является множеством, что позволяет комбинировать необходимые стили. Например, инструкция программы, устанавливающая стиль "полужирный курсив", выглядит так:
Color	Объект. Canvas . Font : = [fsBold, fs Italic]
Цвет символов.	В качестве значения можно использовать константу типа Tcolor

Внимание!

Область вывода текста закрашивается текущим цветом кисти. Поэтому перед выводом текста свойству Brush.Color нужно присвоить значение bsClear или задать цвет кисти, совпадающий с цветом поверхности, на которую выводится текст.

Следующий фрагмент программы демонстрирует использование функции Textout для вывода текста на поверхность формы:

```
with Form1.Canvas do begin
// установить характеристики шрифта
Font.Name := 'Tahoma';
Font.Size := 20;
Font.Style := [fsltalic, fsBold] ;
Brush.Style := bsClear; // область вывода текста не закраши-
TextOut(0, 10, 'Borland Delphi 7');
end;
```

После вывода текста методом Textout указатель вывода (карандаш) перемещается в правый верхний угол области вывода текста.

Иногда требуется вывести какой-либо текст после сообщения, длина которого во время разработки программы неизвестна. Например, это может быть слово "руб." после значения числа, записанного прописью. В этом случае необходимо знать координаты правой границы уже выведенного текста. Координаты правой границы текста, выведенного методом Textout, можно получить, обратившись к свойству PenPos.

Следующий фрагмент программы демонстрирует возможность вывода строки текста при помощи двух инструкций Textout.

```
with Form1.Canvas do begin
TextOut(0, 10, 'Borland ') ;
TextOut(PenPos.X, PenPos.Y, 'Delphi 7');
end;
```

2.6. Методы вычерчивания графических примитивов

Любая картинка, чертеж, схема могут рассматриваться как совокупность графических примитивов: точек, линий, окружностей, дуг и др. Таким образом, для того чтобы на экране появилась нужная картинка, программа должна обеспечить

вычерчивание (вывод) графических примитивов, составляющих эту картинку.

Вычерчивание графических примитивов на поверхности компонента (формы или области вывода иллюстрации) осуществляется применением соответствующих методов к свойству Canvas этого компонента.

2.7. Линия

Вычерчивание прямой линии осуществляет метод LineTo, инструкция вызова которого в общем виде выглядит следующим образом:

Компонент.Canvas.LineTo(x,y)

Метод LineTo вычерчивает прямую линию от текущей позиции карандаша в точку с координатами, указанными при вызове метода.

Начальную точку линии можно задать, переместив карандаш в нужную точку графической поверхности. Сделать это можно при помощи метода MoveTo, указав в качестве параметров координаты нового положения карандаша.

Вид линии (цвет, толщина и стиль) определяется значениями свойств объекта Реп графической поверхности, на которой вычерчивается линия.

Довольно часто результаты расчетов удобно представить в виде графика. Для большей информативности и наглядности графики изображают на фоне координатных осей и оцифрованной сетки. В листинге II.2 приведен текст программы, которая на поверхность формы выводит координатные оси и оцифрованную сетку (рис. II.4).

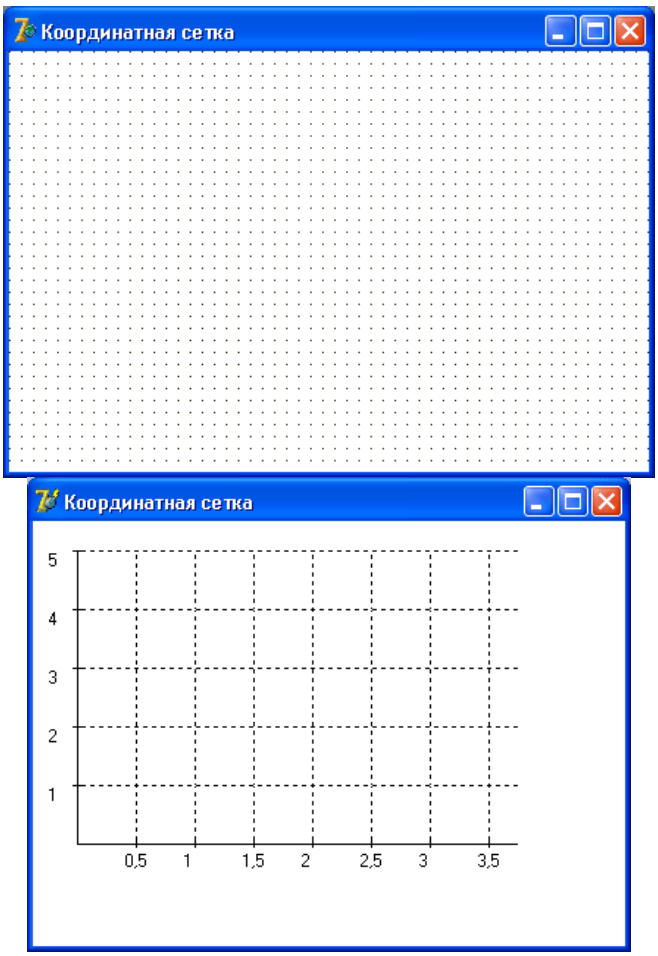


Рис. II.4. Форма приложения **Координатная сетка**

Листинг II.2. Оси координат и оцифрованная сетка

```
unit grid_;
interface
uses
Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls;
type
TForm1 = class(TForm)
procedure FormPaint(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1; implementation
{$R *.DFM}
procedure TForm1.FormPaint(Sender: TObject);
var
```

```
x0,y0:integer; // координаты начала координатных осей
dx,dy:integer; // шаг координатной сетки (в пикселах)
h,w:integer; // высота и ширина области вывода координатной
сетки
x,y:integer;
lx,ly:real; // метки (оцифровка) линий сетки по X и Y
dlx,dly:real; // шаг меток (оцифровки) линий сетки по X и Y
cross:integer; // счетчик неоцифрованных линий сетки
dcross:integer;// количество неоцифрованных линий между
оцифрованными
begin
x0:=30; y0:=220; // оси начинаются в точке (40,250)
dx:=40; dy:=40; // шар координатной сетки 40 пикселов
dcross:=1; // помечать линии сетки X: 1 — каждую;
// 2 — через одну;
// 3 — через две;
dlx:=0.5; // шаг меток оси X
dly:=1.0; // шаг меток оси Y, метками будут: 1, 2, 3 и т. д.
h:=200; w:=300;
with forml.Canvas do begin
cross:=dcross;
MoveTo(x0,v0); LineTo(x0,y0-h); // ось X
MoveTo(x0,y0); LineTo(x0+w, y0); // ось Y
// засечки, сетка и оцифровка по оси X
x:=x0+dx;
lx:=dlx;
repeat
MoveTo(x,y0-3);LineTo(x,yO+3); // засечка
cross:=cross-l;
if cross = 0 then // оцифровка
begin
TextOut(x-8,y0+5,FloatToStr(lx));
cross:=dcross ; end;
Pen.Style:=psDot;
MoveTo(x,y0-3);LineTo(x,y0-h); // линия сетки
Pen.Style:=psSolid;
lx:=lx+dlx;
x:=x+dx;
until (x>x0+w);
// засечки, сетка и оцифровка по оси Y
y:=y0-dy;
ly:=dly;
```

```
repeat
MoveTo(x0-3,y);LineTo(x0+3,y); // засечка
TextOut(x0-20,y,FloatToStr(1y)); // оцифровка
Pen.Style:=psDot;
MoveTo(x0+3,y); LineTo(x0+w,y); // линия сетки
Pen.Style:=psSolid;
y:=y-dy;
ly:=ly+dly; until (y<y0-h);
end;
end;
end.
```

Особенность приведенной программы заключается в том, что она позволяет задавать шаг сетки и оцифровку. Кроме того, программа дает возможность оцифровывать не каждую линию сетки оси x, а через одну, две, три и т. д. Сделано это для того, чтобы предотвратить возможные наложения изображений чисел оцифровки друг на друга в случае, если эти числа состоят из нескольких цифр.

2.8. Ломаная линия

Метод polyline вычерчивает ломаную линию. В качестве параметра метод получает массив типа TPoint. Каждый элемент массива представляет собой запись, поля x и y которой содержат координаты точки перегиба ломаной. Метод Polyline вычерчивает ломаную линию, последовательно соединяя прямыми точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д.

В качестве примера использования метода Polyline в листинге II.3 приведена процедура, которая выводит график изменения некоторой величины. Предполагается, что исходные данные находятся в доступном процедуре массиве Data (тип Integer).

Листинг II.3. График функции (использование метода Polyline)

```
procedure TForml.Button1Click(Sender: TObject);
var
gr: array[1..50] of TPoint; // график — ломаная линия
x0,y0: integer; // координаты точки начала координат
dx,dy: integer; // шаг координатной сетки по осям X и Y
i: integer; begin
x0 := 10; y0 := 200; dx :=5; dy := 5;
```

```
// заполним массив gr
for i:=l to 50 do begin
gr[i].x := x0 + (i-l)*dx;
gr[i].y := y0 - Data[i]*dy;
end;
// строим график
with forml.Canvas do begin
MoveTo(x0,y0); LineTo(x0,10); // ось Y
MoveTo(x0,y0); LineTo(200,y0); // ось X
Polyline(gr); // график
end;
end;
```

Метод Polyline можно использовать для вычерчивания замкнутых контуров. Для этого надо, чтобы первый и последний элементы массива содержали координаты одной и той же точки. В качестве примера использования метода Polybine для вычерчивания замкнутого контура в листинге II.4 приведена программа, которая на поверхности диалогового окна, в точке нажатия кнопки мыши, вычерчивает контур пятиконечной звезды (рис. II.5). Цвет, которым вычерчивается звезда, зависит от того, какая из кнопок мыши была нажата. Процедура обработки нажатия кнопки мыши (событие MouseDown) вызывает процедуру рисования звезды starLine и передает ей в качестве параметра координаты точки, в которой была нажата кнопка. Звезду вычерчивает процедура starLine, которая в качестве параметров получает координаты центра звезды и холст, на котором звезда должна быть выведена. Сначала вычисляются координаты концов и впадин звезды, которые записываются в массив p. Затем этот массив передается в качестве параметра методу Polyline. При вычислении координат лучей и впадин звезды используются функции sin и cos. Так как аргумент этих функций должен быть выражен в радианах, то значение угла в градусах домножается на величину pi/18o, где pi — это стандартная именованная константа равная числу π.

Листинг II.4. Вычерчивание замкнутого контура (звезды) в точке нажатия кнопки мыши

```
unit Stars_; interface
uses
Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls;
```

```
type
TForm1 = class(TForm)
procedure FormMouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Forml: TForml;
implementation
f$R *.dfm}
// вычерчивает звезду
procedure StarLine(x0,y0,r: integer; Canvas: TCanvas);
// x0,y0 — координаты центра звезды
//r — радиус заезды var
p : array [1.. 11] of TPoint;
// массив координат лучей и впадин
a: integer; // угол между осью OX и прямой, соединяющей
// центр звезды и конец луча или впадину i: integer;
begin
a := 18; // строим от правого гор. луча
for i:=l to 10 do begin
if (i mod 2=0) then begin // впадина
p[i].x := x0+Round(r/2*cos(a*pi/180) ) ;
p[i] .y:=y0-Round(r/2*sin(a*pi/180) ) ;
end
else
begin // луч
[i] .x:=x0+Round(r*cos (a*pi/180) ) ;
[i] .y:=y0-Round(r*sin(a*pi/180) ) ;
end;
a := a+36;
end;
p[ll].X := p[l].X; // чтобы замкнуть контур звезды
Canvas. Polyline (p) ; // начертить звезду
end;
// нажатие кнопки мыши
procedure TForm1 . FormMouseDown { Sender : TObject;
Button: TMouseButton;
```

```
Shift: TShiftState; X, Y: Integer);
begin
if Button = mbLeft // нажата левая кнопка?
then Form1. Canvas . Pen . Color : = clRed
else Form1. Canvas. Pen. Color := clGreen;
StarLine(x, y, 30, Forml. Canvas );
end;
end.
```

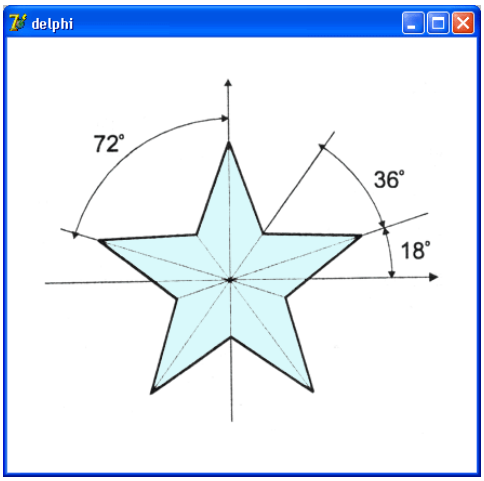


Рис.II.5. Звезда

Примечание
Обратите внимание, что размер массива p на единицу больше, чем количество концов и впадин звезды, и что значения первого и последнего элементов массива совпадают.

2.9. Окружность и эллипс

Метод Ellipse вычерчивает эллипс или окружность, в зависимости от значений параметров. Инструкция вызова метода в общем виде выглядит следующим образом:
Объект.Canvas.Ellipse(x1,y1, x2,y2]
где:

- объект — имя объекта (компонента), на поверхности которого выполняется вычерчивание;
- x1, y1, x2, y2 — координаты прямоугольника, внутри которого вычерчивается эллипс или, если прямоугольник является квадратом, окружность (рис. II.6).

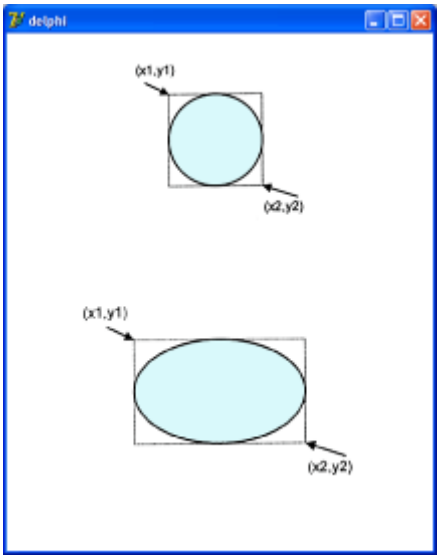


Рис. II.6. Значения параметров метода Ellipse определяют вид геометрической фигуры

Цвет, толщина и стиль линии эллипса определяются значениями свойства Реп, а цвет и стиль заливки области внутри эллипса — значениями свойства Brush поверхности (canvas), на которую выполняется вывод.

2.10. Дуга

Вычерчивание дуги выполняет метод Arc, инструкция вызова которого в общем виде выглядит следующим образом:
Объект.Canvas.Arc(x1,y1,x2,y2,x3,y3,x4,y4)
где:

- x1, y1, x2, y2 — параметры, определяющие эллипс (окружность), частью которого является вычерчиваемая дуга;
- x3, y3 — параметры, определяющие начальную точку дуги; П x4, y4 — параметры, определяющие конечную точку дуги.

Начальная (конечная) точка — это точка пересечения границы эллипса и прямой, проведенной из центра эллипса в точку с координатами x3 и y3 (x4, y4). Дуга вычерчивается против часовой стрелки от начальной точки к конечной (рис. II.7).

Цвет, толщина и стиль линии, которой вычерчивается дуга, определяются значениями свойства Реп поверхности (canvas), на которую выполняется вывод.

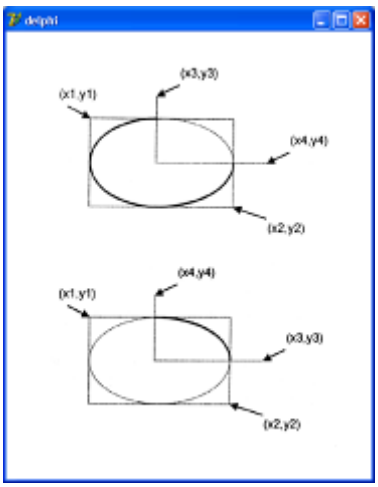


Рис. II.7. Значения параметров метода Arc определяют дугу как часть эллипса (окружности)

2.11. Прямоугольник

Прямоугольник вычерчивается методом Rectangle, инструкция вызова которого в общем виде выглядит следующим образом:

Объект.Canvas.Rectangle(x1, y1,x2, y2)

где:

- объект — имя объекта (компонента), на поверхности которого выполняется вычерчивание;
- x1, y1 и x2, y2 — координаты левого верхнего и правого нижнего углов прямоугольника.

Метод RoundRec тоже вычерчивает прямоугольник, но со скругленными углами. Инструкция вызова метода RoundRec выглядит так:

Объект.Canvas.RoundRec(x1,y1,x2, y2, x3, y3)

где:

- x1, y1, x2, y2 -- параметры, определяющие положение углов прямоугольника, в который вписывается прямоугольник со скругленными углами;
- x3 и y3 — размер эллипса, одна четверть которого используется для вычерчивания скругленного угла (рис. II.8).

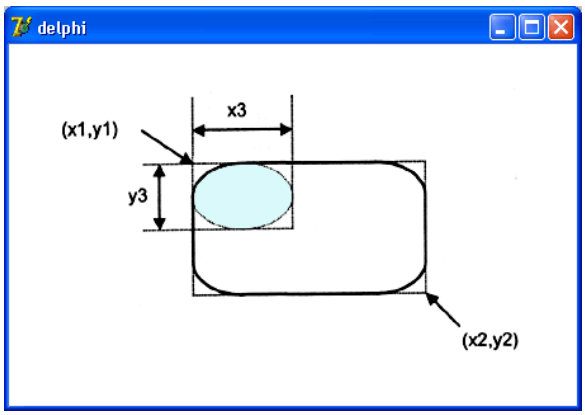


Рис. II.8. Метод RoundRec вычерчивает прямоугольник со скругленными углами

Вид линии контура (цвет, ширина и стиль) определяется значениями свойства Реп, а цвет и стиль заливки области внутри прямоугольника — значениями свойства Brush поверхности (canvas), на которой прямоугольник вычерчивается.

Есть еще два метода, которые вычерчивают прямоугольник, используя в качестве инструмента только кисть (Brush). Метод FillRect вычерчивает закрашенный прямоугольник, а метод FrameRect — только контур. У каждого из этих методов лишь один параметр — структура типа TRect. Поля структуры TRect содержат координаты прямоугольной области, они могут быть заполнены при помощи функции Rect.

Ниже в качестве примера использования методов FillRect и FrameRect приведена процедура, которая на поверхности формы вычерчивает прямоугольник с красной заливкой и прямоугольник с зеленым контуром.

```
procedure TForm1.Button1Click(Sender: TObject);
var
r1, r2: TRect; // координаты углов прямоугольников
begin
// заполнение полей структуры
// зададим координаты углов прямоугольников
r1 := Rect(20,20,60,40);
r2 := Rect(10,10,40,50);
with form1.Canvas do begin
Brush.Color := clRed;
FillRect(r1); // закрашенный прямоугольник
Brush.Color := clGreen;
FrameRect(r2}; // только граница прямоугольника
end;
```

```
end;
```

2.12. Многоугольник

Метод Polygon вычерчивает многоугольник. В качестве параметра метод получает массив типа TPoint. Каждый элемент массива представляет собой запись, поля (x,y) которой содержат координаты одной вершины многоугольника. Метод Polygon вычерчивает многоугольник, последовательно соединяя прямыми линиями точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д. Затем соединяются последняя и первая точки.

Цвет и стиль границы многоугольника определяются значениями свойства Реп, а цвет и стиль заливки области, ограниченной линией границы, — значениями свойства Brush, причем область закрашивается с использованием текущего цвета и стиля кисти.

Ниже приведена процедура, которая, используя метод polygon, вычерчивает треугольник:

```
procedure TForm1.Button2Click(Sender: TObject);
var
pol: array[1..3] of TPoint; // координаты точек треугольника
begin
pol[1].x := 10;
polf1].y := 50;
pol[2].x := 40;
pol[2].y := 10;
pol[3].x := 70;
pol[3].y := 50;
Form1.Canvas.Polygon(pol);
end;
```

2.13. Сектор

Метод pie вычерчивает сектор эллипса или круга. Инструкция вызова метода в общем виде выглядит следующим образом:
Объект. Canvas.Pie(x1,y1,x2,y2,x3,y3,x4,y4)
где:

- x1, y1, x2, y2 — параметры, определяющие эллипс (окружность), частью которого является сектор;

- x3, y3, x4, y4 — параметры, определяющие координаты конечных точек прямых, являющихся границами сектора.

Начальные точки прямых совпадают с центром эллипса (окружности). Сектор вырезается против часовой стрелки от прямой, заданной точкой с координатами (x3, уз), к прямой, заданной точкой с координатами (x4, y4) (рис. II.9).

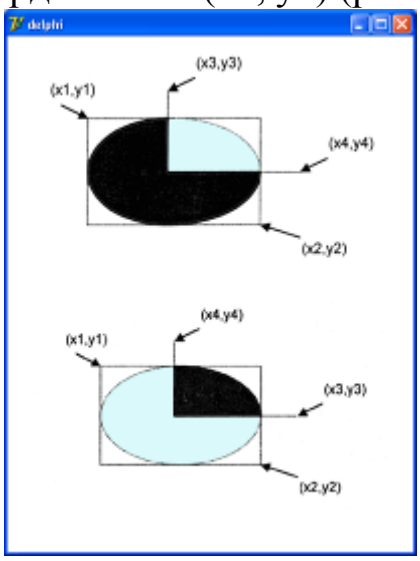


Рис. II.9. Значения параметров метода Pie определяют сектор как часть эллипса (окружности)

2.14. Точка

Поверхности, на которую программа может осуществлять вывод графики, соответствует объект Canvas. Свойство pixels, представляющее собой двумерный массив типа TColor, содержит информацию о цвете каждой точки графической поверхности. Используя свойство Pixels, можно задать требуемый цвет для любой точки графической поверхности, т. е. "нарисовать" точку. Например, инструкция

```
Form1.Canvas.Pixels[10,10]:=clRed
```

окрашивает точку поверхности формы в красный цвет.

Размерность массива pixels определяется размером графической поверхности. Размер графической поверхности формы (рабочей области, которую также называют клиентской) задается значениями свойств ciientwidth и ClientHeight, а размер графической поверхности компонента image — значениями свойств width и Height. Левой верхней точке рабочей области формы соответствует элемент pixels [0,0], а правой нижней - Pixels[Ciientwidth - 1,ClientHeight - 1].

Свойство Pixels можно использовать для построения графиков. График строится, как правило, на основе вычислений по

формуле. Границы диапазона изменения аргумента функции являются исходными данными. Диапазон изменения значения функции может быть вычислен. На основании этих данных можно вычислить масштаб, позволяющий построить график таким образом, чтобы он занимал всю область формы, предназначенную для вывода графика.

Например, если некоторая функция f(x) может принимать значения от нуля до 1000, и для вывода ее графика используется область формы высотой в 250 пикселов, то масштаб оси Y вычисляется по формуле: т = 250/1000. Таким образом, значению f(x) = 70 будет соответствовать точка с координатой Y =233. Значение координаты Y вычислено по формуле

Y= h -f(x) x т = 250 - 70x(250/1000),

где h - высота области построения графика.

Обратите внимание на то, что точное значение выражения 250 - 70x(250/1000) равно 232,5. Но т. к. индексом свойства pixels, которое используется для вывода точки на поверхность Canvas, может быть только целое значение, то число 232,5 округляется к ближайшему целому, которым является число 233.

Следующая программа, текст которой приведен в листинге II.5, используя свойство pixels, выводит график функции y = 2 sin(jc) e*/[5]. Для построения графика используется вся доступная область формы, причем если во время работы программы пользователь изменит размер окна, то график будет выведен заново с учетом реальных размеров окна.

Листинг II.5. График функции

```
unit grfunc_;
interface
Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs;
type
TForm1 = class(TForm)
procedure FormPaint(Sender: TObject);
procedure FormResize(Sender: TObject);
private
{ Private declarations }
public
{Public declarations }
end;
var
Forml: TForml;
```

```
implementation
{$R *.DFM}
// Функция, график которой надо построить
Function f(x:real):real;
begin
f:=2*Sin(x)*exp(x/5) ;
end;
// строит график функции
procedure GrOfFunc;
var
x1,x2:real; // границы изменения аргумента функции
y1,y2:real; // границы изменения значения функции
x:real; // аргумент функции
y:real; // значение функции в точке x
dx:real; // приращение аргумента
l,b:integer; // левый нижний угол области вывода графика
w,h:integer; // ширина и высота области вывода графика
mx,my:real; // масштаб по осям X и Y
x0,y0:integer; // точка — начало координат
begin
// область вывода графика
l:=10; // X — координата левого верхнего угла
b:=Forml.ClientHeight-20;
//У — координата левого верхнего угла
h:=Forml.ClientHeight-40; // высота
w:=Forml.Width-40; // ширина
x1:=0; // нижняя граница диапазона аргумента
x2:=25; // верхняя граница диапазона аргумента
dx:=0.01; // шаг аргумента
// найдем максимальное и минимальное значения
// функции на отрезке [x1,x2]
y1:=f(xl); // минимум
y2:=f(xl); //максимум
x:=x1;
repeat
У := f (x);
if y < yl then yl:=y;
if y > y2 then y2:=y;
x:=x+dx; until (x >= x2);
// вычислим масштаб
my:=h/abs(y2-yl); // масштаб по оси Y
mx:=w/abs(x2-xl); // масштаб по оси X
```

```
x0:=1;
y0:=b-Abs(Round(y1*my)) ;
with form1.Canvas do
begin
// оси
MoveTo(l,b);LineTo(l,b-h);
MoveTo(x0,y0);LineTo(x0+w,y0);
TextOut(l+5,b-h,FloatToStrF(y2,ffGeneral,6,3));
TextOut(l+5,b,FloatToStrF(yl,ffGeneral,6,3));
// построение графика
x:=xl; repeat
y:=f(x);
Pixels[x0+Round(x*mx),y0-Round(y*my)]:=clRed;
x:=x+dx;
until (x >= x2);
end;
end;
procedure TForm1.FormPaint(Sender: TObject);
begin
GrOfFunc; end;
// изменился размер окна программы
procedure TForm1.FormResize(Sender: TObject);
begin
// очистить форму
forml.Canvas.FillRect(Rect(0,0,ClientWidth,
ClientHeight));
// построить график
GrOfFunc;
end;
end.
```

Основную работу выполняет процедура GrOfFunc, которая сначала вычисляет максимальное (y2) и минимальное (yl) значения функции на отрезке [x1l,x2]. Затем, используя информацию о ширине (Forml.Clientwidth -40) и высоте (Form1.ClientHeight - 40) области вывода графика, вычисляет масштаб по осям X (mx) иY(my).

Высота и ширина области вывода графика определяется размерами рабочей (клиентской) области формы, т. е. без учета области заголовка и границ. После вычисления масштаба процедура вычисляет координату у горизонтальной оси (yo) и

вычерчивает координатные оси графика. Затем выполняется непосредственное построение графика (рис. II.10).

Вызов процедуры GrOfFunc выполняют процедуры обработки событий onPaint и onFormResize. Процедура TForm1. FormPaint обеспечивает вычерчивание графика после появления формы на экране в результате запуска программы, а также после появления формы во время работы программы, например, в результате удаления или перемещения других окон, полностью или частично перекрывающих окно программы. Процедура TForm1.FormResize обеспечивает вычерчивание графика после изменения размера формы.

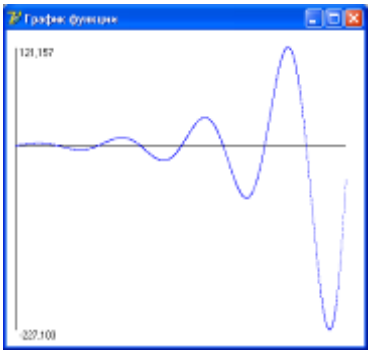


Рис. II.10. График, построенный процедурой GrOfFunc

Приведенная программа довольно универсальна. Заменив инструкции в теле функции f (x), можно получить график другой функции. Причем независимо от вида функции ее график будет занимать всю область, предназначенную для вывода.

Примечание

Рассмотренная программа работает корректно, если функция, график которой надо построить, принимает как положительные, так и отрицательные значения. Если функция во всем диапазоне только положительная или только отрицательная, то в программу следует внести изменения. Какие — пусть это будет упражнением для читателя.

2.15. Вывод иллюстраций

Наиболее просто вывести иллюстрацию, которая находится в файле с расширением bmp, jpg или ico, можно при помощи компонента image, значок которого находится на вкладке **Additional** палитры (рис. II.III).

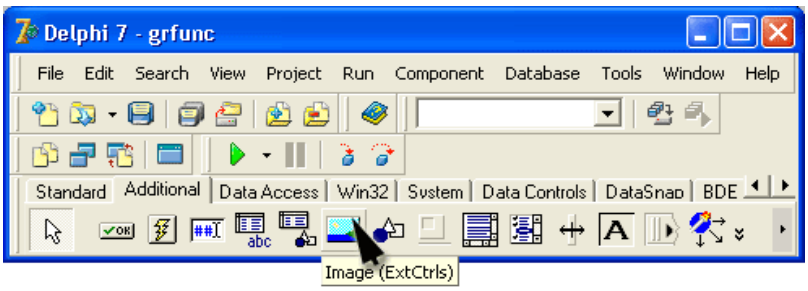


Рис. II.11. Значок компонента Image

В табл. II.8 перечислены основные свойства компонента image.

Таблица II.8. Свойства компонента image

Свойство	**Определяет**
Picture	Иллюстрацию, которая отображается в поле компонента
Width, Height	Размер компонента. Если размер компонента меньше размера иллюстрации, и значение свойств AutoSize и strech равно False, то отображается часть иллюстрации
AutoSize	Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации
Strech	Признак автоматического масштабирования иллюстрации в соответствии с реальным размером компонента. Чтобы было выполнено масштабирование, значение свойства AutoSize должно быть False
Visible	Отображается ли компонент, и, соответственно, иллюстрация, на поверхности формы

Иллюстрацию, которая будет выведена в поле компонента image, можно задать как во время разработки формы приложения, так и во время работы программы.

Во время разработки формы иллюстрация задается установкой значения свойства picture путем выбора файла иллюстрации в стандартном диалоговом окне, которое появляется в результате щелчка на командной кнопке **Load** окна **Picture Editor** (рис. II.12). Чтобы запустить Image Editor, нужно в окне **Object Inspector** выбрать свойство Picture и щелкнуть на кнопке с тремя точками.

Если размер иллюстрации больше размера компонента, то свойству strech нужно присвоить значение True и установить значения свойств width и Height пропорционально реальным размерам иллюстрации.

Чтобы вывести иллюстрацию в поле компонента image во время работы программы, нужно применить метод LoadFromFile к свойству Picture, указав в качестве параметра имя файла иллюстрации. Например, инструкция

Form1.Image1.Picture.LoadFromFile('e:\temp\bart.bmp')

загружает иллюстрацию из файла bart.bmp и выводит ее в поле вывода иллюстрации (imagel).

Метод LoadFromFile позволяет отображать иллюстрации различных графических форматов: BMP, WMF, JPEG (файлы с расширением jpg).

Следующая программа, ее текст приведен в листинге II.6, использует компонент image для просмотра иллюстраций, которые находятся в указанном пользователем каталоге. Диалоговое окно программы приведено на рис. II.13.

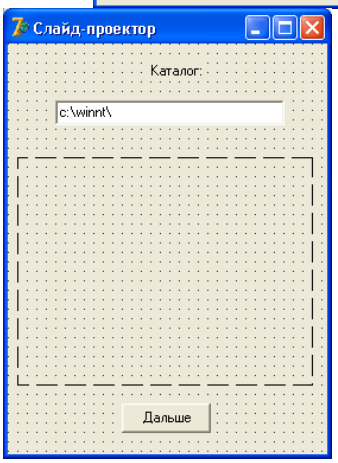


Рис. II.12. Окно **Picture Editor** **Рис. II.13.** Слайд-проектор

Листинг II.6. Слайд-проектор

unit shpic_;

```
interface
uses
Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms,
Dialogs, ExtCtrls, StdCtrls, Menu
type
TForm1 = class(TForm) Image1: TImage;
Button1: TButton;
procedure FormActivate(Sender: TObject);
procedure ButtonlClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
aSearchRec : TSearchRec;
aPath : String; // каталог, в котором находятся иллюстрации
aFile : String; // файл иллюстрации
iw,ih: integer; // первоначальный размер компонента Image
implementation
$R *.DFM}
// изменение размера области вывода иллюстрации
// пропорционально размеру иллюстрации
Procedure Scalelmage;
var
pw, ph : integer; // размер иллюстрации
scaleX, scaleY : real; // масштаб по X и Y
scale : real; // общий масштаб
begin
// иллюстрация уже загружена
// получим ее размеры
pw := Form1.Image1.Picture.Width;
ph := Form1.Image1.Picture.Height;
if pw > iw // ширина иллюстрации больше ширины
компонента Image
then scaleX := iw/pw // нужно масштабировать
else scaleX := 1;
if ph > ih // высота иллюстрации больше высоты компонента
then scaleY := ih/ph // нужно масштабировать
else scaleY := 1;
```

```
// выберем наименьший коэффициент
if scaleX < scaleY
then scale := scaleX
else scale := scaleY;
// изменим размер области вывода иллюстрации
Form1.Image1.Height :=
Round(Form1.Image1.Picture.Height*scale)
Form1.Image1.Width :=
Round(Form1.Image1.Picture.Width*scale);
// т. к. Strech = True и размер области пропорционален
// размеру картинки, то картинка масштабируется без
искажений
end;
// вывести первую иллюстрацию
procedure FirstPicture;
var
r : integer; // результат поиска файла
begin
aPath := 'f:\temp\';
r := FindFirst(aPath+'*.bmp',faAnyFile,aSearchRec);
if r = 0 then
begin // в указанном каталоге есть bmp-файл
aFile := aPath + aSearchRec.Name;
Form1.Image1.Picture.LoadFromFile(aFile); // загрузить
// иллюстрацию
ScaleImage; //-установить размер компонента
Image r := FindNext(aSearchRec); // найти следующий файл
if r = 0 then // еще есть файлы иллюстраций
Forml.Button1.Enabled := True;
end;
end;
// вывести следующую иллюстрацию
Procedure NextPicture();
var
r : integer;
begin
aFile := aPath + aSearchRec.Name;
Forml.Image1.Picture.LoadFromFile(aFile);
ScaleImage;
// подготовим вывод следующей иллюстрации
r := FindNext(aSearchRec); // найти следующий файл
if r<>0
```

```
then // больше нет иллюстраций
Forml.Buttonl.Enabled := False;
end;
procedure TForml.FormActivate(Sender: TObject);
begin
Image1.AutoSize := False; // запрет автоизменения размера
компонента
Image1.Stretch := True; // разрешим масштабирование
// запомним первоначальный размер области вывода
иллюстрации
iw := Imagel.Width;
in := imagel.Height;
Button1.Enabled := False; // сделаем недоступной кнопку
Дальше
FirstPicture; // вывести первую иллюстрацию
end;
//щелчок на кнопке Дальше
procedure TForm1.Button1Click(Sender: TObject);
begin
NextPicture;
end;
end.
```

Программа выполняет масштабирование выводимых иллюстраций без искажения, чего нельзя добиться простым присвоением значения True свойству strech. Загрузку и вывод первой и остальных иллюстраций выполняют соответственно процедуры FirstPicture и NextPicture. Процедура FrirstPicture использует функцию FindFirst для того, чтобы получить имя первого BMP-файла. В качестве параметров функции FindFirst передаются:

- имя каталога, в котором должны находиться иллюстрации;
- структура asearchRec, поле Name которой, в случае успеха, будет содержать имя файла, удовлетворяющего критерию поиска;
- маска файла иллюстрации.

Если в указанном при вызове функции FindFirst каталоге есть хотя бы один BMP-файл, значение функции будет равно нулю. В этом случае метод LoadFromFiie загружает файл иллюстрации, после чего вызывается функция scaieimage, которая устанавливает размер компонента пропорционально размеру иллюстрации. Размер загруженной иллюстрации можно получить, обратившись к

свойствам Form1.Image1.Picture.Width и Form1.Шmage1.Picture.Height, значения которых не зависят от размера компонента Image.

2.16. Битовые образы

При работе с графикой удобно использовать объекты типа TBitMap (битовый образ). Битовый образ представляет собой находящуюся в памяти компьютера, и, следовательно, невидимую графическую поверхность, на которой программа может сформировать изображение. Содержимое битового образа (картинка) легко и, что особенно важно, быстро может быть выведено на поверхность формы или области вывода иллюстрации (image). Поэтому в программах битовые образы обычно используются для хранения небольших изображений, например, картинок командных кнопок.

Загрузить в битовый образ нужную картинку можно при помощи метода LoadFromFlie, указав в качестве параметра имя BMP-файла, в котором находится нужная иллюстрация.

Например, если в программе объявлена переменная pic типа TBitMap, то после выполнения инструкции

pic.LoadFromFiie('e:\images\aplane.bmp')

битовый образ pic будет содержать изображение самолета.

Вывести содержимое битового образа (картинку) на поверхность формы или области вывода иллюстрации можно путем применения метода Draw к соответствующему свойству поверхности (canvas). Например, инструкция

Image1.Canvas.Draw(x,y, bm)

выводит картинку битового образа bm на поверхность компонента image 1 (параметры x и y определяют положение левого верхнего угла картинки на поверхности компонента).

Если перед применением метода Draw свойству Transparent объекта TBitMap присвоить значение True, то фрагменты рисунка, окрашенные цветом, совпадающим с цветом левого нижнего угла картинки, не будут выведены — через них будет как бы проглядывать фон. Если в качестве "прозрачного" нужно использовать цвет, отличный от цвета левой нижней точки рисунка, то свойству Transparentcoior следует присвоить значение символьной константы, обозначающей необходимый цвет.

Следующая программа, текст которой приведен в листинге II.7, демонстрирует использование битовых образов для формирования изображения из нескольких элементов.

Листинг II.7. Использование битовых образов

```
unit aplanes_; interface
uses
Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs;
type
TForml = class(TForm)
procedure FormPaint(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Forml: TForm1;
sky,aplane: TBitMap; // битовые образы: небо и самолет
implementation
($R *.DFM}
procedure TForm1.FormPaint(Sender: TObject);
begin
// создать битовые образы
sky := TBitMap.Create;
aplane := TBitMap.Create;
// загрузить картинки
sky.LoadFromFile('sky.bmp');
aplane.LoadFromFile('aplane.bmp') ;
Form1.Canvas.Draw(0,0,sky); // отрисовка фона
Form1.Canvas.Draw(20,20,aplane); // отрисовка левого
самолета
aplane.Transparent:=True;
// теперь элементы рисунка, цвет которых совпадает с цветом
// левой нижней точки битового образа, не отрисовываются
Form1.Canvas.Draw(120,20,aplane);
// отрисовка правого самолета
// освободить память sky.free; aplane.free;
end;
end.
```

После запуска программы в окне приложения (рис. II.14) появляется изображение летящих на фоне неба самолетов. Фон и изображение самолета -битовые образы, загружаемые из файлов.

Белое поле вокруг левого самолета показывает истинный размер картинки битового образа aplane. Белое поле вокруг правого самолета отсутствует, т. к. перед его выводом свойству Transparent битового образа было присвоено значение True.

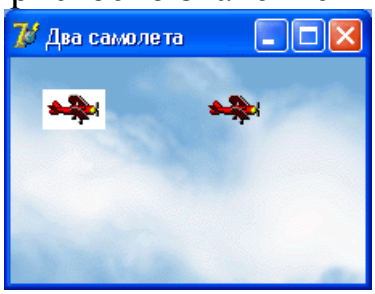


Рис. II.14. Влияние значение свойства **Transparent** на вывод изображения

2.17. Мультипликация

Под мультипликацией обычно понимается движущийся и меняющийся рисунок. В простейшем случае рисунок может только двигаться или только меняться.

Как было показано выше, рисунок может быть сформирован из графических примитивов (линий, окружностей, дуг, многоугольников и т. д.). Обеспечить перемещение рисунка довольно просто: надо сначала вывести рисунок на экран, затем через некоторое время стереть его и снова вывести этот же рисунок, но уже на некотором расстоянии от его первоначального положения. Подбором времени между выводом и удалением рисунка, а также расстояния между старым и новым положением рисунка (шага перемещения), можно добиться того, что у наблюдателя будет складываться впечатление, что рисунок равномерно движется по экрану.

Следующая простая программа, текст которой приведен в листинге II.8, а вид формы — на рис. II.15, демонстрирует движение окружности от левой к правой границе окна программы.

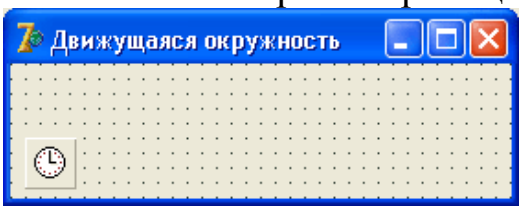


Рис. II.15. Форма программы **Движущаяся окружность**

Листинг II.8. Движущаяся окружность

```
unit mcircle_;
interface
```

```
uses
Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms,
Dialogs, ExtCtrls, StdCtrls;
type
TForm1 = class(TForm) Timer1: TTimer;
procedure Timer1Timer(Sender: TObject};
procedure FormActivate(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
implementation
{$R *.DFM}
var
Form1: TForml;
x,y: byte; // координаты центра окружности
dx: byte; // приращение координаты x при движении
окружности
// стирает и рисует окружность на новом месте
procedure Ris;
begin
// стереть окружность
form1.Canvas.Pen.Color:=form1.Color;
form1.Canvas.Ellipse(x,y,x+10,y+10);
x:=x+dx;
// нарисовать окружность на новом месте
form1.Canvas.Pen.Color:=clBlack;
form1.Canvas.Ellipse(x,y, x+10, y+10) ;
end;
// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin Ris; end;
procedure TForm1.FormActivate(Sender: TObject);
begin
x:=0;
y:=10;
dx:=5;
timer1.Interval:=50;
// период возникновения события OnTimer —0.5 сек
form1.canvas.brush.color:=forml.color;
```

```
end;
end.
```

Основную работу выполняет процедура Ris, которая стирает окружность и выводит ее на новом месте. Стирание окружности выполняется путем перерисовки окружности поверх нарисованной, но цветом фона.

Для обеспечения периодического вызова процедуры Ris в форму программы добавлен невизуальный компонент Timer (таймер), значок которого находится на вкладке **System** палитры компонентов (рис. II.16). Свойства компонента Timer, перечислены в табл. II.9.

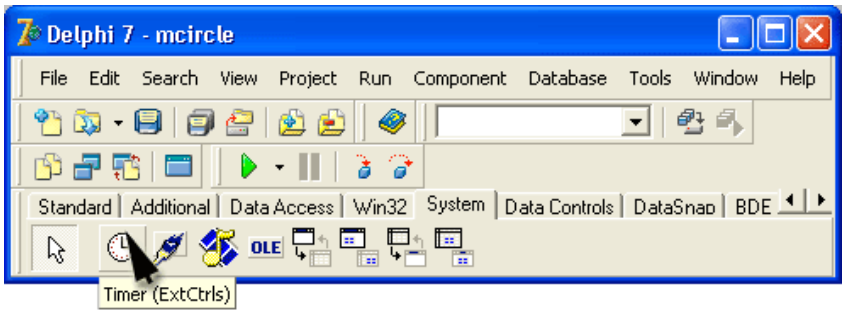


Рис. II.16. Значок компонента Timer

Таблица II.9. Свойства компонента Timer

Свойство	**Определяет**
Name Interval	Имя компонента. Используется для доступа к компоненту Период генерации события OnTimer. Задается в миллисекундах
Enabled	Разрешение работы. Разрешает (значение True) или запрещает (значение False) генерацию события OnTimer

Добавляется компонент Timer к форме обычным образом, однако, поскольку компонент Timer является невизуальным, т. е. во время работы программы не отображается на форме, его значок можно поместить в любое место формы.

Компонент Timer генерирует событие OnTimer. Период возникновения события OnTimer измеряется в миллисекундах и определяется значением свойства Interval. Следует обратить внимание на свойство Enabled. Оно дает возможность программе "запустить" или "остановить" таймер. Если значение свойства Enabled равно False, то событие OnTimer не возникает.

Событие onTimer в рассматриваемой программе обрабатывается процедурой TimeriTimer, которая, в свою очередь, вызывает процедуру Ris. Таким образом, в программе реализован механизм периодического вызова процедуры Ris.

Примечание

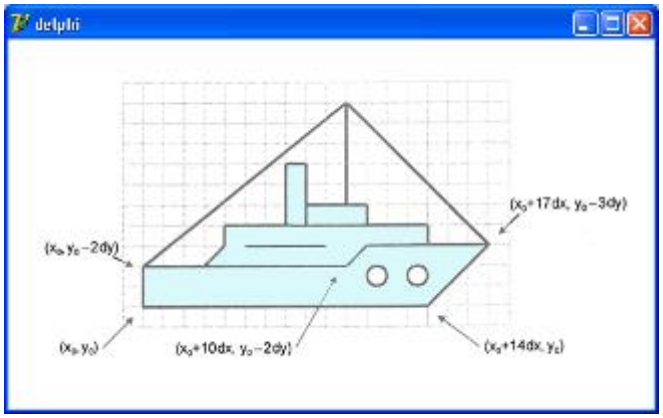
Переменные x, y (координаты центра окружности) и dx (приращение координаты x при движении окружности) объявлены вне процедуры Ris, т. е. они являются глобальными. Поэтому надо не забыть выполнить их инициализацию (в программе инициализацию глобальных переменных реализует процедура FormActivate).

2.18. Метод базовой точки

При программировании сложных изображений, состоящих из множества элементов, используется метод, который называется методом базовой точки. Суть этого метода заключается в следующем:

1. Выбирается некоторая точка изображения, которая принимается за базовую.
2. Координаты остальных точек отсчитываются от базовой точки.
3. Если координаты точек изображения отсчитывать от базовой в относительных единицах, а не в пикселах, то обеспечивается возможность масштабирования изображения.

На рис. II.17 приведено изображение кораблика. Базовой точкой является точка с координатами (X_0 Y_0). Координаты остальных точек отсчитываются именно от этой точки.



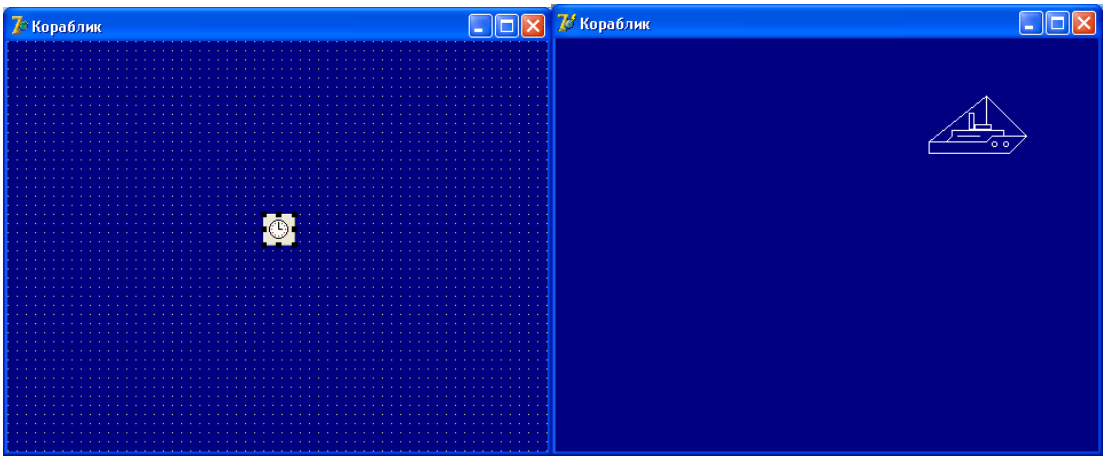


Рис. II.17. Определение координат изображения относительно базовой точки

В листинге II.9 приведен текст программы, которая выводит на экран изображение перемещающегося кораблика.

Листинг II.9. Кораблик

```
unit ship_;
interface
uses
Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs,
StdCtrls, ExtCtrls;
type
TForm1 = class(TForm)
Timer1: TTimer;
procedure Timer1Timer(Sender: TObject);
procedure FormActivate(Sender: TObject);
private
{ Private declarations } public
{ Public declarations } end;
var
Form1: TForm1;
x,y: integer; // координаты корабля (базовой точки)
implementation
{$R *.DFM}
// вычерчивает кораблик
procedure Titanik(x,y: integer; // координаты базовой точки
color: TColor); // цвет корабля
const dx = 5; dy = 5;
var
buf: TColor;
```

```
begin
with form1.canvas do begin
buf:=pen.Color; // сохраним текущий цвет
pen.Color:=color;
// установим нужный цвет
// рисуем . . .
// корпус MoveTo(x,y);
LineTo(x,y-2*dy) ;
LineTo (x+10*dx, y-2*dy) ;
LineTo (x+ll*dx, y-3*dy) ;
LineTo (x+17*dx,y-3*dy) ;
LineTo (x+14*dx, y) ;
LineTo (x,y) ;
// надстройка
MoveTo(x+3*dx,y-2*dy) ;
LineTo (x+4*dx, y-3*dy) ;
LineTo (x+4*dx, y-4*dy) ;
LineTo (x+13*dx,y-4*dy) ;
LineTo (x+13*dx, y-3*dy) ;
MoveTo(x+5*dx,y-3*dy) ;
LineTo (x+9*dx, y-3*dy) ;
// капитанский мостик
Rectangle (x+8*dx, y-4*dy, x+ll*dx, y-5*dy)
// труба
Rectangle (x+7*dx, y-4*dy, x+8*dx, y-7*dy) ;
// иллюминаторы
Ellipse (x+ll*dx,y-2*dy,x+12*dx,y-l*dy) ;
Ellipse (x+13*dx, y-2*dy, x+14*dx, y-l*dy) ;
// мачта
MoveTo(.x+10*dx,y-5*dy) ; LineTo(x+10*dx,y-10*dy);
// оснастка
MoveTo(x+17*dx,y-3*dy);
LineTo(x+10*dx,y-10*dy);
LineTo(x,y-2*dy);
pen.Color:=buf; // восстановим старый цвет карандаша
end;
end;
// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
Titanik(x,y,form1.color); // стереть рисунок
if x < Form1.ClientWidth
```

```
then x := x+5
else begin // новый рейс x := 0;
y := Random(50) + 100;
end;
Titanik(x,y,clWhite); // нарисовать в новой точке end;
procedure TForml.FormActivate(Sender: TObject);
begin
x:=0; y:=100;
Form1.Color:=clNavy;
Timerl.Interval := 50; // сигнал таймера каждые 50
миллисекунд
end;
end.
```

Отрисовку и стирание изображения кораблика выполняет процедура Titanik, которая получает в качестве параметров координаты базовой точки и цвет, которым надо вычертить изображение кораблика. Если при вызове процедуры цвет отличается от цвета фона формы, то процедура рисует кораблик, а если совпадает — то "стирает". В процедуре Titanik объявлены константы dx и dy, определяющие шаг (в пикселах), используемый при вычислении координат точек изображения. Меняя значения этих констант, можно проводить масштабирование изображения.

2.19. Использование битовых образов

В предыдущем примере изображение формировалось из графических примитивов. Теперь рассмотрим, как можно реализовать перемещение одного сложного изображения на фоне другого, например перемещение самолета на фоне городского пейзажа.

Эффект перемещения картинки может быть создан путем периодической перерисовки картинки с некоторым смещением относительно ее прежнего положения. При этом предполагается, что перед выводом картинки в новой точке сначала удаляется предыдущее изображение. Удаление картинки может быть выполнено путем перерисовки всей фоновой картинки или только той ее части, которая перекрыта битовым образом движущегося объекта.

В рассматриваемой программе используется второй подход. Картинка выводится применением метода Draw к свойству canvas компонента Image, а стирается путем копирования (метод

copyRect) нужной части фона из буфера на поверхность компонента Image.

Форма программы приведена на рис. II.18, а текст — в листинге II.10.

Компонент image используется для вывода фона, а компонент Timer — для организации задержки между циклами удаления и вывода на новом месте изображения самолета.

Листинг II.10. Летящий самолет

```
unit anim_;
interface
uses
Windows, Messages, SysUtils,
Classes, Graphics, Controls,
Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;
type
TForm1 = class(TForm)
Timer1: TTimer;
Image1: Tlmage;
procedure FormActivate(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure FormClose(Sender: TObject;
var Action: TCloseAction); private
{ Private declarations } public
{ Public declarations } end;
var
Form1: TForm1;
implementation
{$R *.DFM}
var
Back, bitmap, Buf : TBitMap; // фон, картинка, буфер
BackRct : TRect; // область фона, которая должна быть
// восстановлена из буфера
BufRet: Trect; // область буфера, которая используется для
// восстановления фона
x,y:integer; // текущее положение картинки
W,H: integer; // размеры картинки
procedure TForm1.FormActivate(Sender: TObject);
begin
// создать три объекта — битовых образа
Back := TBitmap.Create; // фон
bitmap := TBitmap.Create; // картинка
```

```
Buf := TBitmap.Create; // буфер
// загрузить и вывести фон
Back.LoadFromFile('factory.bmp');
Form1.Image1.canvas.Draw(0,0,Back);
// загрузить картинку, которая будет двигаться
bitmap.LoadFromFile('aplane.bmp');
// определим "прозрачный" цвет
bitmap.Transparent := True;
bitmap.TransParentColor := bitmap.canvas.pixels[1,1];
// создать буфер для сохранения копии области фона,
// на которую накладывается картинка
W:= bitmap.Width;
H:= bitmap.Height;
Buf.Width:= W;
Buf.Height:=H;
Buf.Palette:=Back.Palette;
// Чтобы обеспечить соответствие палитр //
Buf.Canvas.CopyMode:=cmSrcCopy;
// определим область буфера, которая
// будет использоваться
// для восстановления фона
BufRct:=Bounds(0,0,W,H);
// начальное положение картинки
x := -W; y := 20;
// определим сохраняемую область фона
BackRct:=Bounds(x,y,W,H); // и сохраним ее
Buf.Canvas.CopyRect(BufRet,Back.Canvas,BackRct);
end;
// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
// восстановлением фона (из буфера) удалим рисунок
Forml.image1.canvas.Draw(x,y,Buf);
x:=x+2;
if x>form1.Image1.Width then x:=-W;
// определим сохраняемую область фона
BackRct:=Bounds(x,y,W,H);
// сохраним ее копию
Buf.Canvas.CopyRect(BufRct,Back.Canvas,BackRct);
// выведем рисунок
Forml.image1.canvas.Draw(x,y,bitmap);
end;
```

```
// завершение работы программы
procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
// освободим память, выделенную
// для хранения битовых образов
Back.Free;
bitmap.Free;
Buf.Free; end;  end.
```

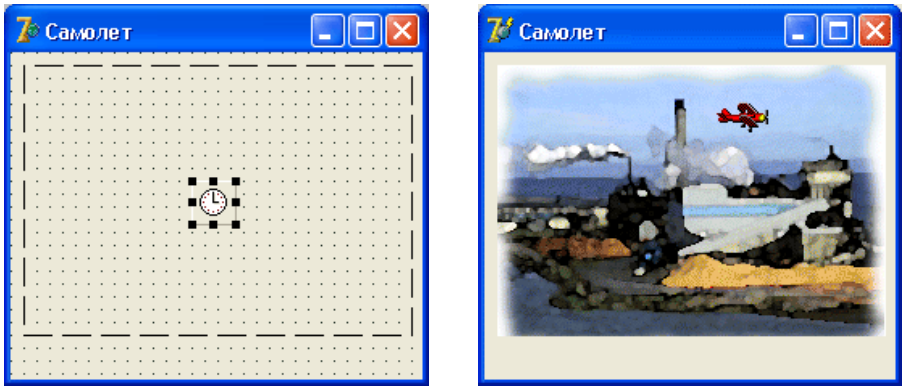


Рис. II.18. Форма программы **Самолет**

Для хранения битовых образов (картинок) фона и самолета, а также копии области фона, перекрываемой изображением самолета, используются объекты типа TBitMap, которые создаются динамически процедурой FormActivate. Эта же процедура загружает из файлов картинки фона (factory.bmp) и самолета (aplane.bmp), а также сохраняет область фона, на которую первый раз будет накладываться картинка.

Сохранение копии фона выполняется при помощи метода CopyRect, который позволяет выполнить копирование прямоугольного фрагмента одного битового образа в другой. Объект, к которому применяется метод CopyRect, является приемником копии битового образа. В качестве параметров методу передаются координаты и размер области, куда должно быть выполнено копирование, поверхность, откуда должно быть выполнено копирование, а также положение и размер копируемой области. Информация о положении и размере копируемой в буфер области фона, на которую будет наложено изображение самолета и которая впоследствии должна быть восстановлена из буфера, находится в структуре BackRct типа TRect. Для заполнения этой структуры используется функция Bounds.

Следует обратить внимание на то, что начальное значение переменной x, которая определяет положение левой верхней точки

битового образа движущейся картинки, — отрицательное число, равное ширине битового образа картинки. Поэтому в начале работы программы изображение самолета не появляется, картинка отрисовывается за границей видимой области. С каждым событием OnTimer значение координаты x увеличивается, и на экране появляется та часть битового образа, координаты которой больше нуля. Таким образом, у наблюдателя создается впечатление, что самолет вылетает из-за левой границы окна.

2.20. Загрузка битового образа из ресурса программы

В приведенной в листинге II.10 программе битовые образы фона и картинки загружаются из файлов. Это не всегда удобно. Delphi позволяет поместить необходимые битовые образы в виде ресурса в файл исполняемой программы и по мере необходимости загружать битовые образы из ресурса, т. е. из файла исполняемой программы (EXE-файла).

2.21. Создание файла ресурсов

Для того чтобы воспользоваться возможностью загрузки картинки из ресурса, необходимо сначала создать файл ресурсов, поместив в него нужные картинки.

Файл ресурсов можно создать при помощи утилиты Image Editor (Редактор изображений), которая запускается выбором команды **Image Editor** меню **Tools.**

Для того чтобы создать новый файл ресурсов, надо из меню **File** выбрать команду New, а затем в появившемся подменю — команду **Resource File** (Файл ресурсов)

В результате открывается окно нового файла ресурсов, а в строке меню окна **Image Editor** появляется новый пункт — **Resource**.

Для того чтобы в этот файл добавить новый ресурс, необходимо выбрать команду New меню **Resource** и из открывшегося списка — тип ресурса. В данном случае следует выбрать **Bitmap** (битовый образ). После выбора **Bitmap** открывается диалоговое окно **Bitmap Properties** (Свойства битового образа), используя которое можно установить размер (в пикселах) и количество цветов создаваемой картинки.

Нажатие кнопки ОК в диалоговом окне **Bitmap Properties** вызывает появление элемента **Bitmap1** в иерархическом списке **Contents.** Этот элемент соответствует новому ресурсу, добавленному в файл .

Bitmap1 — это автоматически созданное имя ресурса, которое может быть изменено выбором команды **Rename** меню **Resource** и вводом нужного имени. После изменения имени **Bitmap1** можно приступить к созданию битового образа. Для этого необходимо выбрать команду **Edit** меню **Resource,** в результате чего открывается окно графического редактора.

Графический редактор Image Editor предоставляет программисту стандартный для подобных редакторов набор инструментов, используя которые можно нарисовать нужную картинку. Если во время работы надо изменить масштаб отображения картинки, то для увеличения масштаба следует выбрать команду **Zoom In** меню **View,** а для уменьшения — команду **Zoom Out.** Увидеть картинку в реальном масштабе можно, выбрав команду **Actual Size** меню View.

Если нужная картинка уже существует в виде отдельного файла, то ее можно через буфер обмена (clipboard) поместить в битовый образ файла ресурсов. Делается это следующим образом.

1. Сначала надо запустить графический редактор, например Microsoft Paint, загрузить в него файл картинки и выделить всю картинку или ее часть. В процессе выделения следует обратить внимание на информацию о размере (в пикселах) выделенной области (Paint выводит размер выделяемой области в строке состояния). Затем, выбрав команду **Копировать** меню **Правка,** следует поместить копию выделенного фрагмента в буфер.
2. Далее нужно переключиться в Image Editor, выбрать ресурс, в который надо поместить находящуюся в буфере картинку, и установить значения характеристик ресурса в соответствии с характеристиками картинки, находящейся в буфере. Значения характеристик ресурса вводятся в поля диалогового окна **Bitmap Properties,** которое открывается выбором команды **Image Properties** меню **Bitmap.** После установки характеристик ресурса можно вставить картинку в ресурс, выбрав команду **Past** меню **Edit.**
3. После добавления всех нужных ресурсов файл ресурса следует сохранить в том каталоге, где находится программа, для которой этот файл создается. Сохраняется файл ресурса обычным образом, т. е. выбором команды Save меню **File.** Image Editor присваивает файлу ресурсов расширение res.

2.22. Подключение файла ресурсов

Для того чтобы ресурсы были доступны программе, необходимо в текст программы включить инструкцию (директиву), которая сообщит компилятору, что в файл исполняемой программы следует добавить содержимое файла ресурсов.

В общем виде эта директива выглядит следующим образом:

{$R ФайлРесурсов}

где ФайлРесурсов — имя файла ресурсов. Например, директива может выглядеть так:

{$R images.res}

Директиву включения файла ресурсов в файл исполняемой программы обычно помещают в начале текста модуля.

Примечание

Если имена файла модуля программы и файла ресурсов совпадают, то вместо имени файла ресурсов можно поставить "*". В этом случае директива включения файла ресурсов в файл исполняемой программы выглядит так:

{$R *.res}

Загрузить картинку из ресурса в переменную типа TBitMap можно при помощи метода LoadFromResourceName, который имеет два параметра: идентификатор программы и имя ресурса. В качестве идентификатора программы используется глобальная переменная Hinstance. Имя ресурса должно быть представлено в виде строковой константы.

Например, инструкция загрузки картинки в переменную Pic может выглядеть так:

```
Pic.LoadFromResourceName(Hinstance,'FACTORY') ;
```

В качестве примера в листинге II.11 приведен текст программы, в которой изображение фона и самолета загружается из ресурсов.

Листинг II.11. Пример загрузки картинок из ресурса

```
unit aplanel_;
{$R images.res} // включить файл ресурсов interface
uses
Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs,
ExtCtrls, StdCtrls, Buttons;
type
TForm1 = class(TForm)
Timer1: TTimer;
Image1: TImage;
```

```
procedure FormActivate(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure FormClose(Sender: TObject;
var Action: TCloseAction); private
{ Private declarations } public
{ Public declarations } end;
var
Form1: TForm1;
Back, bitmap, Buf : TBitMap;
// фон, картинка, буфер
BackRct, BufRet: TRect;
// область фона, картинки, буфера
x,y:integer;
// координаты левого верхнего угла картинки
W,H: integer; // размеры картинки
implementation
{$R *.DFM}
procedure TForm1.FormActivate(Sender: TObject);
begin
Back := TBitmap.Create; // фон
bitmap := TBitmap.Create; // картинка
Buf := TBitmap.Create; // буфер
// загрузить из ресурса фон
Back.LoadFromResourceName(HInstance,'FACTORY');
Forml.Image1.canvas.Draw(0,0,Back);
// загрузить из ресурса картинку, которая будет двигаться
bitmap.LoadFromResourceName(HInstance,'APLANE');
bitmap.Transparent := True;
bitmap.TransParentColor := bitmap.canvas.pixels[1,1];
// создать буфер для сохранения копии области фона, на
которую
// накладывается картинка
W:= bitmap.Width;
H:= bitmap.Height;
Buf.Width:= W;
Buf.Height:=H;
Buf.Palette:=Back.Palette; // Чтобы обеспечить соответствие
палитр !!
Buf.Canvas.CopyMode:=cmSrcCopy;
BufRct:=Bounds(0,0,W,H);
x:=-W; y:=20;
// определим сохраняемую область фона
```

```
BackRct:=Bounds(x,y,W,H); // и сохраним ее
Buf.Canvas.CopyRect(BufRet,Back.Canvas, BackRct);
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
// восстановлением фона (из буфера) удалим рисунок
Form1.image1.canvas.Draw(x,y, Buf);
x:=x+2;
if x>form1.Image1.Width then x:=-W;
// определим сохраняемую область фона
BackRct:=Bounds(x,y,W,H);
// сохраним ее копию
Buf.Canvas.CopyRect(BufRct,Back.Canvas,BackRct);
// выведем рисунок
Form1.image1.canvas.Draw(x,y,bitmap);
end;
procedure TForm1.FormClose(Sender: TObject;
var Action: TCloseAction);
begin
Back.Free;
bitmap.Free ;
Buf.Free;
end;
end.
```

Преимущества загрузки картинок из ресурса программы очевидны: при распространении программы не надо заботиться о том, чтобы во время работы программы были доступны файлы иллюстраций, все необходимые программе картинки находятся в исполняемом файле.

2.23. Просмотр "мультика"

Теперь рассмотрим, как можно реализовать вывод в диалоговом окне программы простого "мультика", подобного тому, который можно видеть в диалоговом окне **Установка связи** при подключении к Internet .

Эффект бегущего между телефоном и компьютером красного квадратика достигается за счет того, что в диалоговое окно выводятся сменяющие друг друга картинки.

Кадры мультика обычно находятся в одном файле или в одном ресурсе. Перед началом работы программы они загружаются

в буфер, в качестве которого удобно использовать объект типа TBitMap. Задача процедуры, реализующей вывод мультика, состоит в том, чтобы выделить очередной кадр и вывести его в нужное место формы.

Вывести кадр на поверхность формы можно применением метода copyRect к свойству canvas этой формы. Метод CopyRect копирует прямоугольную область одной графической поверхности на другую.

Инструкция применения метода CopyRect в общем виде выглядит так:

Canvas1.CopyRect(Область1, Canvas2, 06ласть2)

где:

- canvas1 — графическая поверхность, на которую выполняется копирование;
- Canvas2 — графическая поверхность, с которой выполняется копирование;
- параметр Область2 —- задает положение и размер копируемой прямоугольной области, а параметр областьі — положение копии на поверхности Canvas1.

В качестве параметров область! и область2 используются структуры типа TRect, поля которых определяют положение и размер области.

Заполнить поля структуры TRect можно при помощи функции Bounds, инструкция обращения к которой в общем виде выглядит так:

Bounds(x,y,Width,Height)

где:

- x и y — координаты левого верхнего угла области;
- width и Height — ширина и высота области.

Следующая программа, текст которой приведен в листинге II.12, выводит в диалоговое окно простой мультик — дельфийскую колонну, вокруг которой "летает" некоторый объект. На рис. II.19 приведены кадры этого мультика (содержимое файла film.bmp).

Диалоговое окно программы приведено на рис. II.20, оно содержит один единственный компонент — таймер.

Рис. II.19. Кадры мультика

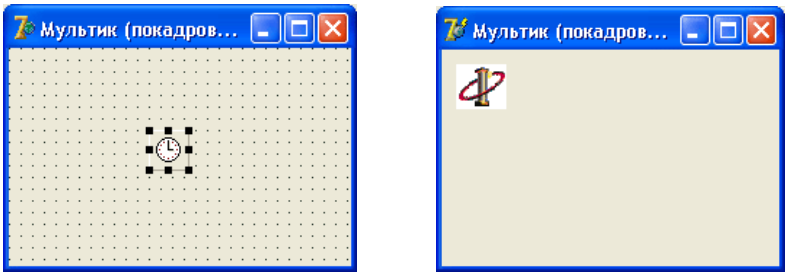


Рис. II.20. Форма программы

Листинг II.12. Мультик (использование метода CopRect)

```
unit multik ;
interface
uses
Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs,
ExtCtrls, StdCtrls;
type
TForm1 = class(TForm)
Timer1: TTimer;
procedure FormActivate(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1l: TForm1;
implementation
($R *.DFM}
const
FILMFILE = 'film2.bmp'; // фильм — bmp-файл
N_KADR=12; // кадров в фильме (для данного файла)
var
Film: TBitMap; // фильм — все кадры
WKadr,HKadr: integer; // ширина и высота кадра
CKadr: integer; // номер текущего кадра
RectKadr: TRect; // положение и размер кадра в фильме
Rect1 : Trect; // координаты и размер области отображения
фильма
procedure TForm1.FormActivate(Sender: TObject);
begin
Film := TBitMap.Create;
```

```
Film.LoadFromFile(FILMFILE);
WKadr := Round(Film.Width/N_Kadr);
HKadr := Film.Height;
Rect1 := Bounds(10,10,WKadr,HKadr);
Ckadr:=0;
Form1.Timerl.Interval := 150; // период обновления кадров —
0.15 с
Form1.Timerl.Enabled:=True; // запустить таймер
end;
// отрисовка кадра procedure DrawKadr;
begin
// определим положение текущего кадра в фильме
RectKadr:=Bounds(WKadr*CKadr,0,WKadr,HKadr);
// вывод кадра из фильма
Form1.Canvas.CopyRect(Rect1,Film*.Canvas,RectKadr);
// подготовимся к выводу следующего кадра
CKadr := CKadr+1;
if CKadr = N_KADR then CKadr:=0;:
end;
// обработка сигнала от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
DrawKadr;
end;
end.
```

Программа состоит из трех процедур. Процедура TForm1. FormActivate создает объект Film и загружает в него фильм — BMP-файл, в котором находятся кадры фильма. Затем, используя информацию о размере загруженного битового образа, процедура устанавливает значения характеристик кадра: высоту и ширину.

После этого создается объект Kadr (типа TBitMap), предназначенный для хранения текущего кадра. Следует обратить внимание, что после создания объекта Kadr принудительно устанавливаются значения свойств width и Height. Если этого не сделать, то созданный объект будет существовать, однако память для хранения битового образа не будет выделена. В конце своей работы процедура TForml. FormActivate устанавливает номер текущего кадра и запускает таймер.

Основную работу в программе выполняет процедура DrawKadr, которая выделяет из фильма очередной кадр и выводит его в форму. Выделение кадра и его отрисовку путем копирования

фрагмента картинки с одной поверхности на другую выполняет метод copyRect (рис. II.21), которому в качестве параметров передаются координаты области, куда нужно копировать, поверхность и положение области, откуда нужно копировать. Положение фрагмента в фильме, т. е. координата x левого верхнего угла, определяется умножением ширины кадра на номер текущего кадра. Запускает процедуру DrawKadr процедура TForm1.Timer1Timer, обрабатывающая событие OnTiner.

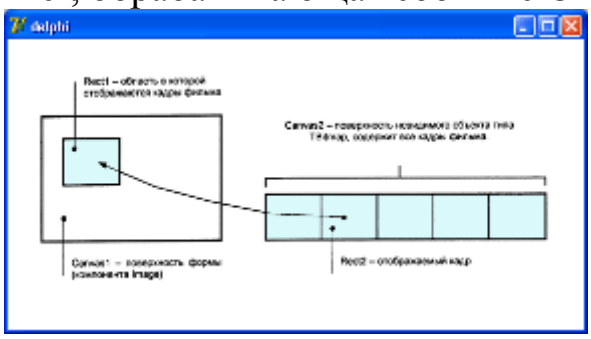


Рис. II.21. Инструкция Canvas1. CopyRect (Rect1, Canvas2, Rect2) **копирует в область** Rectl **поверхности** Canvasl **область** Rect2 **с поверхности** Canvas2

3. Мультимедиа-возможности Delphi

Большинство современных программ, работающих в среде Windows, являются мультимедийными. Такие программы обеспечивают просмотр видеороликов и мультипликации, воспроизведение музыки, речи, звуковых эффектов. Типичными примерами мультимедийных программ являются игры и обучающие программы.

Delphi предоставляет в распоряжение программиста два компонента, которые позволяют разрабатывать мультимедийные программы:

- Animate — обеспечивает вывод простой анимации (подобной той, которую видит пользователь во время копирования файлов);
- MediaPlayer — позволяет решать более сложные задачи, например, воспроизводить видеоролики, звук, сопровождаемую звуком анимацию.

3.1. Компонент Animate

Компонент Animate, значок которого находится на вкладке **Win32** (рис. III.1), позволяет воспроизводить простую анимацию, кадры которой находятся в AVI-файле.

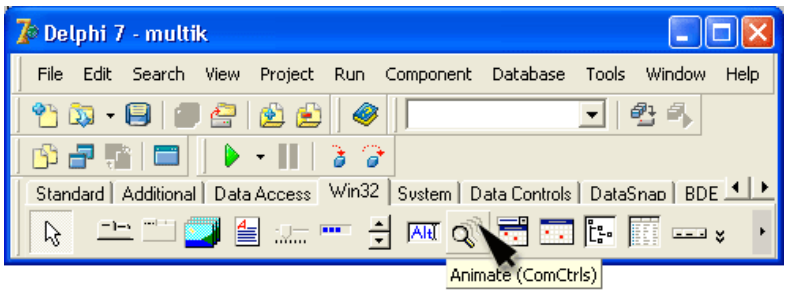


Рис. III.1. Значок компонента Animate

Примечание

Хотя анимация, находящаяся в AVI-файле может сопровождаться звуковыми эффектами (так ли это — можно проверить, например, при помощи стандартной программы Проигрыватель Windows Media), компонент Animate обеспечивает воспроизведение только изображения. Для полноценного воспроизведения сопровождаемой звуком анимации следует использовать компонент mediaPlayer.

Компонент Animate добавляется к форме обычным образом. После добавления компонента к форме следует установить значения его свойств. Свойства компонента Animate перечислены в табл. III.1.

Таблица III.1. Свойства компонента Animate

Свойство	**Определяет**
Name	Имя компонента. Используется для доступа к свойствам компонента и управлением его поведением
FileName	Имя AVI-файла в котором находится анимация, отображаемая при помощи компонента
StartFrame	Номер кадра, с которого начинается отображение анимации
stopFrame	Номер кадра, на котором заканчивается отображение анимации
Activate	Признак активизации процесса отображения кадров анимации
Color	Цвет фона компонента (цвет "экрана"), на котором воспроизводится анимация
Transparent	Режим использования "прозрачного" цвета при отображении анимации
Repetitions	Количество повторов отображения анимации

Следует еще раз обратить внимание, что компонент Animate предназначен для воспроизведения AVI-файлов, которые содержат только анимацию. При попытке присвоить записать в свойство FileName имя файла, который содержит звук, Delphi выводит сообщение о невозможности открытия указанного файла **(Cannot open** AVI). Чтобы увидеть, что находиться в AVI-файле: анимация и звук или только анимация, нужно из Windows раскрыть нужную папку, выделить AVI-файл и из контекстного меню выбрать команду **Свойства.** В результате этого откроется окно **Свойства,** на вкладке **Сводка** (рис. III.2) которого будет выведена подробная информация о содержимом выбранного файла.

Следующая программа, текст которой приведен в листинге III.1, демонстрирует использование компонента Animate для отображения в диалоговом окне программы анимации. Вид формы программы приведен на рис. III.3, а значения свойств компонента Animatel — в таблице III.2.

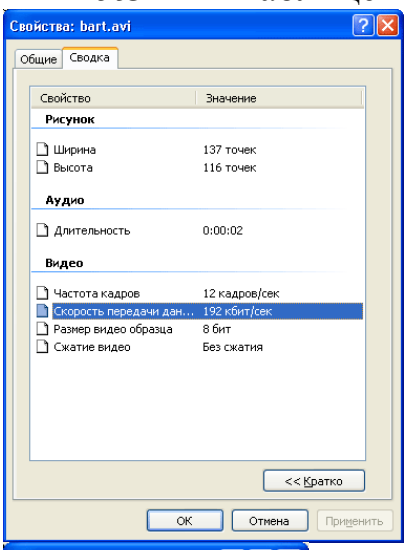


Рис. III.2. На вкладке **Сводка** отражается информация об AVI-файле

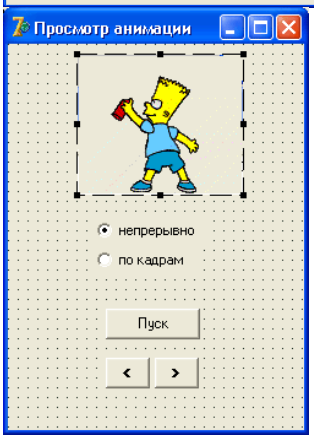


Рис. III.3. Форма программы **Просмотр анимации**

Таблица III.2. Значения свойств компонента Animate1

Свойство	Значение
FileName	bart.avi
Active	False
Transparent	True

После запуска программы в форме выводится первый кадр анимации. Программа обеспечивает два режима просмотра анимации: непрерывный и покадровый.

Кнопка Button1 используется как для инициализации процесса воспроизведения анимации, так и для его приостановки. Процесс непрерывного воспроизведения анимации инициирует процедура обработки события Onclick на кнопке Пуск, которая присваивает значение True свойству Active. Эта же процедура заменяет текст на кнопке Button1 с Пуск на Стоп. Режим воспроизведения анимации выбирается при помощи переключателей Ra-dioButton1 и RadioButton2. Процедуры обработки события Onclick на этих переключателях изменением значения свойства Enabled блокируют или, наоборот, делают доступными кнопки управления: активизации воспроизведения анимации (Buttoni), перехода к следующему (Button2) и предыдущему (Buttons) кадру. Во время непрерывного воспроизведения анимации процедура обработки события OnCkick на кнопке Стоп (Buttoni) присваивает значение False свойству Active и тем самым останавливает процесс воспроизведения анимации.

Листинг III.1. Использование компонента Animate

```
unit ShowAVI_; interface
uses
Windows, Messages, SysUtils,
Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, ComCtrls, ExtCtrls;
type
TForm1 = class(TForm)
Animate1: TAnimate; // компонент Animate
Button1: TButton; // кнопка Пуск-Стоп
Button2: TButton; // следующий кадр
Button3: TButton; // предыдущий кадр
RadioButton1: TRadioButton; // просмотр всей анимации
RadioButton2: TRadioButton; // покадровый просмотр
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
```

```
procedure Button3Click(Sender: TObject);
procedure RadioButton1Click(Sender: TObject);
procedure RadioButton2Click(Sender: TObject);
private
{ Private declarations } public
{ Public declarations ) end;
var
Form1: TForm1; // форма
CFrame: integer; // номер отображаемого кадра
// в режиме покадрового просмотра
implementation {$R *.DFM}
// к следующему кадру
procedure TForm1.Button2Click(Sender: TObject);
begin
if CFrame = 1 then Button2.Enabled := True;
if CFrame < Animate1.FrameCount then begin
CFrame := CFrame + 1;
// вывести кадр
Animate1.StartFrame := CFrame;
Animate1.StopFrame := CFrame;
Animate1.Active := True;
if CFrame = Animatel.FrameCount // текущий кадр —
последний
then Button2.Enabled:=False;
end;
end;
// к предыдущему кадру
procedure TForm1.Button3Click(Sender: TObject);
begin
if CFrame = Animate1.FrameCount
then Button2.Enabled := True;
if CFrame > 1 then begin
CFrame := CFrame — 1;
// вывести кадр
Animate1.StartFrame := CFrame;
Animate1.StopFrame := CFrame;
Animate1.Active := True;
if CFrame = 1 // текущий кадр — первый
then Form1.Button3.Enabled := False;
end;
end;
// активизация режима просмотра всей анимации
```

```
procedure TForml.RadioButtonlClick(Sender: TObject);
begin
Buttonl.Enabled:=True; //доступна кнопка Пуск
// сделать недоступными кнопки покадрового просмотра
Form1.Button3.Enabled:=False ;
Form1.Button2.Enabled:=False;
end;
// активизация режима покадрового просмотра
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
Button2.Enabled:=True; // кнопка Следующий кадр доступна
Buttons.Enabled:=False; // кнопка Предыдущий кадр
недоступна
// сделать недоступной кнопку Пуск — вывод всей анимации
Buttonl.Enabled:=False; end;
// пуск и остановка просмотра анимации
procedure TForm1.ButtonlClick(Sender: TObject);
begin
if Animate1.Active = False // в данный момент анимация не
выводится
then begin
Animate1.StartFrame:=l; // вывод с первого
Animate1.StopFrame:=Animate1.FrameCount; // по последний
кадр
Animate1.Active:=True;
Button1.caption:='Стоп';
RadioButton2.Enabled:=False;
end
else // анимация отображается
begin
Animate1.Active:=False; // остановить отображение
Button1.caption:='Пуск';
RadioButton2.Enabled:=True;
end;
end;
end.
```

Компонент Animate позволяет программисту использовать в своих программах стандартные анимации Windows. Вид анимации определяется значением свойства CommonAVI. Значение свойства задается при помощи именованной константы. В табл. III.3

приведены некоторые значения констант, вид анимации и описание процесса, для иллюстрации которого используется эти анимации.

Таблица III.3. Значение свойства comonAVi определяет анимацию

Значение	**Анимация**	**Процесс**
aviCopyFiles		Копирование файлов
AviDeleteFile		Удаление файла
aviRecycleFile		Удаление файла в корзину

3.2. Компонент MediaPlayer

Компонент MediaPlayer, значок которого находится на вкладке **System** (рис. III.4), позволяет воспроизводить видеоролики, звук и сопровождаемую звуком анимацию.

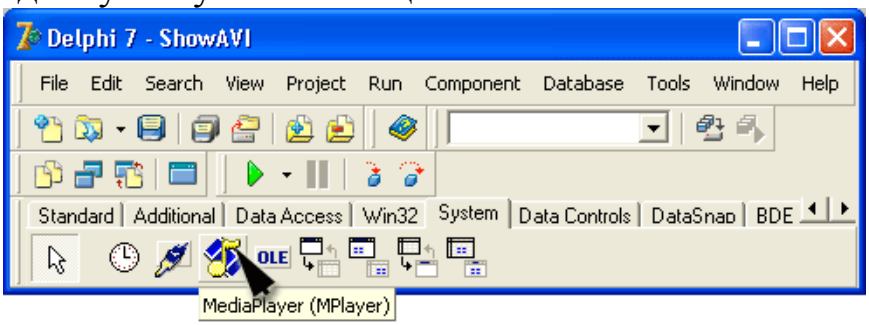


Рис. III.4. Значок компонента MediaPlayer

В результате добавления к форме компонента MediaPlayer на форме появляется группа кнопок (рис. III.5), подобных тем, которые можно видеть на обычном аудио- или видеоплеере. Назначение этих кнопок пояснено в табл. III.4. Свойства компонента MediaPlayer приведены в табл. III.5.

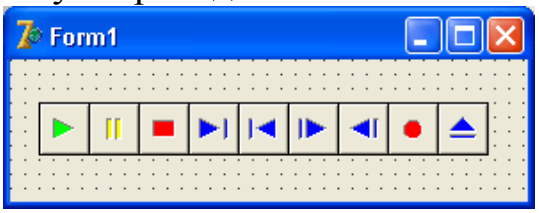


Рис. III.5. Компонент MediaPlayer

Таблица III.4. Кнопки компонента MediaPlayer

Кнопка	**Обозначение**	**Действие**
Воспроизведение	btPlay	Воспроизведение звука или видео
Пауза	btPause	Приостановка воспроизведения
Стоп	btStop	Остановка воспроизведения
Следующий	btNext	Переход к следующему кадру
Предыдущий	btPrev	Переход к предыдущему кадру
Шаг	btStep	Переход к следующему звуковому фрагменту, например, к следующей песне на CD
Назад	btBack	Переход к предыдущему звуковому фрагменту, например, к предыдущей песне на CD
Запись	btRecord	Запись
Открыть/Закрыть	btEject	Открытие или закрытие CD-дисковода компьютера

Таблица III.5. Свойства компонента MediaPiayer

Свойство	**Описание**
Name	Имя компонента. Используется для доступа к свойствам компонента и управлением работой плеера
DeviceType	Тип устройства. Определяет конкретное устройство, которое представляет собой компонент MediaPiayer. Тип устройства задается именованной константой: dtAutoSelect — тип устройства определяется автоматически; dtVaweAudio — проигрыватель звука; dtAVivideo — видеопроигрыватель; dtCDAudio — CD-проигрыватель
FileName	Имя файла, в котором находится воспроизводимый звуковой фрагмент или видеоролик
AutoOpen	Признак автоматического открытия сразу после запуска программы, файла видеоролика или звукового фрагмента
Display	Определяет компонент, на поверхности которого воспроизводится видеоролик (обычно в качестве экрана для отображения видео используют компонент Panel)
VisibleButtons	Составное свойство. Определяет видимые кнопки компонента. Позволяет сделать невидимыми некоторые

кнопки

3.3. Воспроизведение звука

Звуковые фрагменты находятся в файлах с расширением WAV. Например, в каталоге C:\Winnt\Media можно найти файлы со стандартными звуками Windows.

Следующая программа (вид ее диалогового окна приведен на рис. III.6, а текст - в листинге III.2) демонстрирует использование компонента ediaPiayer для воспроизведения звуковых фрагментов, находящихся в WAV-файлах.

Помимо компонента MediaPiayer на форме находится компонент ListBox и два компонента Label, первый из которых используется для вывода информационного сообщения, второй — для отображения имени WAV-файла, выбранного пользователем из списка.

Работает программа следующим образом. После появления диалогового окна воспроизводится "Звук Microsoft", затем пользователь может из списка выбрать любой из находящихся в каталоге C:\Windows\Media звуковых файлов и после щелчка на кнопке **Воспроизведение** услышать, что находится в этом файле.

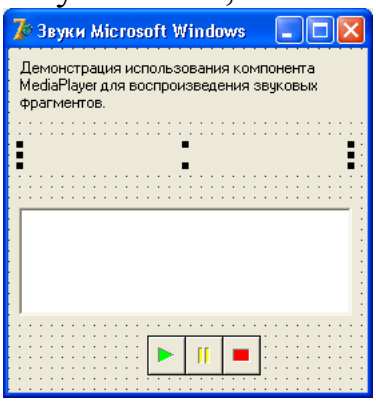


Рис. III.6. Форма программы **Звуки Microsoft Windows**

Значения измененных свойств компонента MediaPlayerl приведены в табл. III.6, значения остальных свойств оставлены без изменения.

Таблица III.6. Значения свойств компонента MediaPlayer1

Компонент	**Значение**
DeviceType	DtAutoSelect
FileName	C:\Winnt\Media\Звук Microsoft.wav
AutoOpen	True
VisibleButtons . btNext	False
VisibleButtons .btPrev	False
VisibleButtons . btStep	False
VisibleButtons . btBack	False
VisibleButtons . btRecord	False
VisibleButtons .btEject	False

Листинг III.2. Программа Звуки Microsoft Windows

```
unit WinSound_; interface
uses
Windows, Messages, SysUtils,
Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, MPlayer;
type
TForm1 = class(TForm)
MediaPlayerl: TMediaPlayer; // медиаплеер
Label1: TLabel; // информационное сообщение
ListBox1: TListBox; // список WAV-файлов
Label2: TLabel; // выбранный из списка файл
procedure FormActivate(Sender: TObject);
procedure ListBoxlClick(Sender: TObject);
procedure MediaPlayerlClick(Sender: TObject; Button:
TMPBtnType;
var DoDefault: Boolean); private
{ Private declarations } public
{ Public declarations } end;
const
SOUNDPATCH='c:\winnt\media\'; // положение звуковых
файлов
var
Form1: TForm1;
implementation
```

```
{$R *.DFM}
procedure TForm1.FormActivate(Sender: TObject);
var
SearchRec: TSearchRec; // структура, содержащая
информацию о файле,
// удовлетворяющем условию поиска
begin
Form1.MediaPlayer1.Play ;
// сформируем список WAV-файлов, находящихся
// в каталоге c:\winnt\media
if FindFirst(SOUNDPATCH+'*.wav', faAnyFile, SearchRec) =0
then
begin
// в каталоге есть файл с расширением WAV
// добавим имя этого файла в список
Form1.ListBox1.Items.Add(SearchRec.Name) ;
// пока в каталоге есть другие файлы с расширением WAV
while (FindNext(SearchRec) = 0) do
Form1.ListBox1.Items.Add(SearchRec.Name);
end;
end;
// щелчок на элементе списка
procedure TForm1.ListBoxlClick(Sender: TObject);
begin
// вывести в поле метки Label2 имя выбранного файла
Label2.Caption:=ListBox1.Items[ListBox1.itemlndex];
end;
// щелчок на кнопке компонента Media Player
procedure TForm1.MediaPlayerlClick(Sender: TObject; Button:
TMPBtnType;
var DoDefault: Boolean); begin
if (Button = btPlay) and (Label2.Caption <> '') then
begin
// нажата кнопка Play
with MediaPlayerl do begin
FileName:=SOUNDPATCH+Label2.Caption; // имя выбранного
файла
Open; // открыть и проиграть звуковой файл
end;
end;
end;
end.
```

Воспроизведение звука сразу после запуска программы активизирует процедура обработки события onFormActivate путем применением метода Play к компоненту MediaPlayerl (действие этого метода аналогично щелчку на кнопке **Воспроизведение).** Эта же процедура формирует список WAV-файлов, находящихся в каталоге C:\Winnt\Media. Для формирования списка используются функции FindFirst и FindNext, которые, соответственно, выполняют поиск первого и следующего (по отношению к последнему, найденному функцией FindFirst или FindNext) файла, удовлетворяющего указанному при вызове функций критерию. Обеим функциям в качестве параметров передаются маска WAV-файла (критерий поиска) и переменная -структура searchRec, поле Name которой в случае успешного поиска будет содержать имя файла, удовлетворяющего критерию поиска.

Щелчок на элементе списка обрабатывается процедурой TForm1.ListBox1Click, которая выводит в поле метки Label2 имя файла, выбранного пользователем (во время работы программы свойство ItemIndex содержит номер элемента списка на котором выполнен щелчок).

В результате щелчка на одной из кнопок компонента MediaPiayeri активизируется процедура TForm1.MediaPiayer1Click, которая проверяет, какая из кнопок компонента была нажата. Если нажата кнопка **Воспроизведение** (btPlay), то в свойство FileName **компонента** MediaPiayeri **записывается** имя выбранного пользователем файла, затем метод open загружает этот файл и активизирует процесс его воспроизведения.

Наличие у компонента MediaPiayer свойства visible позволяет скрыть компонент от пользователя и при этом применять его для воспроизведения звука без участия пользователя. Например, следующая программа пересчитывает вес из фунтов в килограммы и сопровождает выдачу результата звуковым сигналом. В случае, если пользователь забудет ввести исходные данные или введет их неверно, программа выведет сообщение об ошибке, также сопровождаемое звуковым сигналом. Вид диалогового окна программы во время ее разработки приведен на рис. III.7, значения свойств компонента MediaPlaer в табл. III.7. Текст модуля программы приведен в листинге III.3.

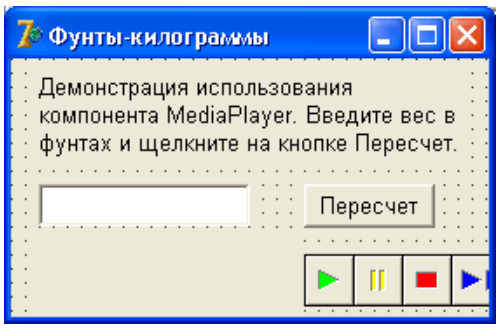


Рис. III.7. Диалоговое окно программы **Фунты-килограммы**

Таблица III.7. Значения свойств компонента MediaPiayer1

Свойство	**Значение**
Name DeviceType	MediaPiayer1
FileName	dtAutoSelect c : \winnt\media\ding . wav

Свойство	**Значение**
AutoOpen	True
Visible	False

Листинг III.3. Использование компонента MediaPlayer для вывода звука

```
unit FuntToKg1_; interface
uses
Windows, Messages, SysUtils,
Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, MPlayer;
type
TForm1 = class(TForm)
Edit1: TEdit; // поле ввода веса в фунтах
Button1: TButton; // кнопка Пересчет
Label2: TLabel; // поле вывода результата
Label1: TLabel; // поле информационного сообщения
MediaPlayer1: TMediaPlayer; // медиаплеер
procedure Button1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.DFM)
```

```
// щелчок на кнопке Пересчет
procedure TForm1.ButtonlClick(Sender: TObject);
var
f: real; // вес в фунтах k: real; // вес в килограммах
begin
form1.Label2.Caption: =' ';
try // возможна ошибка, если в поле
// Edit1 будет не число
f:=StrToFloat(Edit1.Text);
Forml.MediaPlayer1.Play;
// звуковой сигнал k:=f*0.4095;
Label2.caption:=Editl.text+' ф. - это ' +
FloatToStrF(k,ffGeneral,4,2}+' кг. ';
except
on EConvertError do // ошибка преобразования
begin
// определим и проиграем звук "Ошибка"
Form1.MediaPlayer1.FileName:=
'c:\windows\media\chord.wav';
Form1.MediaPlayer1.Open;
Form1.MediaPlayer1.Play; // звуковой сигнал
ShowMessage('Ошибка! Вес следует ввести числом.');
form1.Edit1.SetFocus; // курсор в поле ввода
// восстановим звук
Forml.MediaPlayer1.FileName:=
'c:\windows\media\ding.wav';
Forml.MediaPlayer1.Open;
end;
end;
end;
end.
```

3.4. Запись звука

В некоторых случаях программисту могут потребоваться специфические звуки или музыкальные фрагменты, которые не представлены на диске компьютера в виде WAV-файла. В этом случае возникает задача создания, или, как говорят, записи WAV-файла.

Наиболее просто получить представление нужного звукового фрагмента в виде WAV-файла можно при помощи входящей в состав Windows программы **Звукозапись.** Программа

Звукозапись, вид ее диалогового окна приведен на рис. III.8, запускается из главного меню Windows при помощи команды **Пуск | Программы | Стандартные | Развлечения | Звукозапись.**

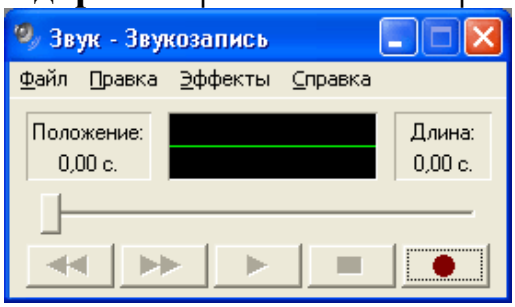


Рис. III.8. Диалоговое окно программы **Звукозапись**

Источником звука для программы **Звукозапись** может быть микрофон, аудио-CD или любое другое подключенное к линейному входу звуковой платы компьютера устройство, например аудиомагнитофон. Кроме того, возможно микширование (смешение) звуков различных источников.

Создается WAV-файл следующим образом. Сначала нужно определить источник (или источники) звука. Чтобы это сделать, надо открыть **Регулятор громкости** (для этого надо щелкнуть на находящемся на панели задач изображении динамика и из появившегося меню выбрать команду **Регулятор громкости)** и из меню **Параметры** выбрать команду **Свойства.** Затем в появившемся окне **Свойства** (рис. III.9) выбрать переключатель **Запись** и в списке **Отображаемые регуляторы громкости** установить флажки, соответствующие тем устройствам, сигнал с которых нужно записать. После щелчка на кнопке ОК на экране появляется окно **Уровень записи** (рис. III.10), используя которое, можно управлять уровнем сигнала (громкостью) каждого источника звука в общем звуке и величиной общего, суммарного сигнала, поступающего на вход программы **Звукозапись.** Величина сигнала задается перемещением движков соответствующих регуляторов. Следует обратить внимание на то, что движки регуляторов группы Уровень доступны только во время процесса записи звука. На этом подготовительные действия заканчиваются. Теперь можно приступить непосредственно к записи звука.

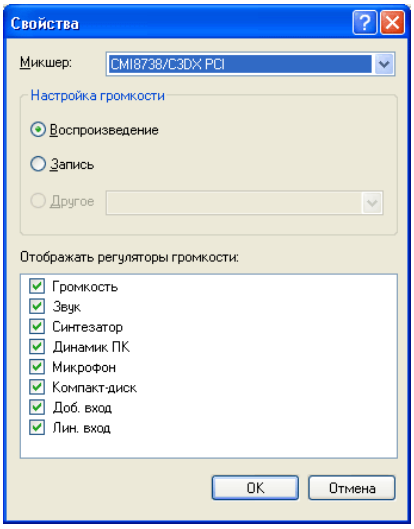


Рис. III.9. Диалоговое окно **Свойства**

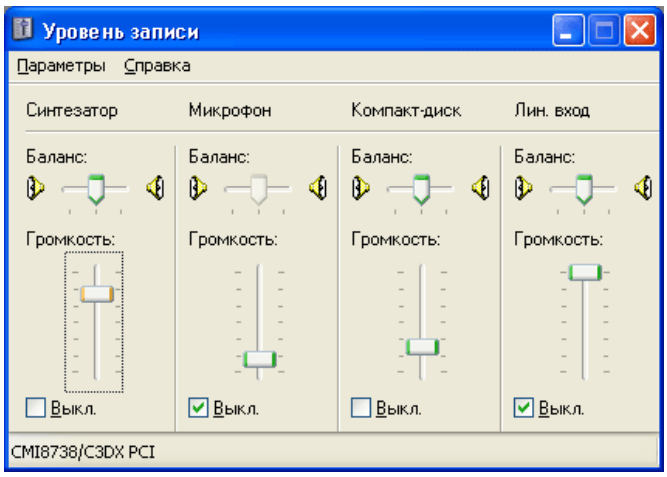


Рис. III.10. Диалоговое окно **Уровень записи** позволяет управлять записываемым сигналом

Чтобы записать музыкальный или речевой фрагмент, надо запустить программу **Звукозапись,** активизировать диалоговое окно **Уровень,** выбрать устройство-источник звука, инициировать процесс звучания (если запись осуществляется, например с CD) и в нужный момент времени щелкнуть на кнопке **Запись.**

Во время записи в диалоговых окнах можно наблюдать изменение сигнала на выходе микшера (индикатор **Громкость** диалогового окна **Уровень)** и на входе программы записи. На рис. III.11 в качестве примера приведен вид диалогового окна **Звукозапись** во время записи звука.

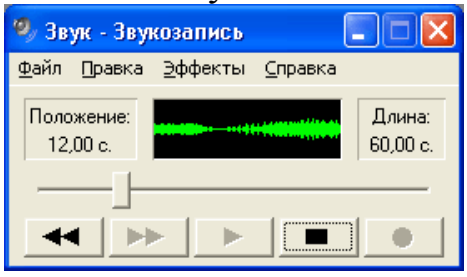


Рис. III.11. Диалоговое окно **Звукозапись** во время записи

Для остановки процесса записи следует щелкнуть на кнопке **Стоп.**

Сохраняется записанный фрагмент в файле обычным образом, т. е. выбором из меню **Файл** команды **Сохранить или Сохранить как.** При выборе команды **Сохранить как** можно выбрать формат, в котором будет сохранен записанный звуковой фрагмент.

Существует несколько форматов звуковых файлов. В частности, возможно сохранение звука с различным качеством как стерео, так и моно. Здесь следует понимать, что чем выше качество записи, тем больше места на диске компьютера требуется для хранения соответствующего WAV-файла. Считается, что для речи приемлемым является формат "22050 Гц, 8 бит, моно", а музыки - "44100 Гц, 16 бит, моно" или "44100 Гц, 16 бит, стерео".

3.5. Просмотр видеороликов и анимации

Помимо воспроизведения звука, компонент MediaPiayer позволяет просматривать видеоролики и мультипликации, представленные как AVI-файлы (AVI — это сокращение от Audio Video Interleave, что переводится как чередование звука и видео, т. е. AVI-файл содержит как звуковую, так и видеоинформацию) .
Процесс использования компонента MediaPiaer для посмотра содержимого AVI-файла рассмотрим на примере программы, которая в результате щелчка на командной кнопке воспроизводит на поверхности формы простую сопровождаемую звуковым эффектом мультипликацию — вращающееся по часовой стрелке слово Delphi (файл delphi.avi, содержащий этот мультик, находится на прилагаемом к книге диске).

Вид диалогового окна программы приведен на рис. III.12, а значения свойств компонента MediaPlayerl — В табл. III.8.

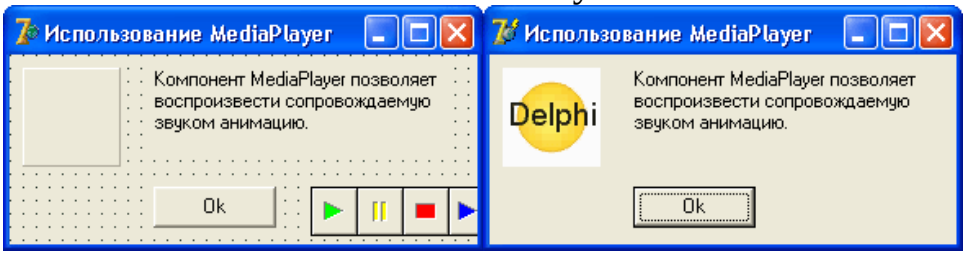


Рис. III.12. Форма и диалоговое окно программы **Использование MediaPlayer**

Таблица III.8. Значения свойств компонента MediaPlayer1

Свойство	**Значение**
Name	MediaPlayer1
FileName	delphi.avi
DeviceType	dtAVIVideo
AutoOpen	True
Display	Panel1
Visible	False

Создается форма приложения обычным образом. Компонент Paneii используется в качестве экрана, на который осуществляется вывод анимации, и его имя принимается в качестве значения свойства Display компонента MediaPlayeri. Поэтому сначала к форме лучше добавить компонент Panel и затем — MediaPlayer. Такой порядок создания формы позволяет установить значение свойства Display путем выбора из списка.

Следует особо обратить внимание на то, что размер области вывода анимации на панели определяется не значениями свойств width и Height панели (хотя их значения должны быть как минимум такими же, как ширина и высота анимации). Размер области определяется значением свойства

DisplayRect компонента MediaPlayer. Свойство DisplayRect ВО время разработки программы недоступно (его значение не выводится в окне **Object Inspector).** Поэтому значение свойства DisplayRect устанавливается во время работы программы в результате выполнения инструкции

MediaPlayer1.DisplayReet:=Rect(0,0,60,60).

Замечание

Чтобы получить информацию о размере кадров AVI-файла, надо, используя возможности Windows, открыть папку, в которой находится этот файл, щелкнуть правой кнопкой мыши на имени файла, выбрать команду **Свойства** и в появившемся диалоговом окне — вкладку Сводка, в которой выводится подробная информация о файле, в том числе и размер кадров.

Текст программы приведен в листинге III.4.

Листинг III.4. Воспроизведение анимации, сопровождаемой звуком

```
uses
Windows, Messages, SysUtils,
Classes, Graphics, Controls,
Forms, Dialogs, MPlayer, StdCtrls, ExtCtrls;
```

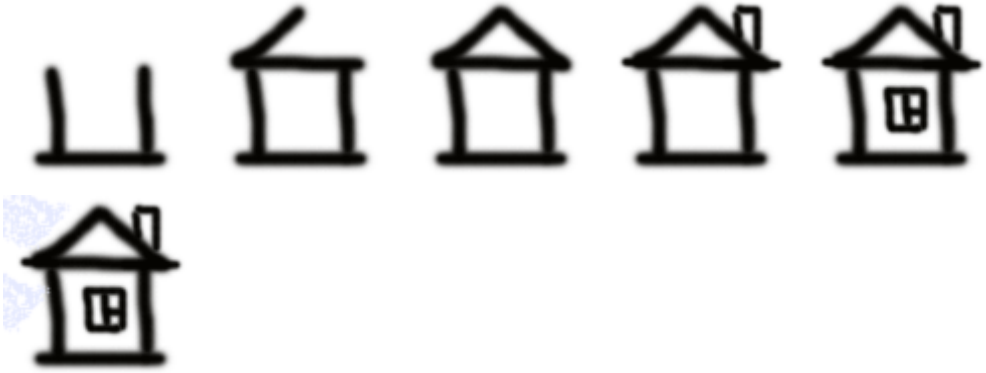
```
type
TForm1 = class(TForm)
Label1: TLabel; // информационное сообщение
Panel1: TPanel; // панель, на которую выводится анимация
Button1: TButton; // кнопка OK
MediaPlayer1: TMediaPlayer; // универсальный проигрыватель
procedure ButtonlClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
private
{ Private declarations ) public
{ Public declarations } end;
var
Form1: TForm1 ;
implementation
($R *.DFM}
procedure TForm1.ButtonlClick(Sender: TObject);
begin
MediaPlayer1.Play; // воспроизведение анимации
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
// зададим размер области вывода анимации
// на поверхности формы
MediaPlayer1.DisplayRect:=Rect(0,0,60,60);
end;
end.
```

Процесс воспроизведения анимации активизируется применением метода Play, что эквивалентно нажатию кнопки **Play** в случае, если кнопки компонента MediaPlayer доступны пользователю.

3.6. Создание анимации

Процесс создания файла анимации (AVI-файла) рассмотрим на примере. Пусть надо создать анимацию, которая воспроизводит процесс рисования эскиза Дельфийского храма (окончательный вид рисунка представлен на рис. III.13, несколько кадров анимации — на рис. III.14).

Рис. III.13. Эскиз Дельфийского храма



1 2 3 4 5 6

7 ... 37

Рис. III.14. Кадры анимации процесса рисования Дельфийского храма

Для решения поставленной задачи можно воспользоваться популярной программой Macromedia Flash 5.

В Macromedia Flash анимация, которую так же довольно часто называют роликом (Movie), состоит из слоев. В простейшем случае ролик представляет собой один единственный слой (Layer). Слой — это последовательность кадров (Frame), которые в процессе воспроизведения анимации выводятся последовательно, один за другим. Если ролик состоит из нескольких слоев, то кадры анимации получаются путем наложения кадров одного слоя на кадры другого. Например, один слой может содержать изображение фона, на котором разворачивается действие, а другой — изображение персонажей. Возможность формирования изображения путем наложения слоев существенно облегчает процесс создания анимации. Таким образом, чтобы создать анимацию, нужно распределить изображение по слоям и для каждого слоя создать кадры.

После запуска Macromedia Flash на фоне главного окна программы появляется окно **Move1** (рис. III.15), которое используется для создания анимации. В верхней части окна, которая называется Timeline, отражена структура анимации, в нижней части, которая называется рабочей областью, находится изображение текущего кадра выбранного слоя. После запуска Macromedia Flash анимация состоит из одного слоя (Layer 1), который в свою очередь представляет один пустой (чистый) кадр.

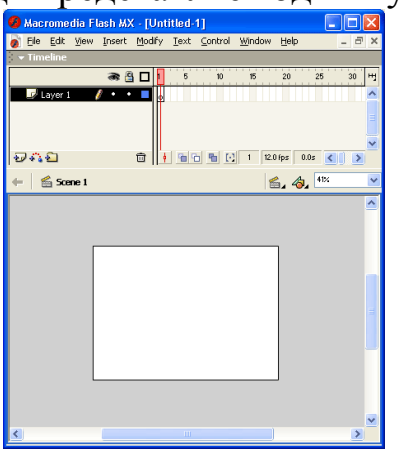


Рис. III.15. Окно **Movie** в начале работы над новой анимацией

Перед тем как приступить непосредственно к созданию кадров анимации, нужно задать общие характеристики анимации (ролика): размер кадров и скорость их воспроизведения. Характеристики вводятся в поля диалогового окна **Movie Properties** (рис. III.16), которое появляется в результате выбора из меню **Modify** команды **Movie.** В поле **Frame Rate** нужно ввести скорость воспроизведения ролика, которая измеряется в кадрах в секунду (fps — frame per second, кадров в секунду), в поля **Width и Height** — ширину и высоту кадров. В этом же окне можно выбрать фон кадров (список **Background Color).**

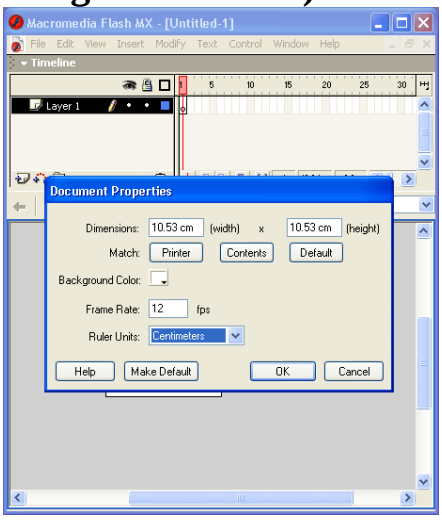


Рис. III.16. Характеристики ролика отображаются в окне **Movie Properties**

После того, как установлены характеристики ролика, можно приступить к созданию кадров анимации.

Первый кадр нужно просто нарисовать. Технология создания изображений Macromedia Flash обычная, используется стандартный набор инструментов: кисть, карандаш, пульверизатор, резинка и др.

Чтобы создать следующий кадр, нужно из меню **Insert** выбрать команду **Keyframe.** В результате в текущий слой будет добавлен кадр, в который будет скопировано содержимое предыдущего кадра (так как в большинстве случаев следующий кадр создается путем изменения предыдущего). Теперь можно нарисовать второй кадр. Аналогичным образом создаются остальные кадры анимации.

Иногда не нужно, чтобы новый кадр содержал изображение предыдущего, в этом случае вместо команды **Keyframe** нужно воспользоваться командой **Blank Keyframe.**

Если некоторое изображение должно оставаться статичным в течение времени, кратного выводу нескольких кадров, то вместо того, чтобы вставлять в слой несколько одинаковых кадров (Keyframe), нужно сделать кадр статичным. Если кадр, изображение которого должно быть статичным, является последним кадром ролика, то в окне Timeline нужно выделить кадр, до которого изображение должно оставаться статичным, и из меню **Insert** выбрать команду **Frame.** Если кадр, изображение которого должно быть статичным, не является последним, то нужно выделить этот кадр и несколько раз из меню **Insert** выбрать команду **Frame.**

Можно значительно облегчить работу по созданию анимации, если разделить изображение на основное и фоновое, поместив каждое в отдельный слой (именно так поступают при создании мультфильмов). Сначала нужно создать кадры слоя фона так, как было описано выше. Затем, выбрав из меню **Insert** команду **Layer,** нужно добавить слой основного действия.

Следует обратить внимание, что все действия по редактированию изображения направлены на текущий кадр выбранного слоя. В списке слоев выбранный слой выделен цветом, номер текущего кадра помечен маркером — красным квадратиком.

Чтобы выводимая анимация сопровождалась звуком, нужно сначала сделать доступным соответствующий звуковой файл. Для

этого надо из меню **File** выбрать команду **Import** и добавить в проект звуковой файл (рис. III.17).

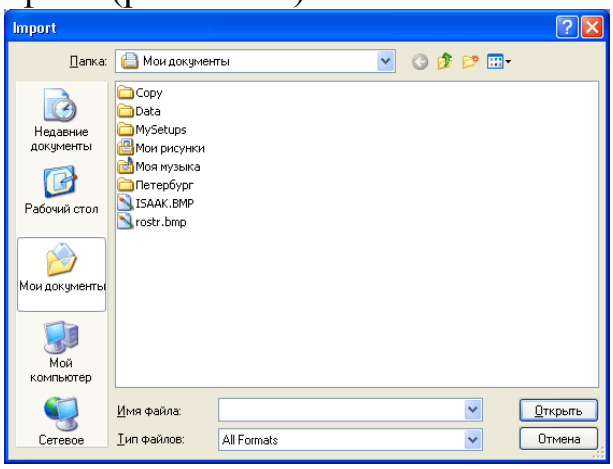


Рис. III.17. Импорт звукового файла

Затем в окне Timeline нужно выделить кадр, при отображении которого должно начаться воспроизведение звукового фрагмента, используя диалоговое окно **Sound** (рис. III.18), выбрать звуковой фрагмент и задать, если нужно, параметры его воспроизведения. Количество повторов нужно ввести в поле **Loops,** эффект, используемый при воспроизведении, можно выбрать из списка **Effect.**

В качестве примера на рис. III.19 приведен вид окна Timeline в конце работы над анимацией. Анимация состоит из двух слоев. Слой Layer 2 содержит фон. Детали фона появляются постепенно, в течение 9 кадров. После этого фон не меняется, поэтому 9 кадр является статичным. Слой Layer 1 содержит слой основного действия, которое начинается после того, как будет выведен фон. Вывод анимации заканчивается стандартным звуком **TADA** (его длительность равна одной секунде). Начало воспроизведения звука совпадает с выводом последнего (49-го, если считать от начала ролика) кадра основного действия, поэтому этот кадр сделан статичным в течение вывода следующих 12 кадров (скорость вывода анимации — 12 кадров в секунду). Сделано это для того, чтобы процесс вывода анимации завершился одновременно с окончанием звукового сигнала.

После того как ролик будет готов, его надо сохранить. Делается это обычным образом, то есть выбором из меню **File** команды Save.

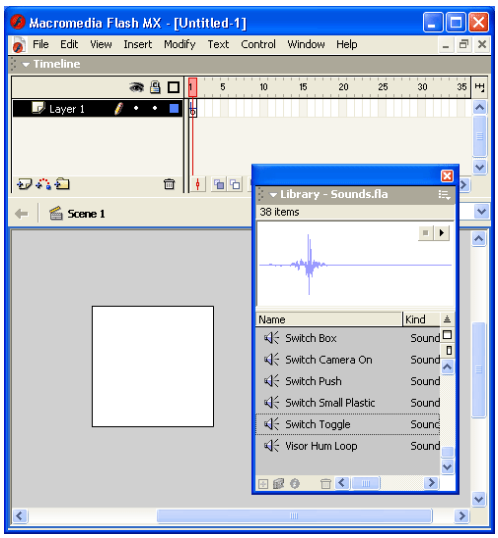


Рис. III.18. Диалоговое окно **Sound**

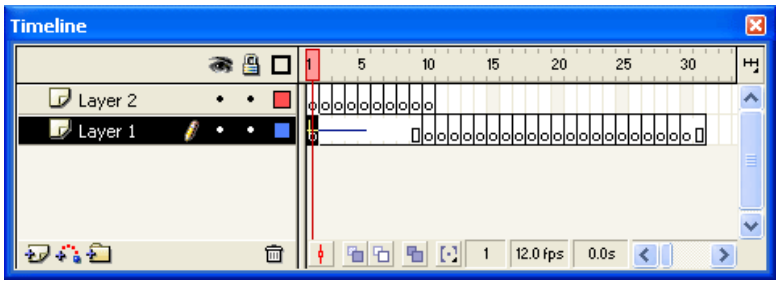


Рис. III.19. Пример анимации

Для преобразования файла из формата Macromedia Flash в AVI-формат нужно из меню **File** выбрать команду **Export Movie** и задать имя файла. Затем в появившемся диалоговом окне **Export Windows AVI** (рис. III.20) нужно задать размер кадра (поля Width и Height), из списка **Video Format** выбрать формат, в котором будет записана видеочасть ролика, а из поля **Sound Format** — формат звука.

Рис. III.20. Окно **Export Windows AVI**

Если установлен переключатель **Compress video,** то после щелчка на кнопке **ОК** появится диалоговое окно, в котором можно будет выбрать один из стандартных методов сжатия видео. При выборе видео и звукового формата нужно учитывать, что чем более высокие требования будут предъявлены к качеству записи звука и

изображения, тем больше места на диске займет AVI-файл. Здесь следует иметь в виду, что завышенные требования не всегда оправданы.

IV.РАСШИРЕННЫЕ СРЕДСТВА ГРАФИКИ

1.Пакет plots

1.1.Общая характеристика пакета plots

Пакет plots содержит почти полсотни графических функций, существенно расширяющих возможности построения двумерных и трехмерных графиков в Maple 7:

> with(plots);

[animate, animate3d, animatecurve, changecoords, complexplot, complexplotSd, conformal, contourplot, contourplotSd, coordplot, coordplotld, cylinderplot, densityplot, display, displayed, fteldplot,fieldplot3d, gradplot, gmdplotSd, implicitplot, implicitplot3d, inequal, listcontplot, HslcontplotSd, listdensityplot, listplot, Iistplot3d, loglogplot, logplot, matrixplot, odeplot>pareto,pointplot, pointplotSd, polarplot, polygonplot, polygonplotSd, palyhedrajsupported, polyhedraplot, replot, rootlocus, semilogplot, setoptions, setoptionsSd, spacecurve, sparsematrixplot, sphereplot, surfdata, textplot, textplotSd, tubeplot]

Ввиду важности этого пакета отметим назначение всех его функций:

- animate — создает анимацию двумерных графиков функций;
- animate3d — создает анимацию трехмерных графиков функций;
- animatecurve — создает анимацию кривых;
- changecoords — смена системы координат;
- complexplot — построение двумерного графика на комплексной плоскости;
- complexplot3d — построение трехмерного графика в комплексном пространстве;
- conformal — конформный график комплексной функции;
- contourplot — построение контурного графика;
- contourplot3d — построение трехмерного контурного графика;
- coordplot — построение координатной системы двумерных графиков;

- coordplot3d — построение координатной системы трехмерных графиков;
- cylinderplot — построение графика поверхности в цилиндрических координатах;
- densityplot — построение двумерного графика плотности;
- display — построение графика для списка графических объектов;
- display3d — построение графика для списка трехмерных графических объектов;
- fieldplot — построение графика двумерного векторного поля;
- fieldplot3d — построение графика трехмерного векторного поля;
- gradplot — построение графика двумерного векторного поля градиента;
- gradplot3d — построение графика трехмерного векторного поля градиента;
- implicitplot — построение двумерного графика неявной функции;
- implicitp1ot3d — построение трехмерного графика неявной функции;
- inequal — построение графика решения системы неравенств;
- listcontplot — построение двумерного контурного графика для сетки значений;
- Iistcontplot3d — построение трехмерного контурного графика для сетки значений;
- listdensityplot — построение двумерного графика плотности для сетки значений;
- listplot — построение двумерного графика для списка значений;
- listplot3d — построение трехмерного графика для списка значений;
- loglogplot — построение логарифмического двумерного графика функции;
- logplot — построение полулогарифмического двумерного графика функции;
- matrixplot — построение трехмерного графика со значениями Z, определенными матрицей;
- odeplot — построение двумерного или трехмерного графика решения дифференциальных уравнений;
- pareto — построение диаграммы (гистограммы и графика);
- pointplot — построение точками двумерного графика;
- poi ntplot3d — построение точками трехмерного графика;

- polarplot — построение графика двумерной кривой в полярной системе координат;
- polygonplot — построение графика одного или нескольких многоугольников;
- polygonplot3d — построение одного или нескольких многоугольников;
- polyhedraplot — построение трехмерного многогранника;
- replot — перестроение графика заново;
- rootlocus — построение графика корней уравнения с комплексными неизвестными;
- semilogplot — построение графика функции с логарифмическим масштабом по оси абсцисс;
- setoptions — установка параметров по умолчанию для двумерных графиков;
- setoptions3d — установка параметров по умолчанию для трехмерных графиков;
- spaeecurve — построение трехмерных кривых;
- sparsematrixplot — построение двумерного графика отличных от нуля значений матрицы;
- sphereplot — построение графика трехмерной поверхности в сферических координатах;
- surfdata — построение трехмерного графика поверхности по численным данным;
- textplot — вывод текста на заданное место двумерного графика;
- textplot3d — вывод текста на заданное место трехмерного графика;
- tubeplot — построение трехмерного графика типа «трубы».

Среди этих функций надо отметить прежде всего средства построения графиков ряда новых типов (например, в виде линий равного уровня, векторных полей и т. д.), а также средства объединения различных графиков в один. Особый интерес представляют две первые функции, обеспечивающие анимацию как двумерных (animate), так и трехмерных графиков (animate3d). Этот пакет вполне заслуживает описания в отдельной книге. Но, учитывая ограниченный объем данной книги, мы рассмотрим лишь несколько характерных примеров его применения. Заметим, что для использования приведенных функций нужен вызов пакета, например командой with(plots).

1.2.Построение графиков функций в двумерной полярной системе координат

В пакете plots есть функция для построения графиков в полярной системе координат. Она имеет вид polarplot(L,o), где L — объекты для задания функции, график которой строится, и o — необязательные параметры. На рис. 12.1, сверху, представлен пример построения графика с помощью функции polarplot. В данном случае для большей выразительности опущено построение координатных осей, а график выведен линией удвоенной толщины. График очень напоминает лист клена, весьма почитаемого в Канаде и ставшего эмблемой Maple.

1.3.Построение двумерных графиков типа implicitplot

В математике часто встречается особый тип задания геометрических фигур, при котором переменные x и y связаны неявной зависимостью. Например, окружность задается выражением $x^2 + y^2 = R^2$, где R — радиус окружности. Для задания двумерного графика такого вида служит функция импликативной графики:

implicitplot(eqn,x=a..b,y=c..d,options)

Пример построения окружности с помощью этой функции показан на рис. 12.1, снизу. Чуть ниже мы рассмотрим подобную функцию и для трехмерного графика.

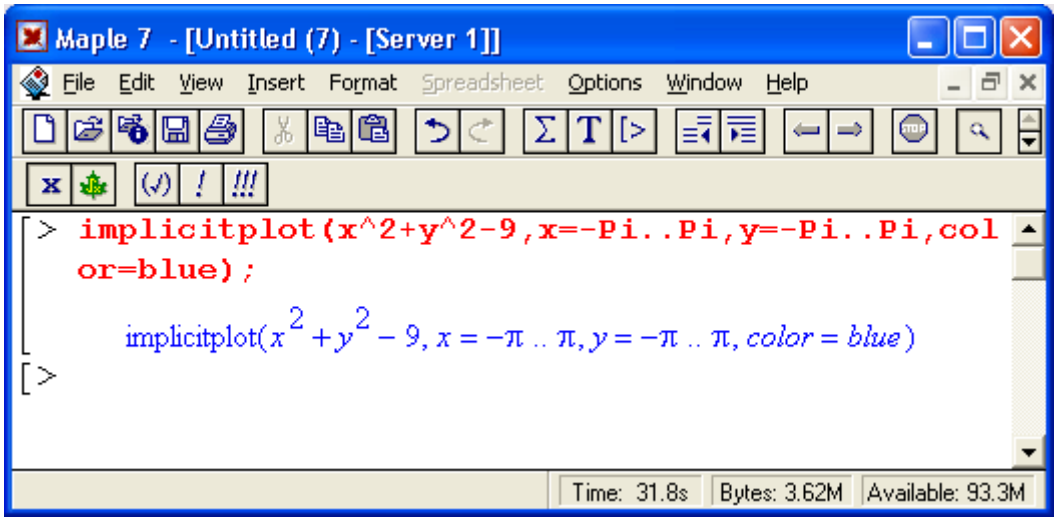


Рис. 12.1. Графики, построенные с помощью функций polarptot и impticitplot

1.4.Построение графиков линиями равного уровня

Графики, построенные с помощью линий равного уровня (их также называют контурными графиками), часто используются в картографии. Эти графики получаются, если мысленно провести через трехмерную поверхность ряд равноотстоящих плоскостей, параллельных плоскости, образованной осями X иY графика. Линии равных высот образуются в результате пересечения этих плоскостей с трехмерной поверхностью.

Для построения таких графиков используется функция contourplot, которая может использоваться в нескольких форматах:

contourplot(exprl,x=a..b,y=c..d)

contourplot(f,a..b,c..d)

contourplot([exprf ,exprg,exprh] S=a. .b,t=c. .d)

contourplot([f.g.h],a..b,c..d)

contourp1ot3d(exprl,x=a..b,y=c. .d)

contourplot3d(f,a..b,c..d)

contourplot3d([exprf,exprg,exprh],s=a..b,t=c,.d)

contourplot3d([f.g.h],a..b,c..d)

Здесь f, g и h — функции; exprl — выражение, описывающее зависимость высоты поверхности от координат x и y; exprf, exprg и exprh — выражения, зависящие от s и t, описывающие поверхность в параметрической форме; a и b — константы вещественного типа; end — константы или выражения вещественного типа; x, y, s и t — имена независимых переменных.

На рис. 12.2 показано построение графика линиями равного уровня для одной функции. Параметр filled=true обеспечивает автоматическую функциональную окраску замкнутых фигур, образованных линиями равного уровня. Порою это придает графику большую выразительность, чем при построении только линий равного уровня.

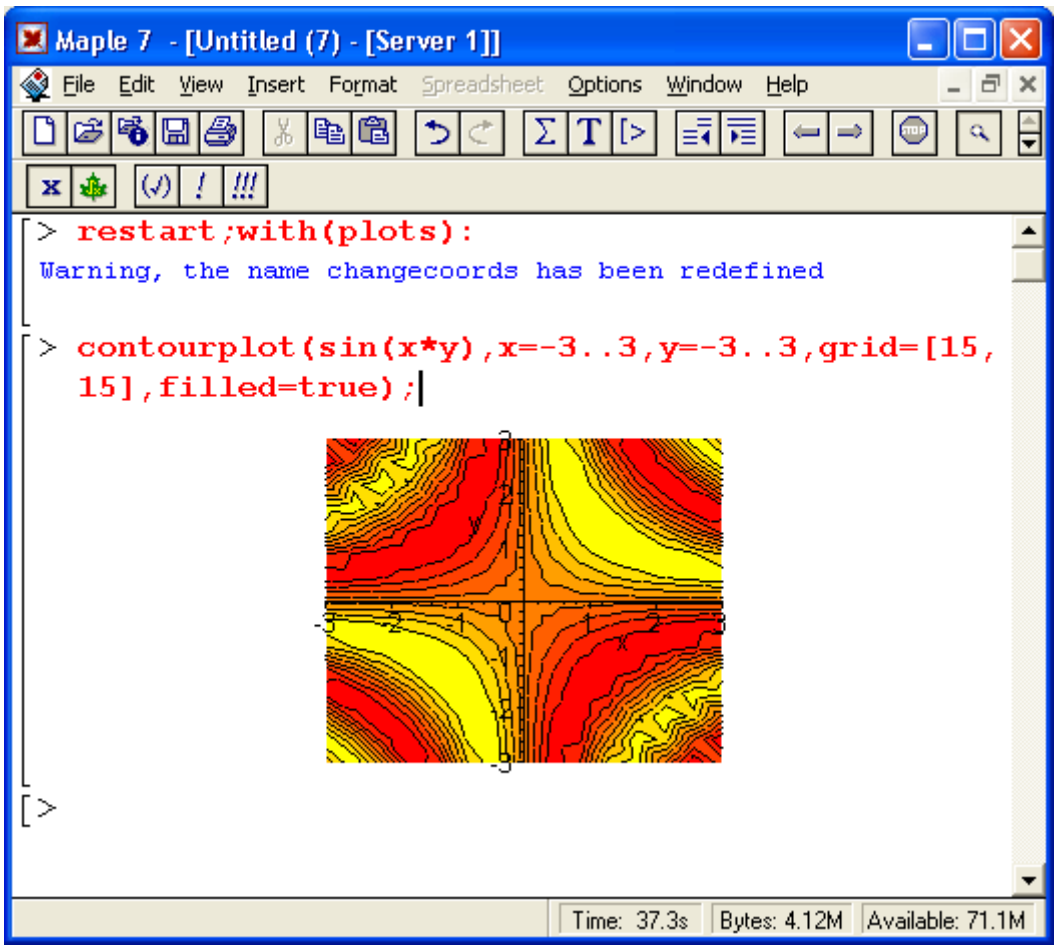


Рис. 12.2. Пример построения графика функции линиями равного уровня

Обратите внимание на то, что данная функция по умолчанию строит легенду — она видна под графиком в виде линий с надписями. К сожалению, в данном варианте окраски сами контурные линии получаются черными и их невозможно отличить. Однако если убрать параметр filled=true, то контурные линии (и линии легенды) будут иметь разный цвет и легко различаться.

Функция contourplot позволяет строить и графики ряда функций. Пример такого построения показан на рис. 12.3. Множество окружностей на этом рисунке создается четырьмя поверхностями, заданными функциями cl, c2, c3 и c4.

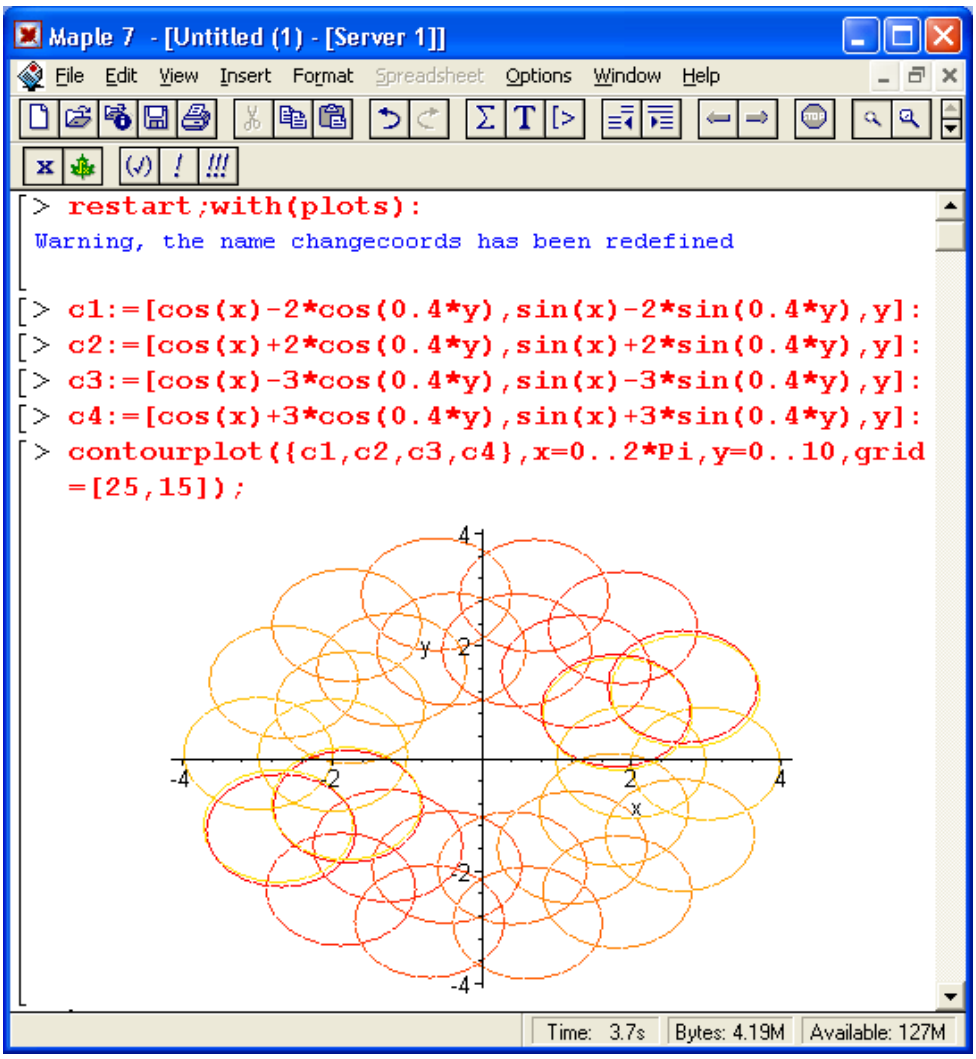


Рис. 12.3. Пример построения графиков многих функций линиями равного уровня

ВНИМАНИЕ

Обратите внимание, что на многих графиках Maple 7 по умолчанию вписывает легенду,то есть список линий с обозначениями. Иногда (как, например, на рис. 12.3) этот список оказывается просто некстати. Легенду можно убрать, расширив заодно место для графика, сняв флажок Show Legend в меню Legend, которое появляется при двойном щелчке на графике (это меню видно на рис. 12.3). То же самое можно сделать с помощью той же команды в контекстном меню. Заодно запомните, что легенду можно редактировать, выполнив команду Edit Legend.

Следует отметить, что хотя графики в виде линий равного уровня выглядят не так эстетично и естественно, как обычные графики трехмерных поверхностей (ибо требуют осмысления результатов), у них есть один существенный плюс - экстремумы функций на

таких графиках выявляются порой более четко, чем на обычных графиках. Например, небольшая возвышенность или впадина за большой «горой» на обычном графике может оказаться невидимой, поскольку заслоняется «горой». На графике линий равного уровня этого эффекта нет. Однако выразительность таких графиков сильно зависит от числа контурных линий.

1.5.График плотности

Иногда поверхности отображаются на плоскости как графики плотности окраски — чем выше высота поверхности, тем плотнее (темнее) окраска. Такой вид графиков создается функцией densityplot. Она может записываться в двух форматах:

densityplot(exprl.x=a..b,y=c..d)

densityplot(f,a..b,c..d)

где назначение параметров соответствует указанному выше для функции contourplot. На рис. 12.4 (верхняя часть) дан пример построения графика такого типа. Нетрудно заметить, что в плоскости XY график разбит на квадраты, плотность окраски которых различна. В нашем случае плотность окраски задается оттенками серого цвета.

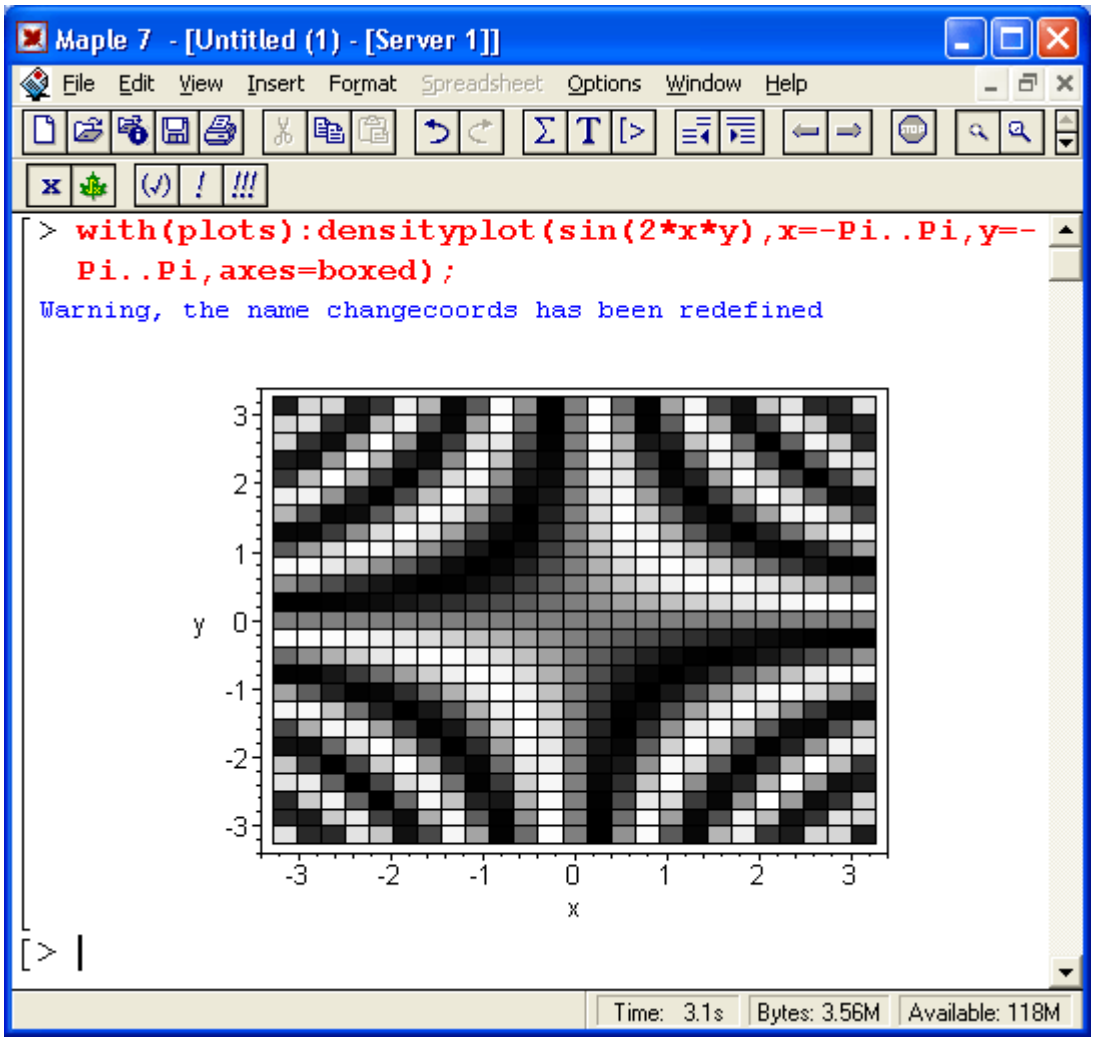


Рис. 12.4. Графики плотности и поля векторов

Обычно графики такого типа не очень выразительны, но имеют свои области применения. К примеру, оттенки окраски полупрозрачной жидкости могут указывать на рельеф поверхности дна емкости, в которой находится эта жидкость.

1.6.Двумерный график векторного поля

Еще один распространенный способ представления трехмерных поверхностей —-графики полей векторов. Они часто применяются для отображения полей, например электрических зарядов. Особенность таких графиков в том, что для их построения используют стрелки, направление которых соответствует направлению изменения градиента поля, а длина — значению градиента. Так что термин «поле векторов» надо понимать в смысле, что поле графика заполнено векторами.

Для построения таких графиков в двумерной системе координат используется функция fieldplot:

fieldplot(f, r1, r2)

fieldplot(f, r1, r2. ...)

где f — вектор или множество векторов, задающих построение; r1 и r2 — пределы.

На рис. 12.4 в нижней части документа показан вид одного из таких графиков. Следует отметить, что для получения достаточного числа отчетливо видных стрелок надо поработать с форматированием графиков. Иначе графики этого типа могут оказаться не очень представительными. Так, слишком короткие стрелки превращаются в черточки и даже точки, не имеющие острия, что лишает графики наглядности.

Несколько позже мы рассмотрим построение на одном рисунке графиков плотности и векторного поля, а также создание более наглядных толстых стрелок.

1.7.Трехмерный график типа implicitplot3d

Трехмерные поверхности также могут задаваться уравнениями неявного вида. В этом случае для построения их графиков используется функция implicitplot3d:

implicitplot3d(exprl,x=a..b,y=c.,d,z=p..q,<options>)

implicitplot3d(f,a..b,c..d,p..q,<options>)

На рис. 12.5 показаны два примера построения объемных фигур с помощью функции implicitplot3d.

Эти примеры хорошо иллюстрируют технику применения функции implicitplot3d. С ее помощью можно строить весьма своеобразные фигуры, что, впрочем, видно и из приведенных примеров. Для наглядности фигур на рис. 12.5 они несколько развернуты в пространстве с помощью мыши.

1.8.Графики в разных системах координат

В пакете plots имеется множество функций для построения графиков в различных системах координат. Объем книги не позволяет воспроизвести примеры всех видов таких графиков, ибо их многие сотни. Да это и не надо — во встроенных в справочную систему примерах можно найти все нужные сведения. Так что ограничимся лишь парой примеров применения функции tubeplot(C, options), позволяющей строить весьма наглядные фигуры в пространстве, напоминающие трубы или иные объекты, образованные фигурами вращения.

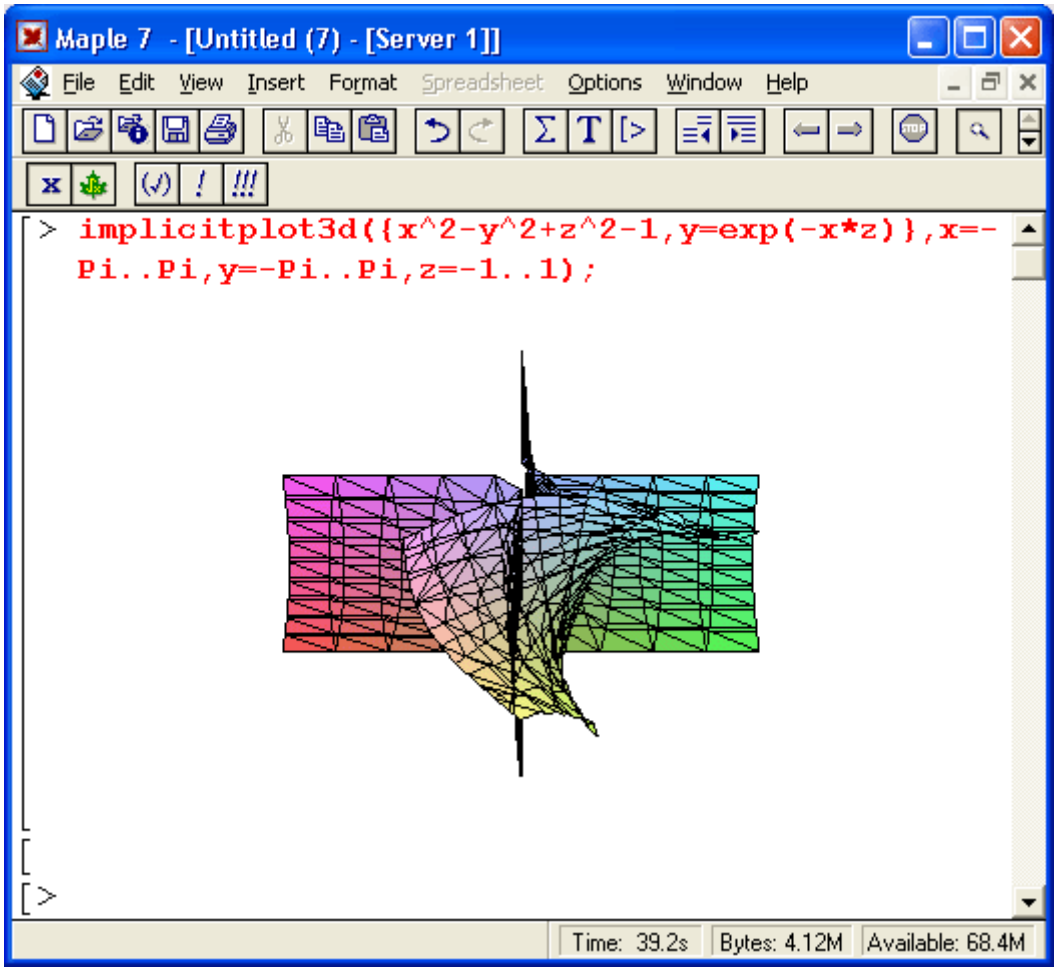


Рис. 12.5. Примеры применения функции implidtplot3d

На рис. 12.6 показана одна из таких фигур. Она поразительно напоминает раковину улитки. Функциональная окраска достигнута доработкой графика с помощью панели форматирования.

Эта функция может использоваться и для построения ряда трубчатых объектов в пространстве. При этом автоматически задается алгоритм удаления невидимых линий даже для достаточно

сложных фигур. Это наглядно иллюстрирует пример на рис. 12.7, показывающий фигуру «цепи». Не правда ли, реалистичность этой фигуры поражает воображение?

Можно долго размышлять о том, как те или иные математические закономерности описывают предметы реального мира, положенные в основу тех или иных геометрических объектов, или, возможно, о гениальности людей, сумевших найти такие закономерности для многих из таких объектов. В наше время Maple 7 открывает огромные возможности для таких людей.

1.9.Графики типа трехмерного поля из векторов

Наглядность ряда графиков можно существенно увеличить, строя их в трехмерном представлении. Например, для такого построения графиков полей из векторов можно использовать графическую функцию fieldplot3d. В отличие от функции fieldplot она строит стрелки как бы в трехмерном пространстве (рис. 12.8).

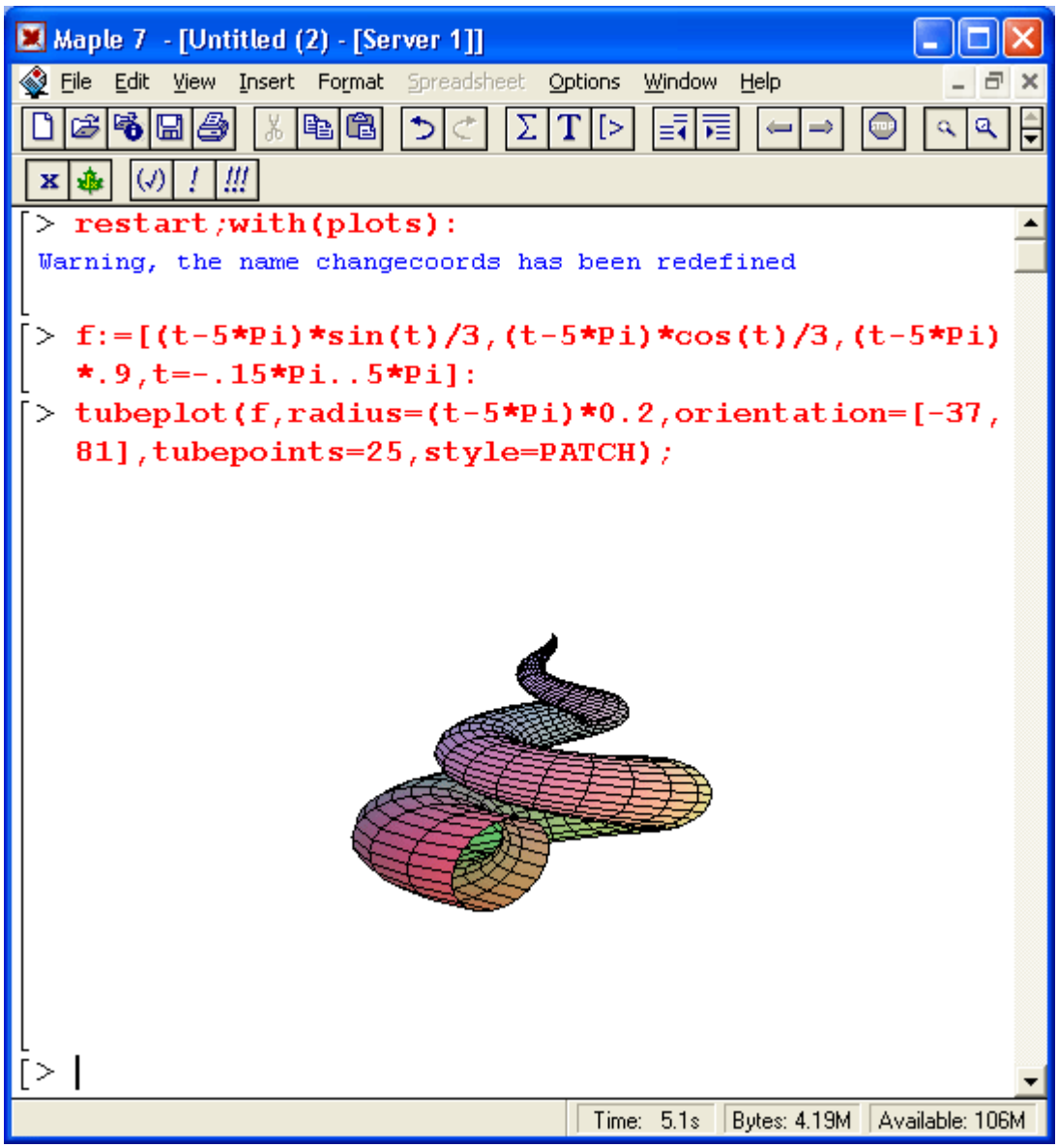


Рис. 12.6. Построение графика- «улитки»

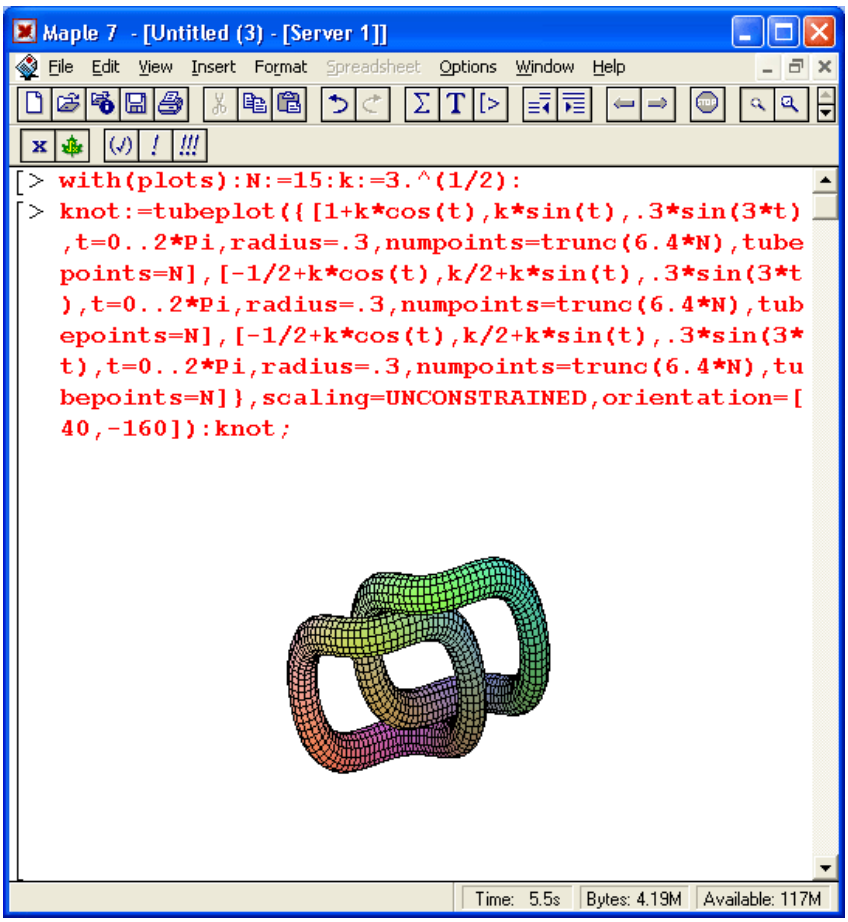


Рис. 12.7. Фигура «цепи», построенная с применением функции tubeplot

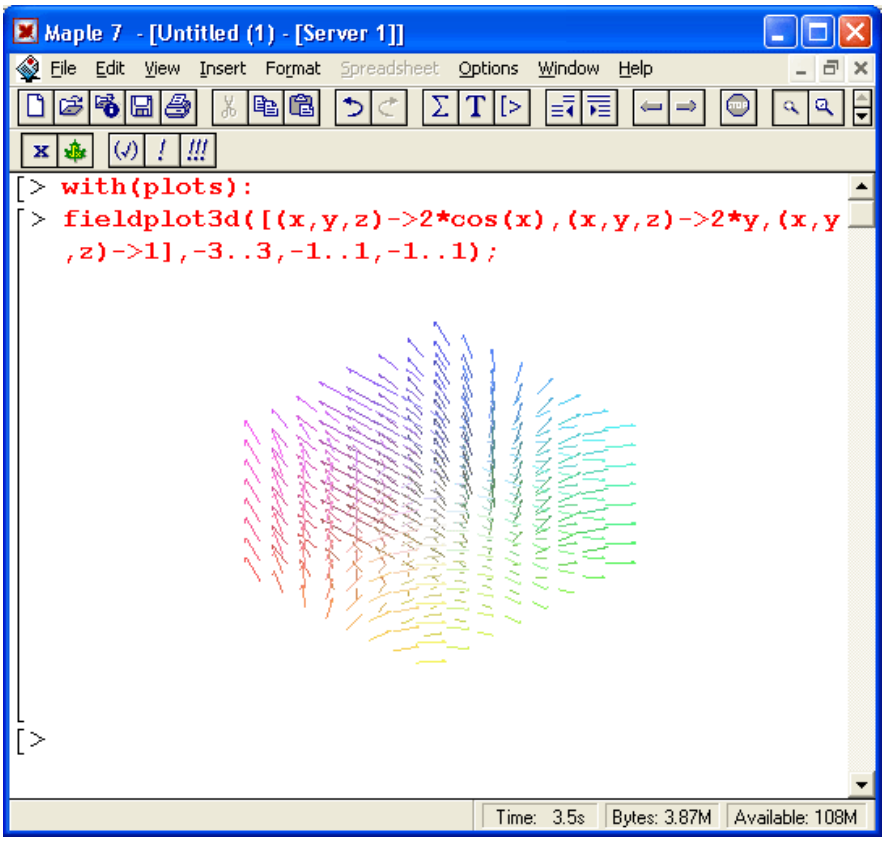


Рис. 12.8. Построение поля в трехмерном пространстве с помощью векторов

Все сказанное об особенностях таких двумерных графиков остается справедливым и для графиков трехмерных. В частности, для обеспечения достаточной наглядности нужно тщательно отлаживать форматы представления таких графиков.

1.10.Контурные трехмерные графики

В отличие от векторных графиков контурные графики поверхностей, наложенные на сами эти поверхности, нередко повышают восприимчивость таких поверхностей — подобно изображению линий каркаса. Для одновременного построения поверхности и контурных линий на них служит функция contourplot3d. Пример ее применения показан на рис. 12.9.

Для повышения наглядности этот график доработан с помощью контекстной панели инструментов графиков. В частности, включена функциональная окраска и подобраны углы обзора фигуры, при которых отчетливо видны ее впадина и пик.

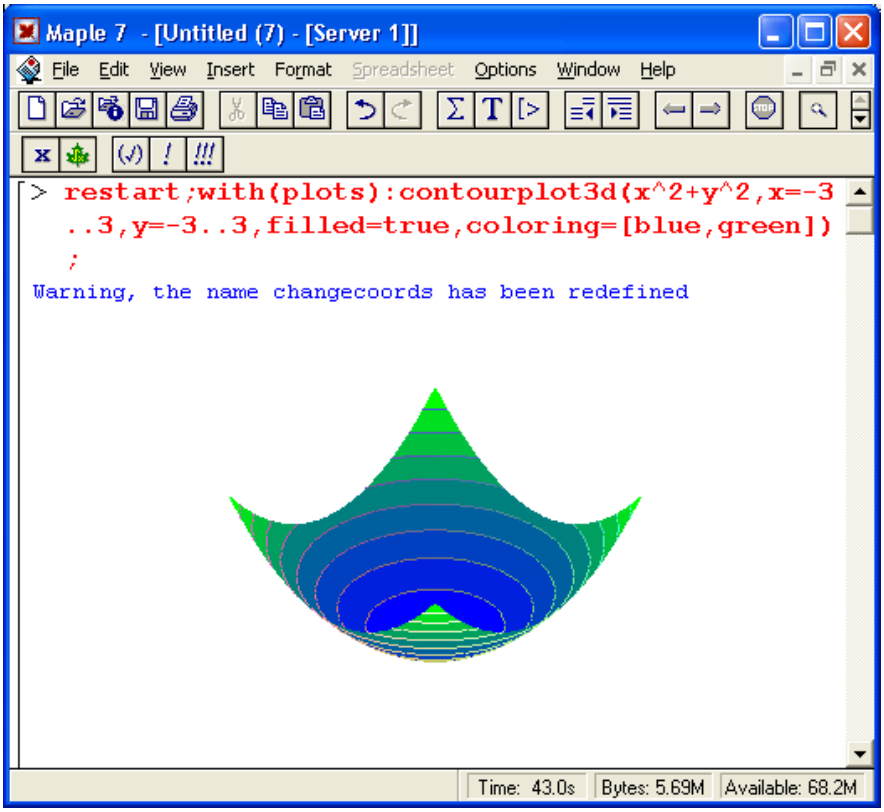


Рис. 12.9. График поверхности с контурными линиями

1.11.Техника визуализации сложных пространственных фигур

Приведенные выше достаточно простые примеры дают представление о высоком качестве визуализации геометрических фигур с помощью пакета plots. Здесь мы рассмотрим еще несколько примеров визуализации трехмерных фигур. Многие видели катушки индуктивности, у которых провод того или иного диаметра намотан на тороидальный магнитный сердечник. Некую математическую абстракцию такой катушки иллюстрирует рис. 12.10.

В документе рис. 12.10 для функции tubeplot использовано довольно большое число параметров. Не всегда их действие очевидно. Поэтому на рис. 12.11 показано построение трех взаимно пересекающихся торов с разными наборами параметров. Этот рисунок дает также наглядное представление о возможности построения нескольких графических объектов (представленных функциями pi, p2 и p3) с помощью функции tubeplot.

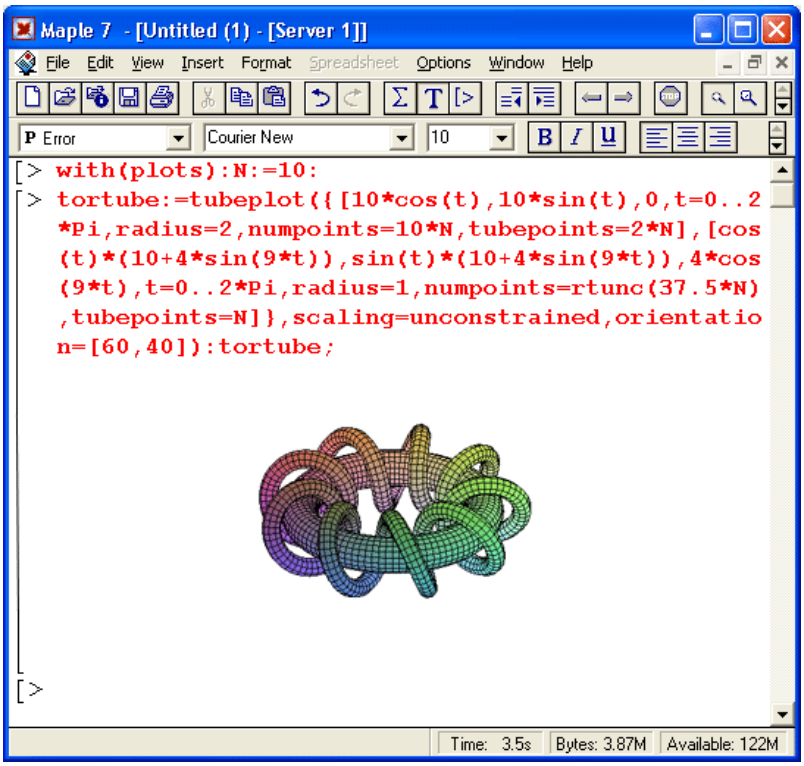


Рис. 12.10. Тор с обмоткой — толстой спиралью

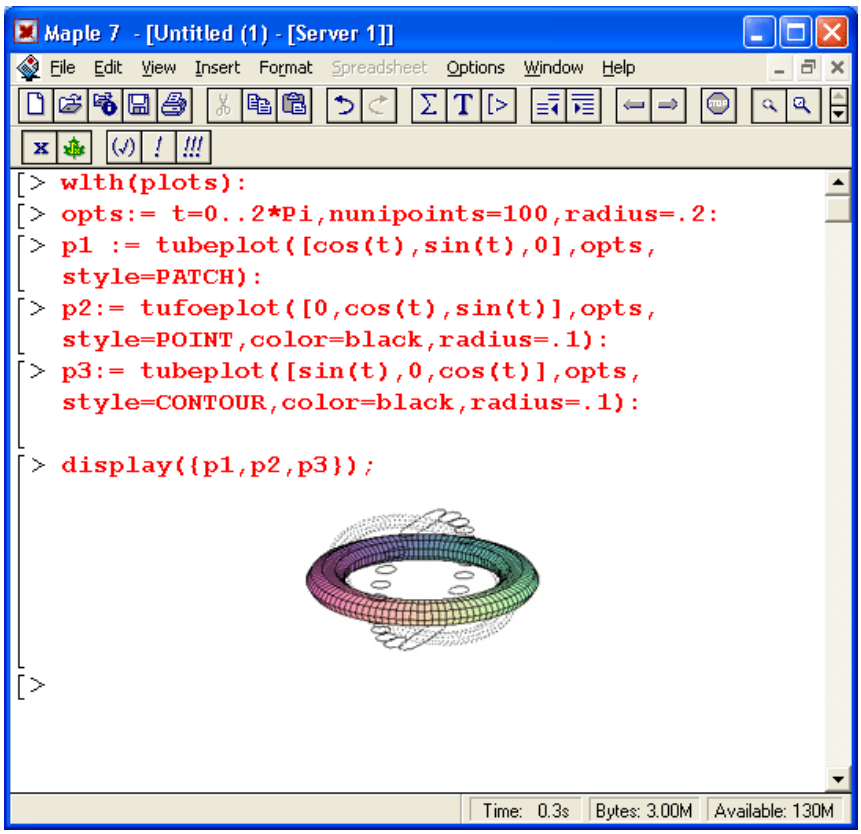


Рис. 12.11. Три пересекающихся тора с разными стилями построения

Наконец, на рис. 12.12 показано построение тора с тонкой обмоткой. Рекомендуется внимательно посмотреть на запись функции tubeplot в этом примере и в примере, показанном на рис. 12.11. Можно также поэкспериментировать с управляющими параметрами графика, от которых сильно зависят его представительность и наглядность.

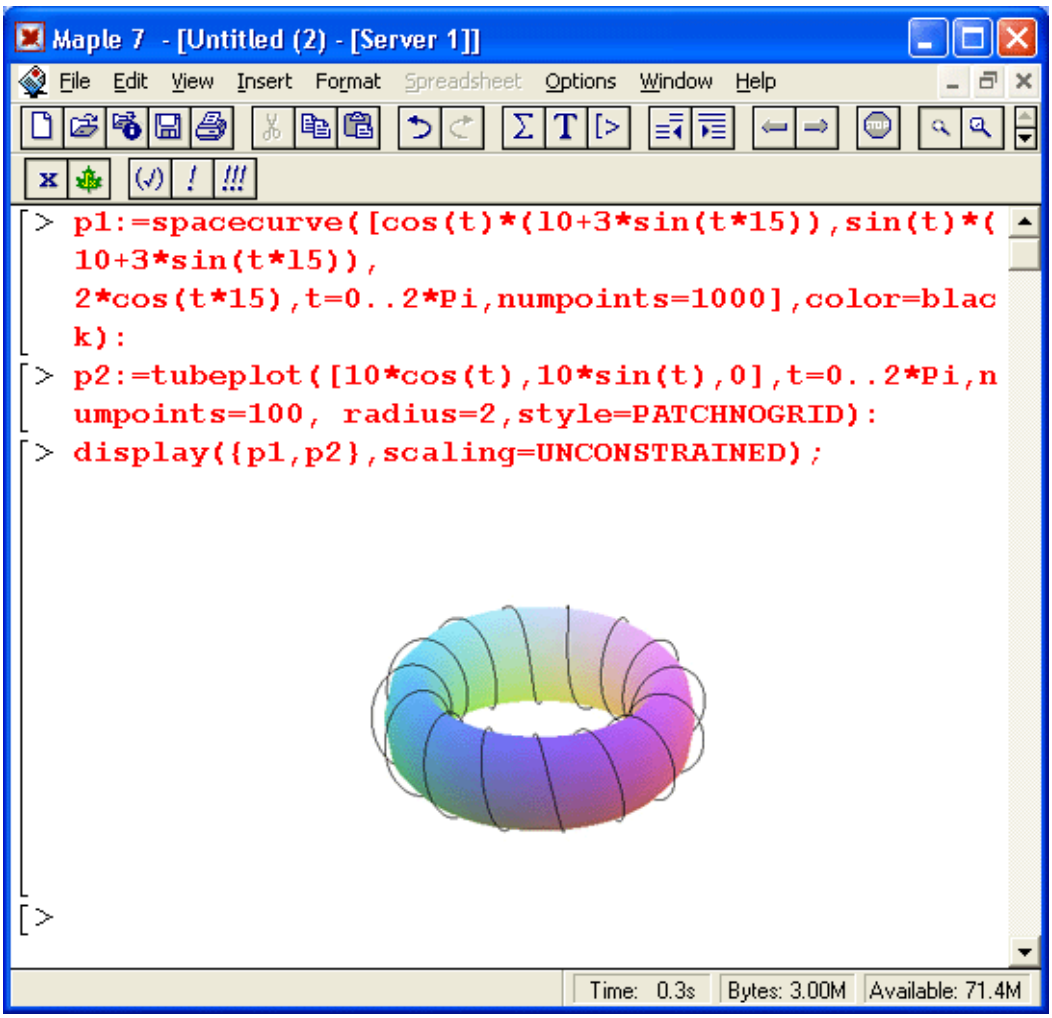


Рис. 12.12. Тор с тонкой обмоткой

В ряде случаев наглядно представленные фигуры можно строить путем объединения однотипных фигур. Пример графика подобного рода представлен на рис. 12.13. Здесь готовится список графических объектов s, смещенных по вертикали. С помощью функции display они воспроизводятся на одном графике, что повышает реалистичность изображения.

Последний пример имеет еще одну важную особенность — он иллюстрирует задание графической процедуры, в теле которой используются функции пакета plots. Параметр n этой процедуры задает число элементарных фигур, из которых строится полная фигура. Таким образом, высотой фигуры (или шириной «шины») можно управлять. Возможность задания практически любых графических процедур средствами Maple-языка существенно расширяет возможности Maple.

Наглядность таких графиков, как графики плотности и векторных полей может быть улучшена их совместным применением. Такой пример показан на рис. 12.14.

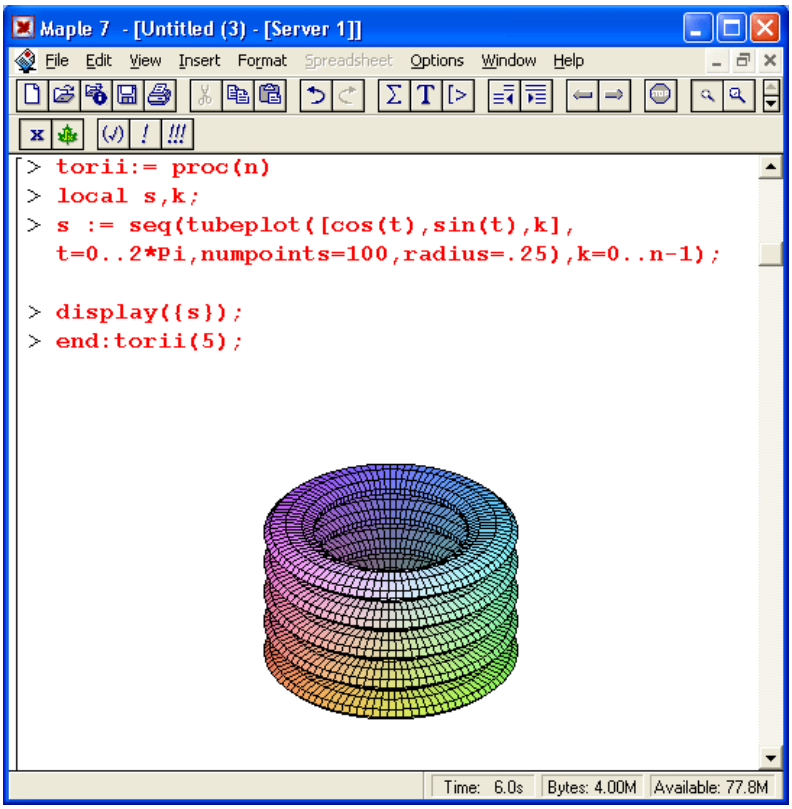


Рис. 12.13. Построение фигуры, напоминающей шину автомобиля

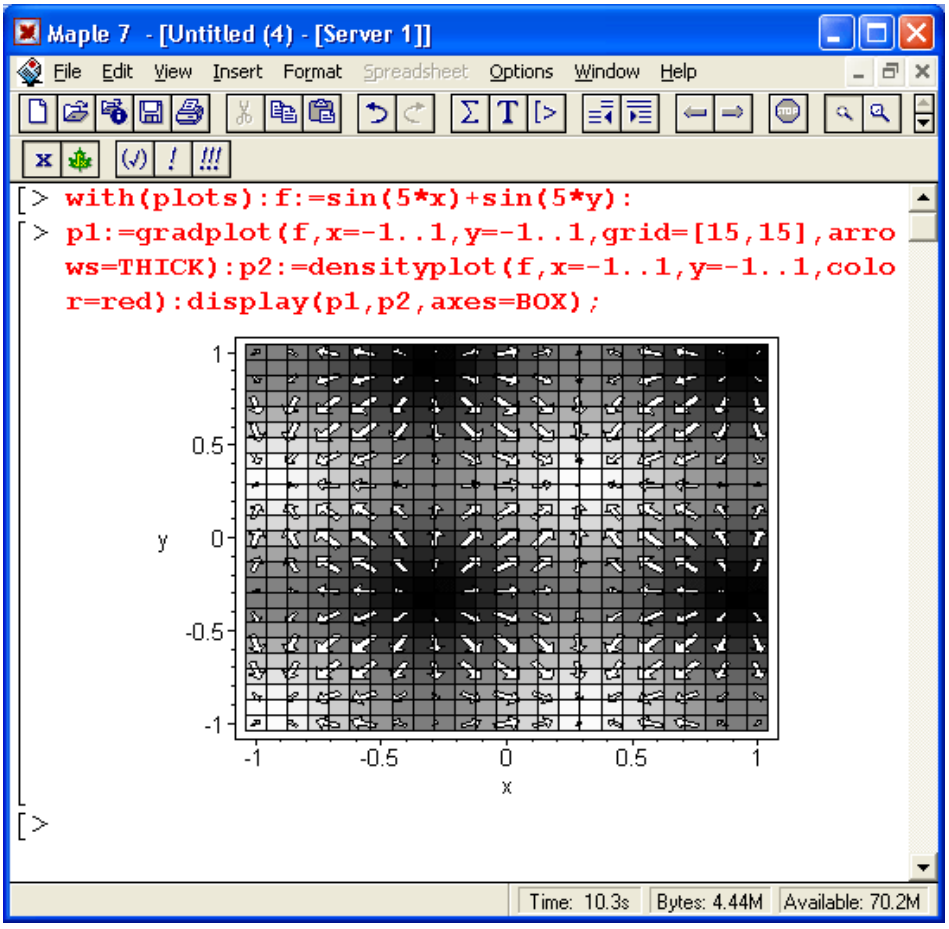


Рис. 12.14. Пример совместного применения графиков плотности и векторного поля

Этот пример иллюстрирует использование «жирных*- стрелок .для обозначения векторного поля. Наглядность графика повышается благодаря наложению стрелок на график плотности, который лучше, чем собственно стрелки, дает представление о плавности изменения высоты поверхности, заданной функцией.

2.Техника анимирования графиков

2.1.Анимация двумерных графиков

Визуализация графических построений и результатов моделирования различных объектов и явлений существенно повышается при использовании средств «оживления» "(анимации) изображений. Пакет plots имеет две простые функции для создания анимированных графиков.

Первая из этих функций служит для создания анимации графиков, представляющих функцию одной переменной F(x):

animatecurve(F, r, ...)

Эта функция просто позволяет наблюдать медленное построение графика. Формат ее применения подобен используемому в функции plot. При вызове данной функции вначале строится пустой шаблон графика. Если активизировать шаблон мышью, то в строке главного меню появляется меню Animation. Меню Animation содержит команды управления анимацией. Такое же подменю появляется и в контекстном (рис. 12.15). Указанное подменю содержит следующие команды анимации:

- Play — запуск построения графика;
- Next — выполнение следующего шага анимации;
- Backward/Forward — переключение направления анимации (назад/вперед);
- Faster — ускорение анимации;
- Slower — замедление анимации;
- Continiuus/Singlecycle — цикличность анимации.

При исполнении команды Play происходит построение кривой (или нескольких кривых). В зависимости от выбора команд Faster или Slower построение идет быстро или медленно. Команда Next выполняет один шаг анимации -построение очередного фрагмента кривой. Переключатель Backward/Forward позволяет задать направление построения кривой - от начала к концу или от конца к началу. Построение может быть непрерывным или циклическим в зависимости от состояния позиции Continiuus/Singlecycle в подменю управления анимацией. При циклической анимации число циклов задается параметром frames=n.

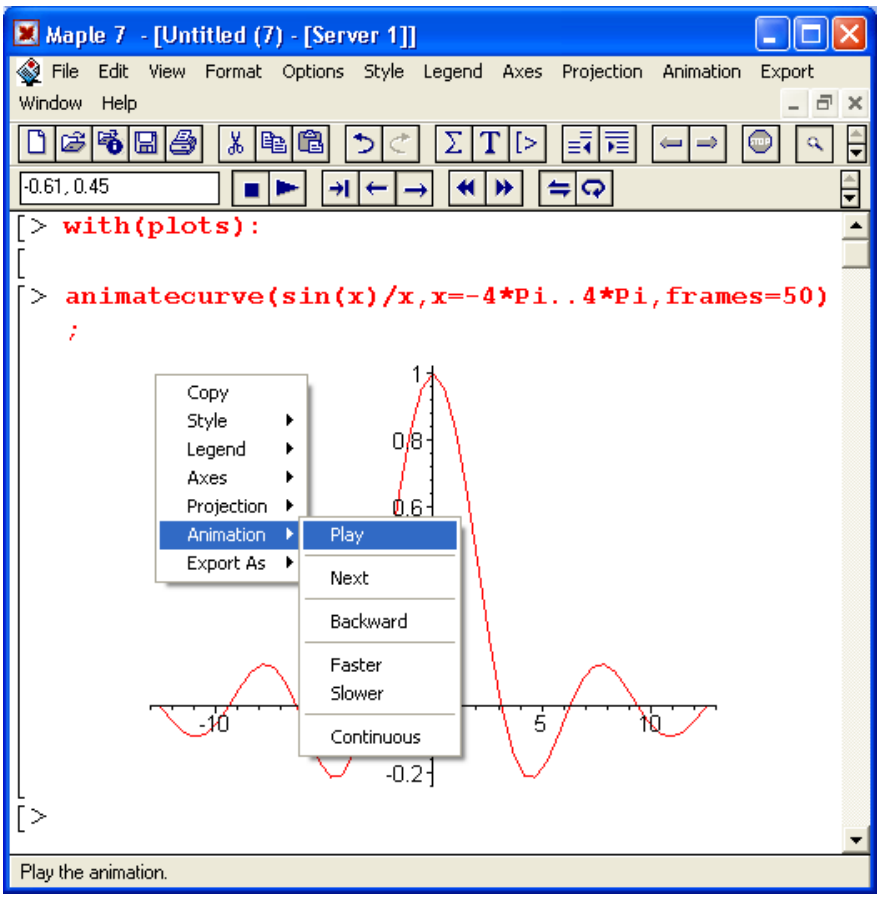


Рис. 12.15. Пример анимационного построения графика функцией animatecurve

2.2.Проигрыватель анимированной графики

При включенном выводе панели форматирования во время анимации она приобретает вид панели проигрывателя клипов (рис. 12.15). Эта панель имеет кнопки управления с обозначениями, принятыми у современных магнитофонов:

1. Поле координат перемещающейся точки графика.

2. Остановка анимации.

3. Пуск анимации.

4. Переход к следующему кадру (фрейму).

5. Установка направления анимации от конца в начало.

6. Установка направления анимации из начала в конец (по умолчанию).

7. Уменьшение времени шага анимации.

8. Увеличение времени шага анимации.

9. Установка одиночного цикла анимации.

10. Установка серии циклов анимации.

Итак, кнопки проигрывателя, по существу, повторяют команды подменю управления анимацией.

Нажав кнопку пуска (с треугольником, острием обращенным вправо), можно наблюдать изменение вида кривой для функции sin(x)/(x). Другие кнопки управляют характером анимации. Проигрыватель дает удобные средства для демонстрации анимации, например, во время занятий со школьниками или студентами.

2.3.Построение двумерных анимированных графиков

Более обширные возможности анимации двумерных графиков обеспечивает функция animate:

ammate(F, x, t)

animate(F,x, t, o)

В ней параметр x задает пределы изменения переменной x, а параметр t — пределы изменения дополнительной переменной t. Суть анимации при использовании данной функции заключается в построении серии кадров (как в мультфильме), причем каждый кадр связан со значением изменяемой во времени переменной t. Если надо явно задать число кадров анимации N, то в качестве o следует использовать frame=N.

Рисунок 12.16 показывает применение функции animate.

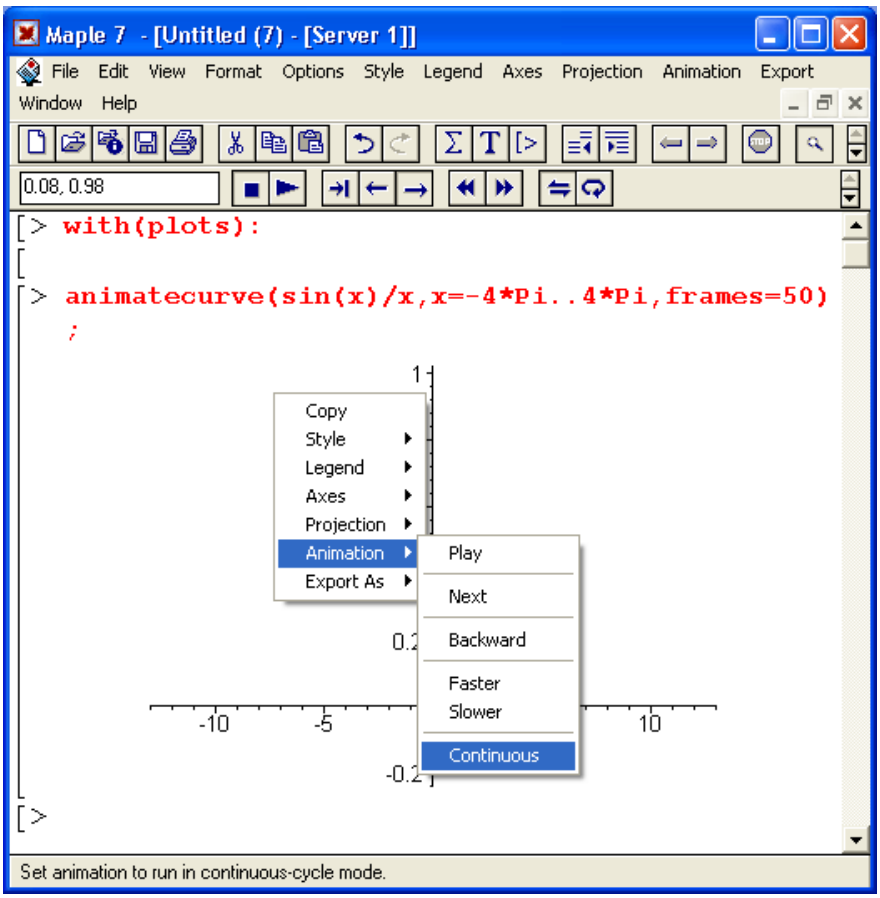


Рис. 12.16. Анимация функции sin(i*x)/(i*x) на фоне неподвижной синусоиды

В документе рис. 12.16 строятся две функции — не создающая анимации функция sin(x) и создающая анимацию функция sin(i*x)/(i*x), причем в качестве переменной t задана переменная i. Именно ее изменение и создает эффект анимации. Проигрыватель анимационных клипов и меню, описанные выше, могут использоваться для управления и этим видом анимации. Обратите внимание на вызов графических функций в этом примере командой with и на синтаксис записи этих функций.

К сожалению, картинки в книгах всегда неподвижны и воспроизвести эффект анимации невозможно. Можно лишь представить несколько текущих кадров анимации. Представленная на рис. 12.16 картина соответствует последнему кадру анимации.

Еще один пример анимации представлен на рис. 12.17. Этот документ показывает кадр анимированного процесса улучшения приближения синусоидальной функции рядом с различным числом членов (и порядком последнего члена ряда). Результирующая картина, изображенная на рис. 12.17, показывает как

приближаемую синусоидальную функцию, так и графики всех рядов, которые последовательно выводятся в ходе анимации.

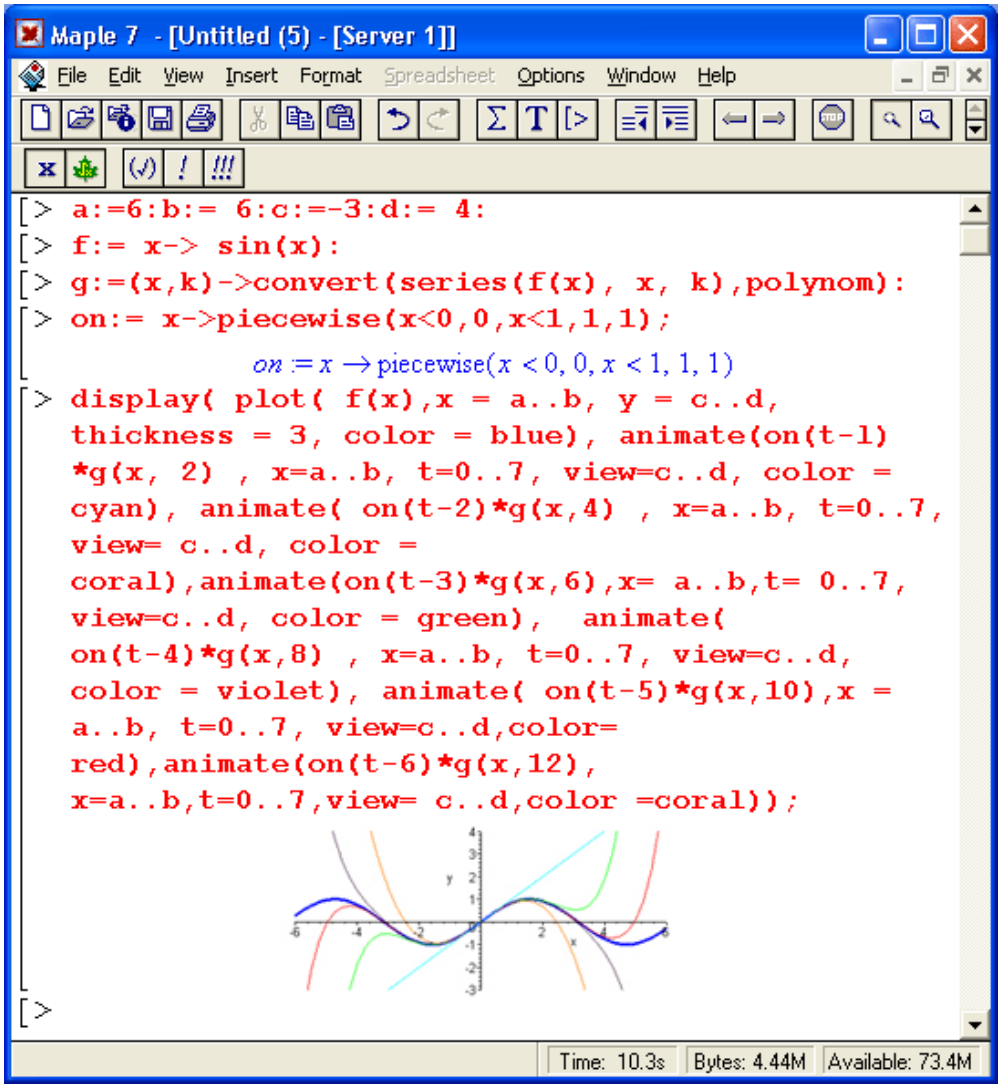


Рис. 12.17. Анимационная демонстрация приближения синусоиды рядом с меняющимся числом членов

Анимация графиков может найти самое широкое применение при создании учебных материалов. С ее помощью можно акцентировать внимание на отдельных параметрах графиков и образующих их функций и наглядно иллюстрировать характер их изменений.

2.4.Построение трехмерных анимационных графиков

Аналогичным образом может осуществляться и анимирование трехмерных фигур. Для этого используется функция animate3d:

animate3d(F,x, y,t,o)

Здесь F — описание функции (или функций); x, y и t — диапазоны изменения переменных x, y и t Для задания числа кадров N надо использовать необязательный параметр o в виде frame=N.

На рис. 12.18 показано построение анимированного графика. После задания функции, график которой строится, необходимо выделить график и запустить проигрыватель, как это описывалось для анимации двумерных графиков.

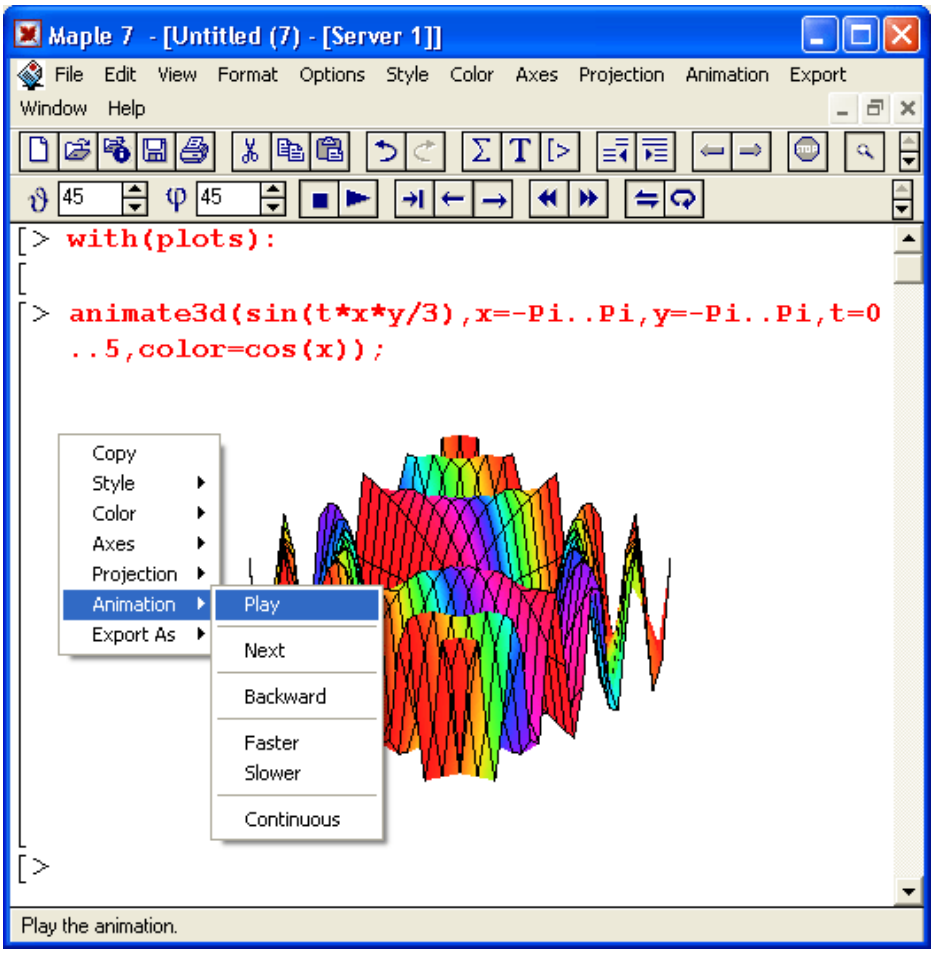


Рис. 12.18. Подготовка трехмерного анимационного графика

На рис. 12.18 показано также контекстное меню поля выделенного графика. Нетрудно заметить, что с помощью этого меню (и содержащихся в нем подменю) можно получить доступ к параметрам трехмерной графики и выполнить необходимые операции форматирования, такие как включение цветовой окраски, выбор ориентации фигуры и т. д.

Назначение параметров, как и средств управления проигрывателем анимационных клипов, было описано выше.

2.5.Анимация с помощью параметра insequence

Еще один путь получения анимационных рисунков — создание ряда графических объектов pi, p2, p3 и т. д. и их последовательный вывод с помощью функций display или display3d:

display (pl,p2.p3.....insequence=true)

display3d (pi. p2. p3..., i nsequence=t rue)

Здесь основным моментом является применение параметра insequence=true. Именно он обеспечивает вывод одного за другим серии графических объектов pi, p2, p3 и т. д. При этом объекты появляются по одному и каждый предшествующий объект стирается перед появлением нового объекта.

3.Графика пакета plottools

3.1.Примитивы пакета plottools

Инструментальный пакет графики plottools служит для создания графических примитивов, строящих элементарные геометрические объекты на плоскости и в пространстве: отрезки прямых и дуг, окружности, конусы, кубики и т. д. Его применение позволяет разнообразить графические построения и строить множество графиков специального назначения. В пакет входят следующие графические примитивы:

arc	arrow	circle	cone	cuboid
curve	cutln	cutout	cylinder	disk
dodecahedron	ellipse	ellipticArc	hemisphere	hexahedron
hyperbola	icosahedron	line	octahedron	pieslice
point tetrahedron	polygon torus	rectangle	semi torus	sphere

ПРИМЕЧАНИЕ

Вызов перечисленных примитивов осуществляется после загрузки пакета в память компьютера командой with(plottools). Только после этого примитивы пакета становятся доступными. Обычно

примитивы используются для задания графических объектов, которые затем выводятся функцией display. Возможно применение этих примитивов совместно с различными графиками.

Большинство примитивов пакета plottools имеет довольно очевидный синтаксис. Например, для задания дуги используется примитив arc(c, r, a..b,...), где c — список с координатами центра окружности, к которой принадлежит дуга, r — радиус этой окружности, a.. b — диапазон углов. На месте многоточия могут стоять обычные параметры, задающие цвет дуги, толщину ее линии и т. д. Конус строится примитивом cone(c,r,h...), где c — список с координатами центра, r — радиус основания, h — высота и т. д. Все формы записи графических примитивов и их синтаксис можно найти в справочной системе. В необходимых случаях стоит проверить синтаксис того или иного примитива с помощью справки по пакету plottools.

3.2.Примеры применения двумерных примитивов пакета plottools

На рис. 12.19 показано применение нескольких примитивов двумерной графики для построения дуги, окружности, закрашенного красным цветом эллипса и отрезка прямой. Кроме того, на графике показано построение синусоиды. Во избежание искажений пропорций фигур надо согласовывать диапазон изменения переменной x. Обычно параметр scalling=constrained выравнивает масштабы и диапазоны по осям координат, что гарантирует отсутствие искажений у окружностей и других геометрических фигур. Однако при этом размеры графика нередко оказываются малыми. Напоминаем, что- этот параметр можно задать и с помощью подменю Projection.

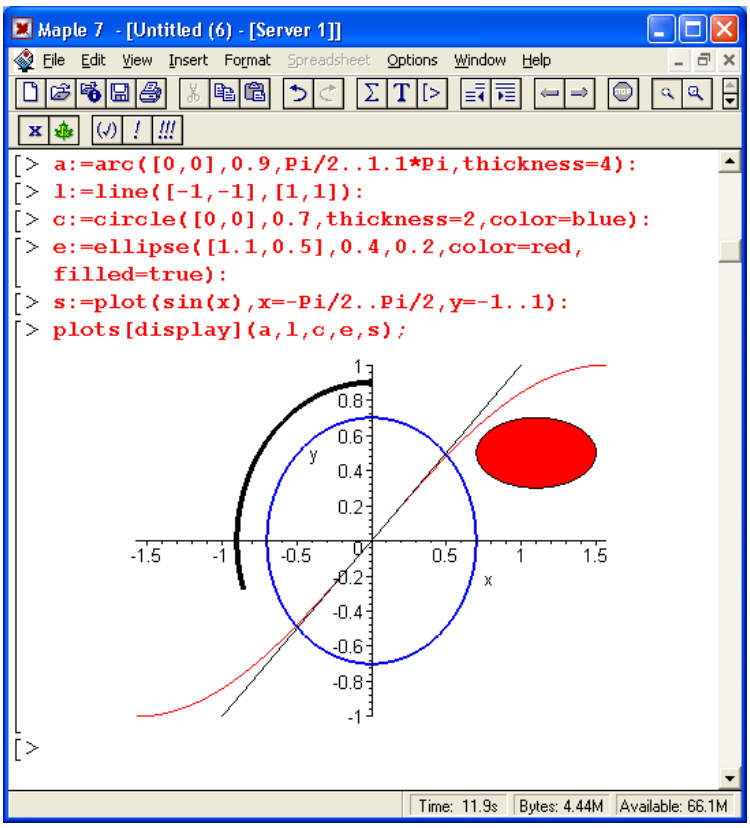


Рис. 12.19. Примеры применения примитивов двумерной графики пакета plottools

Рисунок 12.20 иллюстрирует построение средствами пакета plottools четырех разноцветных стрелок, направленных в разные стороны. Цвет стрелок задан списком цветов c, определенным после команды загрузки пакета. Для построения стрелок используется примитив arrow с соответствующими параметрами.

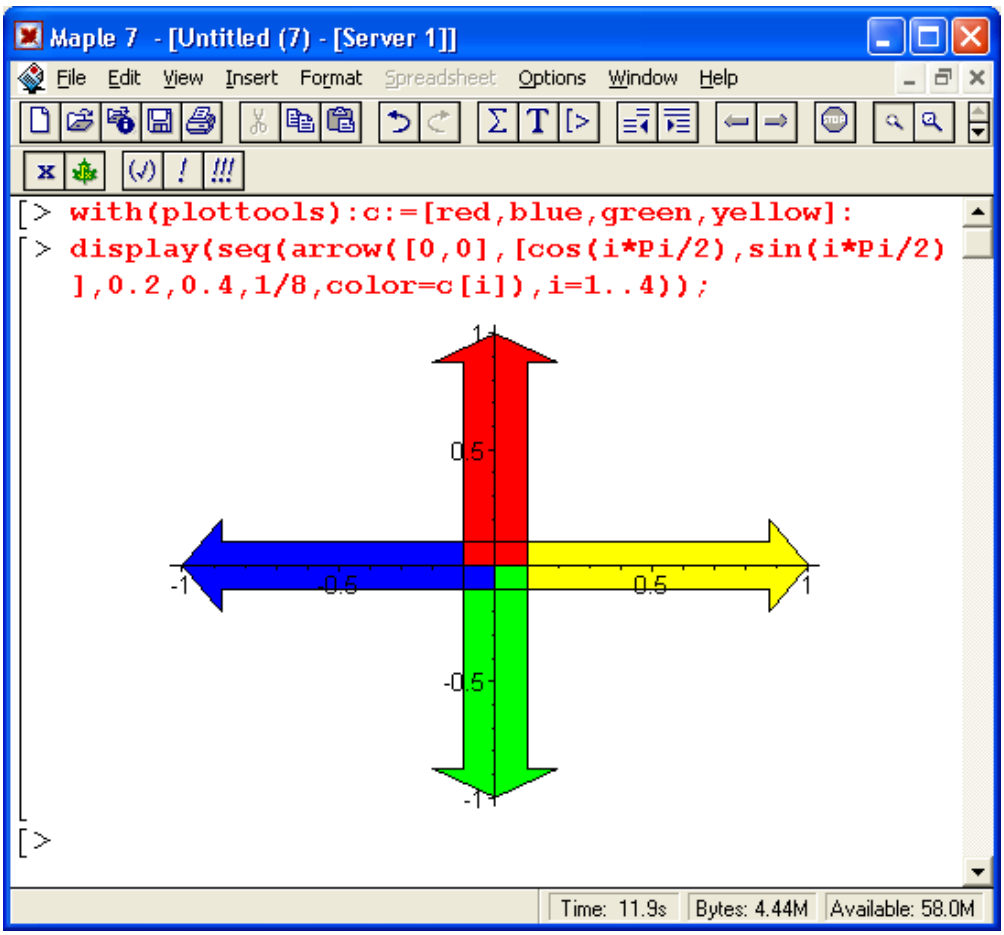


Рис. 12.20. Построение разноцветных стрелок, направленных в разные стороны

Примитивы могут использоваться в составе графических процедур, что позволяет конструировать практически любые типы сложных графических объектов. В качестве примера на рис. 12.21 представлена процедура SmithChart, которая строит хорошо известную электрикам диаграмму Смита (впрочем, несколько упрощенную). В этой процедуре используется примитив построения дуг arc. При этом задается верхняя часть диаграммы, а нижняя получается ее зеркальным отражением.

ПРИМЕЧАНИЕ

Обратите внимание на то, что, начиная с рис. 12.21, мы не указываем загрузку пакета plottools, поскольку она уже была проведена ранее. Однако надо помнить, что все примеры этого раздела предполагают, что такая загрузка обеспечена. Если вы использовали команду restart или только что загрузили систему Maple 7, то для исполнения примера рис. 12.21 и последующих примеров надо исполнить команду with(plottools).

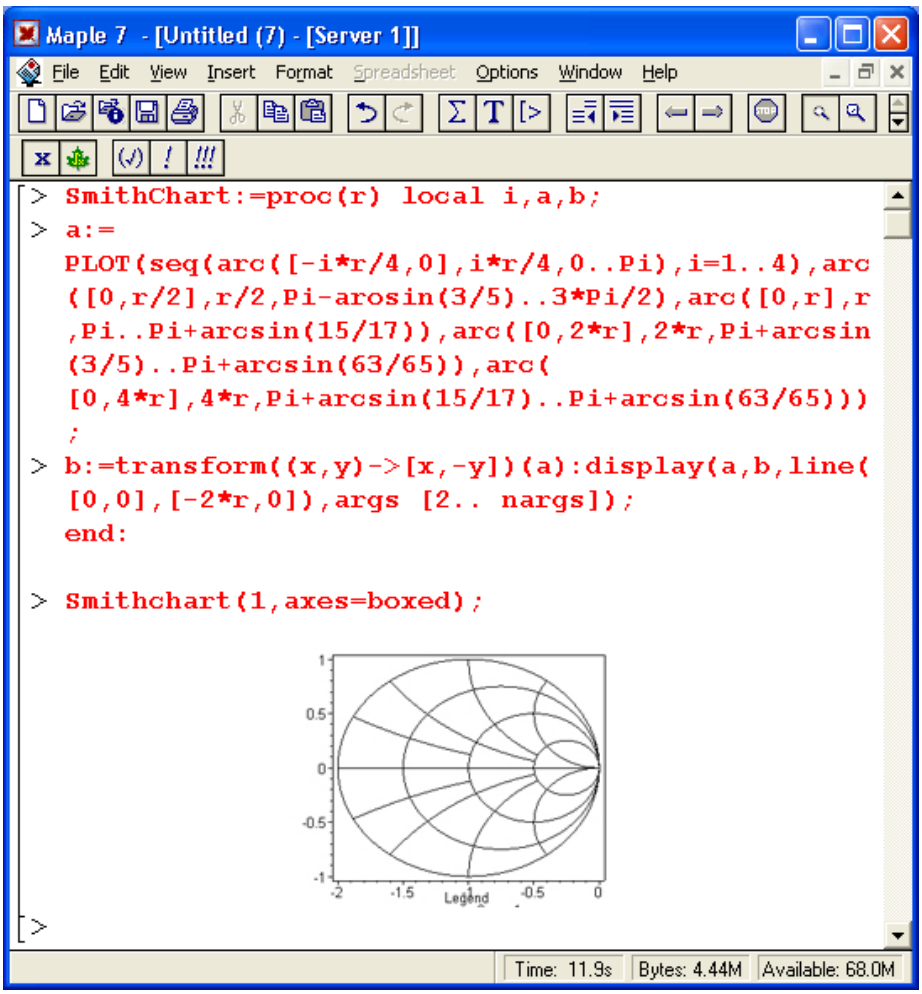


Рис. 12.21. Построение диаграммы Смита

3.3.Примеры применения трехмерных примитивов пакета plottools

Аналогичным описанному выше образом используются примитивы построения трехмерных фигур. Это открывает возможность создания разнообразных иллюстрационных рисунков и графиков, часто применяемых при изучении курса стереометрии. Могут строиться самые различные объемные фигуры и поверхности — конусы, цилиндры, кубы, полиэдры и т. д. Использование средств функциональной окраски делает изображения очень реалистичными.

Рисунок 12.22 показывает построение цилиндра и двух граненых шаров. Цилиндр строится примитивом cylinder, а граненые шары — примитивом icosahedron.

Другой пример (рис. 12.23) иллюстрирует построение на одном графике двух объемных фигур, одна из которых находится внутри

другой фигуры. Этот пример демонстрирует достаточно корректное построение вложенных фигур. На рис. 12.24 показано совместное построение двух пересекающихся кубов и сферы в пространстве. Нетрудно заметить, что графика пакета приблизительно (с точностью до сегмента) вычисляет области пересечения фигур. С помощью контекстно-зависимого меню правой кнопки мыши (оно показано на рис. 12.24) можно устанавливать условия обзора фигур, учитывать перспективу при построении и т. д. В частности, фигуры на рис. 12.24 показаны в перспективе.

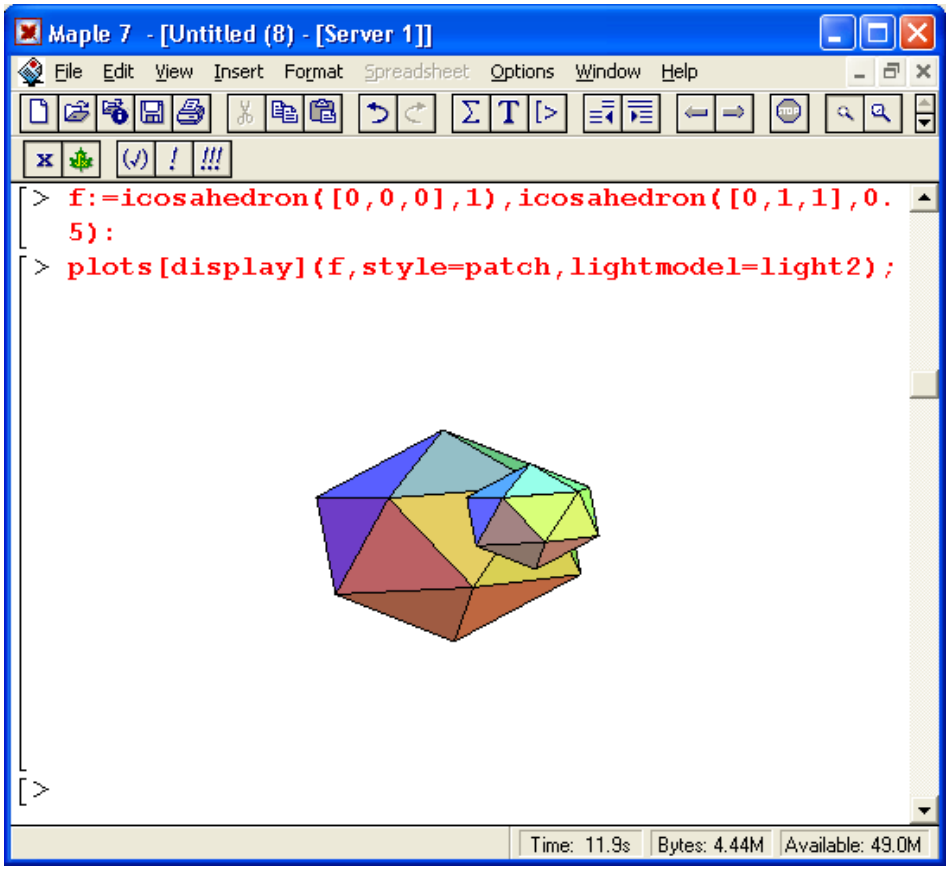


Рис. 12.22. Построение цилиндра и двух граненых шаров

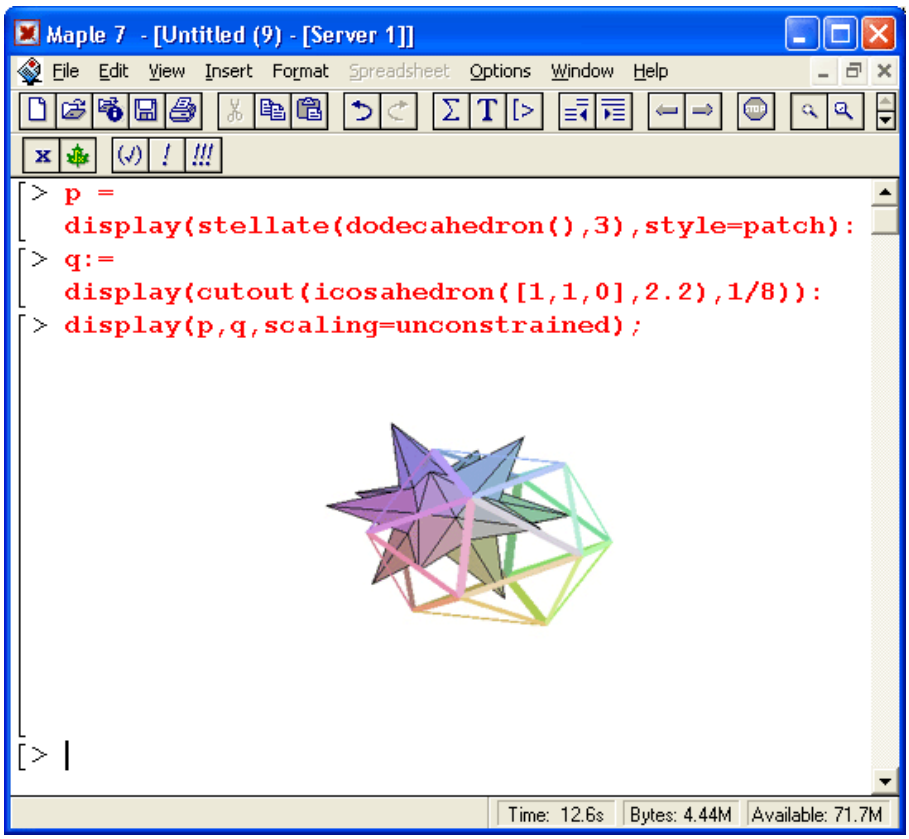


Рис. 12.23. Построение двух объемных фигур

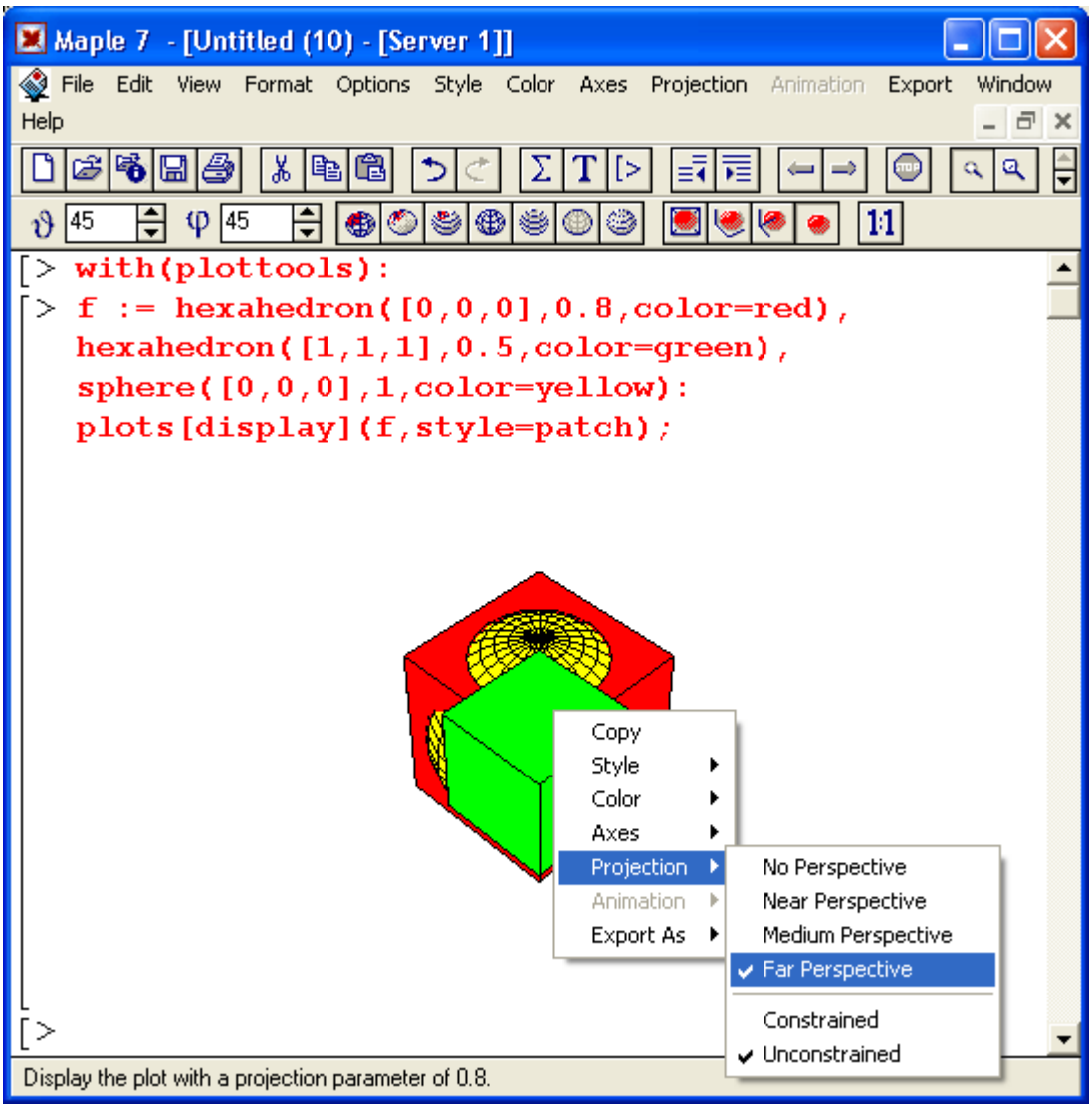


Рис. 12.24. Примеры применения примитивов трехмерной графики пакета plottools

Построение еще одной забавной трехмерной фигуры — «шкурки ежа» — демонстрирует рис. 12.25. В основе построения лежит техника создания полигонов. Построение фигур, очень напоминающих улитки, показано на рис. 12.26. При построении этих фигур используется функция tubeplot. Обратите внимание на то, что строятся две входящие друг в друга «улитки».

Наконец, на рис. 12.27 показано построение фигуры — бутылки Клейна. Фигура задана рядом своих фрагментов, определенных в процедуре cleinpoints. Эта процедура является еще одним наглядным примером программирования графических построений с помощью Maple-языка.

С другими возможностями этого пакета читатель теперь справится самостоятельно или с помощью данных справочной системы.

Много примеров построения сложных и красочных фигур с применением пакета plottools можно найти в Интернете на сайте фирмы Maple Software, в свободно распространяемой библиотеке пользователей системы Maple и в книгах по этой системе.

3.4.Построение графиков из множества фигур

В ряде случаев бывает необходимо строить графики, представляющие собой множество однотипных фигур. Для построения таких графиков полезно использовать функцию повторения seq(f ,i=a. .b). На рис. 12.28 показано построение фигуры, образованной вращением прямоугольника вокруг одной из вершин.

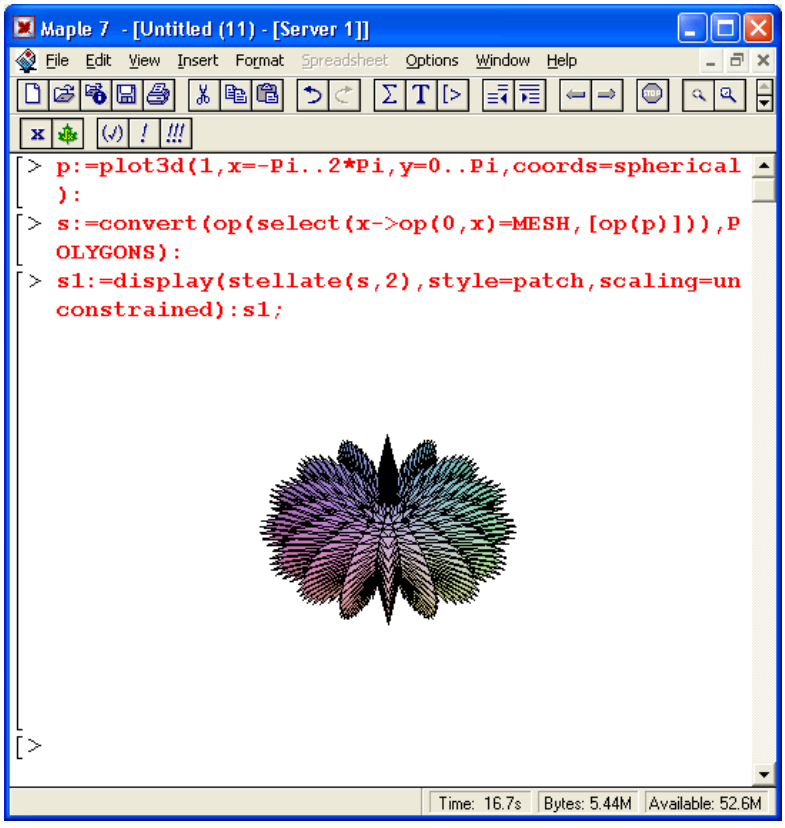


Рис. 12.25. Построение трехмерной фигуры — «шкурки ежа»

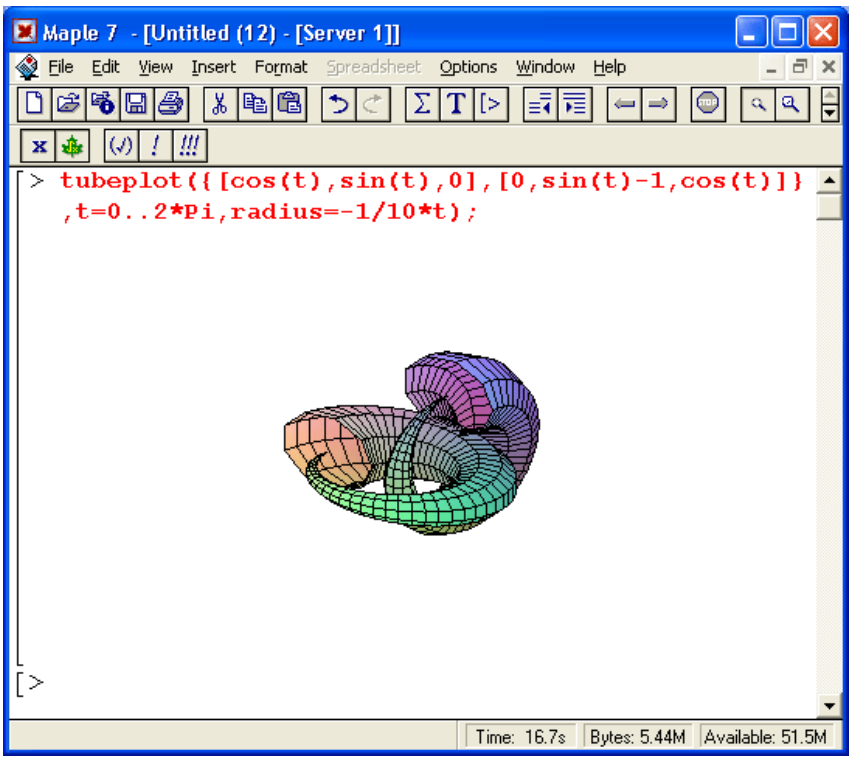


Рис. 12.26. Построение фигуры «улитка»

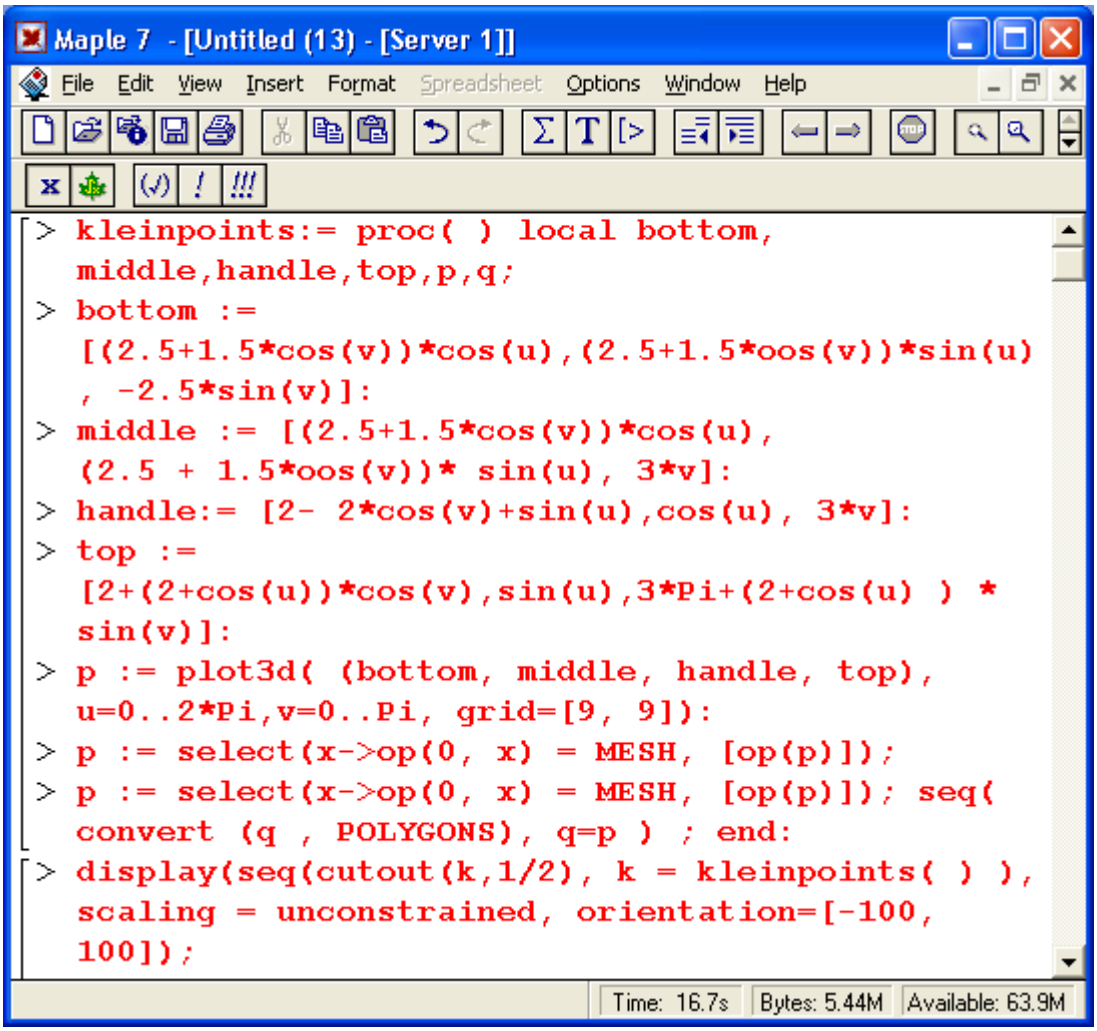


Рис. 12.27. Построение фигуры «бутылка Клейна»

В этом примере полезно обратить внимание еще и на функцию поворота фигуры — rotate. Именно сочетание этих двух функций (мультиплицирования и поворота базовой фигуры — прямоугольника) позволяет получить сложную фигуру, показанную на рис. 12.28.

3.5.Анимация двумерной графики в пакете plottools

Пакет plottools открывает возможности реализации анимационной графики. Мы ограничимся одним примером анимации двумерных графиков. Этот пример представлен на рис. 12.29. В этом примере показана анимационная иллюстрация решения дифференциального уравнения, описывающего незатухающий колебательный процесс. Строится качающийся объект — стрелка с острием вправо, решение дифференциального уравнения в виде синусоиды и большая стрелка с острием влево, которая соединяет текущую точку графика синусоиды с острием стрелки колеблющегося объекта.

Этот пример наглядно показывает возможности применения анимации для визуализации достаточно сложных физических и математических закономерностей. Перспективы применения системы Maple 7 в создании виртуальных физических и иных лабораторий трудно переоценить.

3.6.Анимация трехмерной графики в пакете plottools

Хорошим примером 3D-анимации является документ, показанный на рис. 12.30. Представленная на нем процедура springPlot имитирует поведение упругой системы, первоначально сжатой, а затем выстреливающей шар, установленный на ее верхней пластине. Упругая система, состоит из неподвижного основания, на котором расположена упругая масса (например, из пористой резины), и верхней пластины.

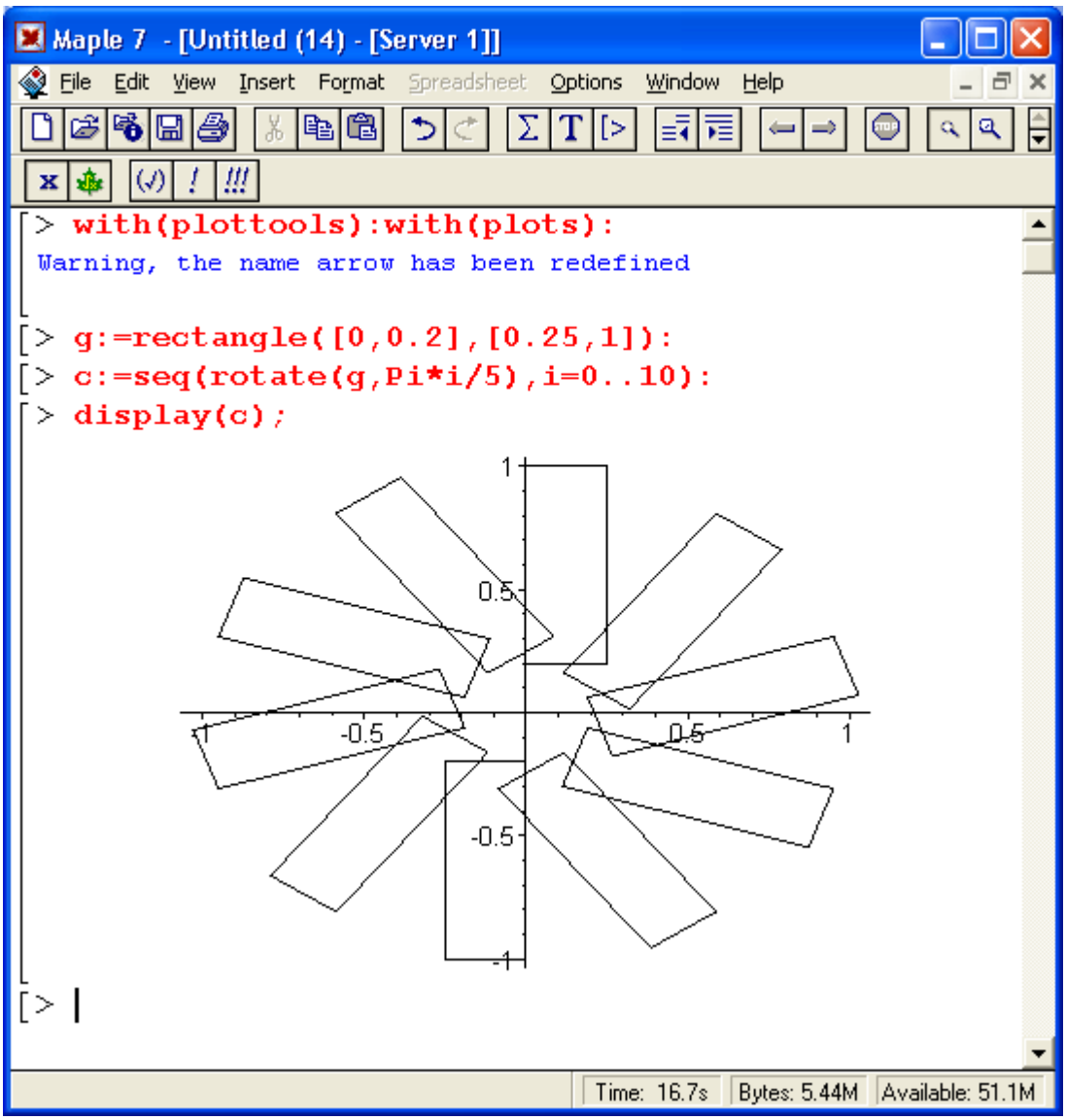


Рис. 12.28. Построение фигуры, образованной вращением прямоугольника

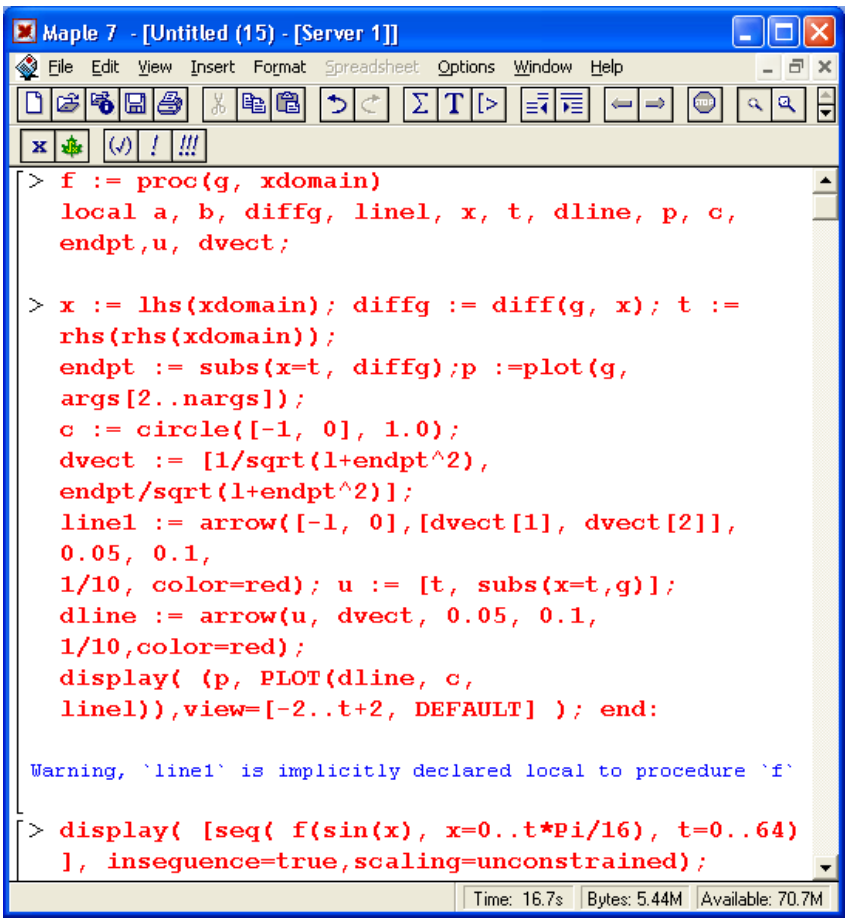


Рис.12.29. Пример анимации двумерной графики

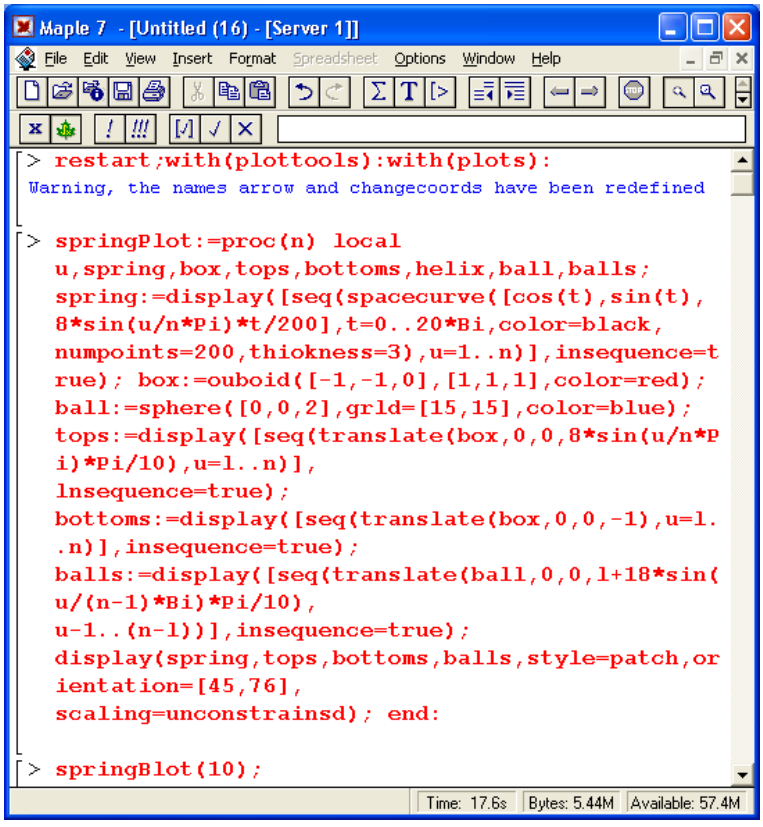


Рис. 12.30. Имитация отстрела шара сжатой упругой системой

Управление анимацией, реализованной средствами пакета plottools, подобно уже описанному ранее. Последний пример также прекрасно иллюстрирует возможности применения Maple 7 при математическом моделировании различных явлений, устройств и систем.

4.Расширенные средства графической визуализации

4.1.Построение ряда графиков, расположенных по горизонтали

Обычно если в строке ввода задается построение нескольких графиков, то в строке вывода все они располагаются по вертикали. Это не всегда удобно, например, при снятии копий экрана с рядом графиков, поскольку экран монитора вытянут по горизонтали, а не по вертикали. Однако при применении функций plots и display можно разместить ряд двумерных графиков в строке вывода по горизонтали. Это демонстрирует пример, показанный на рис. 12.31. Этот пример достаточно прост и нагляден, так что читатель может пользоваться данной возможностью всегда, когда ему это нужно.

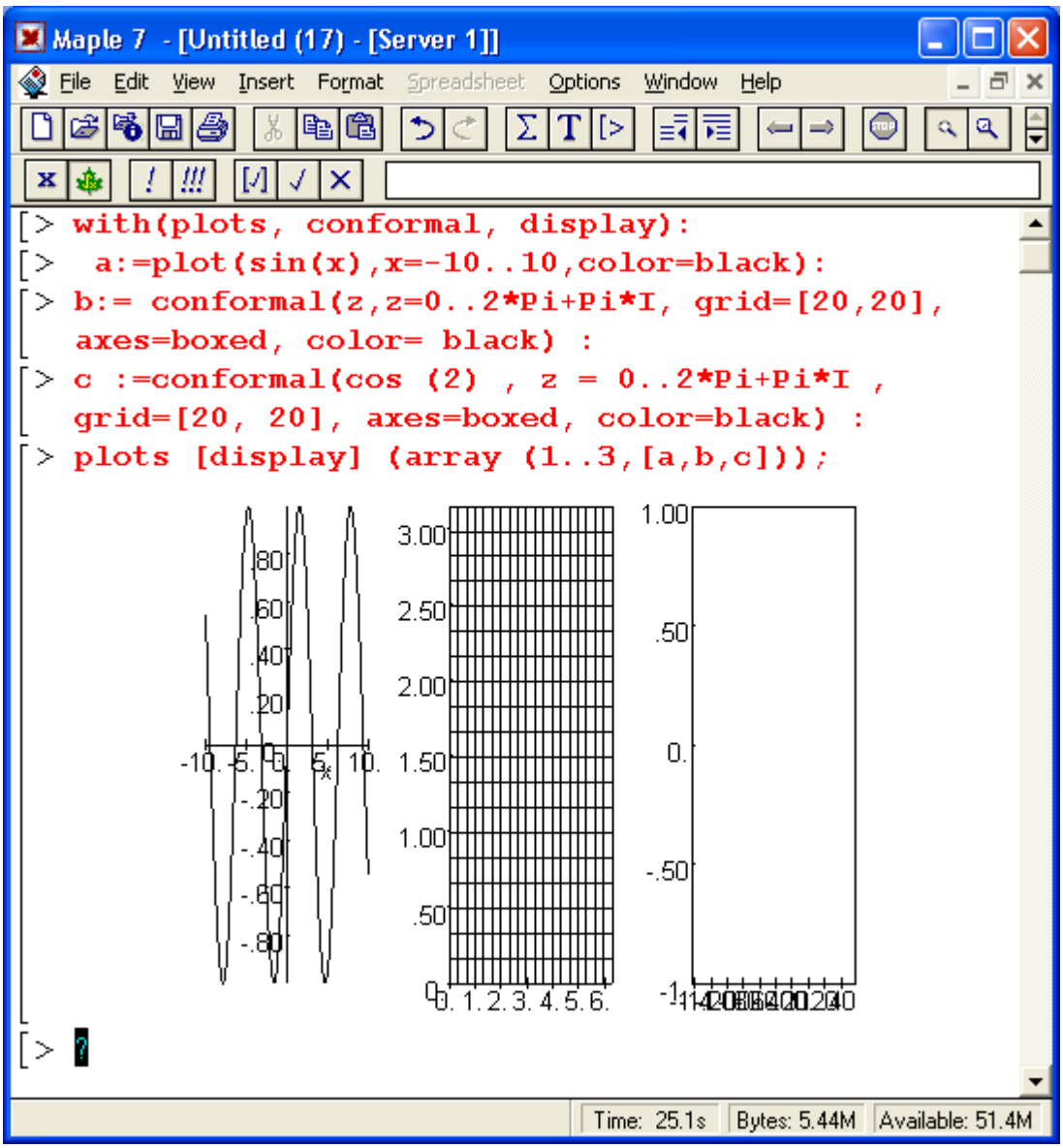


Рис. 12.31. Пример расположения трех графиков в строке вывода по горизонтали

4.2.Визуализация решения систем линейных уравнений

Мы уже не раз использовали графические возможности Maple для визуализации решений математических задач. Так, многие особенности даже функций одной переменной вида f(x) могут быть выявлены с помощью графика этой функции. Затем можно точно вычислить корни функции (точки перехода через 0), экстремумы, "крутизну наклона (производную) в заданных точках и т. д. Еще более информативна в этом отношении трехмерная графика — для большинства функций двух переменных вида z(x, y) нужно очень богатое математическое воображение, чтобы представить их вид — особенно в одной из многих десятков координатных систем.

Однако некоторые виды графиков трудно представить себе даже при наличии такого воображения. В этом отношении Maple 7 предоставляет поистине уникальные возможности, обеспечивая простую и быструю визуализацию решений. Ниже мы рассмотрим несколько наиболее характерных примеров такой визуализации. Системы линейных уравнений могут решаться как с помощью функции solve, так и с помощью матричных методов. Замечательной возможностью функции solve является возможность решения относительно ограниченного числа переменных. Например, систему линейных уравнений с переменными x, y, z, t и v можно решить относительно только первых трех переменных x, y и г. При этом решения будут функциями относительно переменных t и v и можно будет построить наглядный график решения (рис. 12.32).

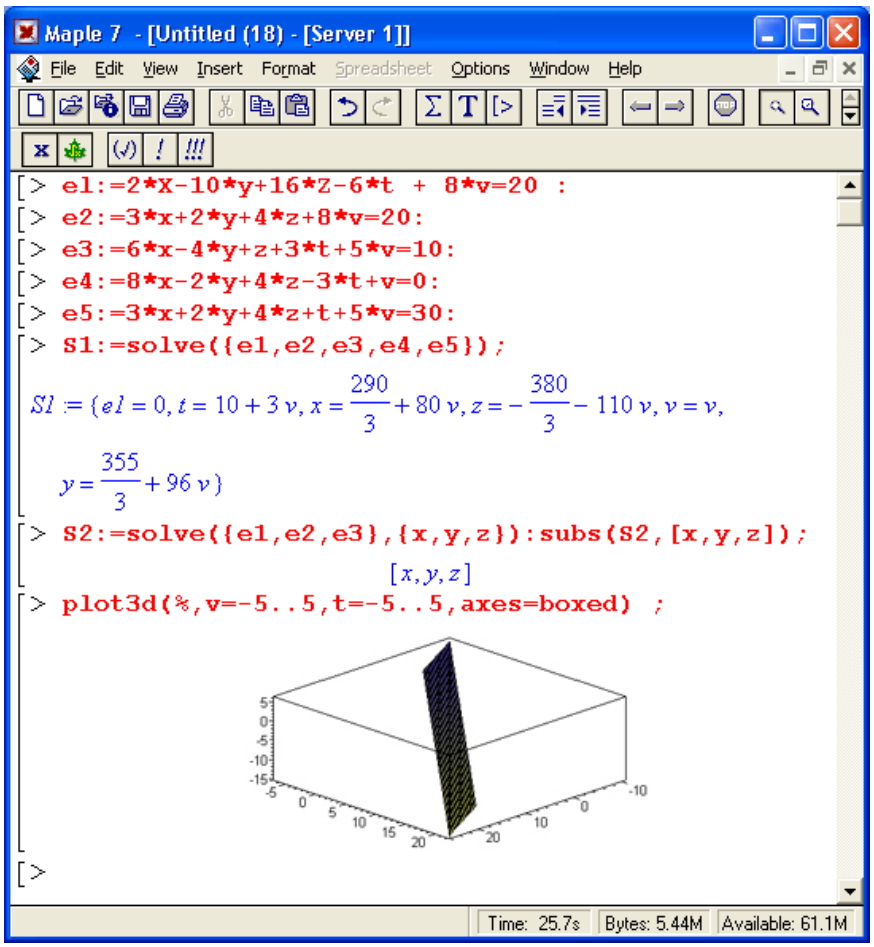


Рис. 12.32. График, представляющий решения системы линейных уравнений

На рис. 12.32 система задана пятью равенствами: el, e2, e3, e4 и e5. Затем функцией solve получено вначале решение для всех переменных (для иллюстрации), а затем для трех переменных x, y и

z. Для получения решения в виде списка, а не множества, как в первом случае для всех переменных, использована функция подстановки subs. После этого функция plot3d строит плоскость решения в пространстве.

4.3.Визуализация решения систем неравенств

Пожалуй, еще более полезным и наглядным средством является визуализация решения системы уравнений в виде неравенств. В пакете plots имеется специальная графическая функция inequal, которая строит все граничные линии неравенств и позволяет раскрасить разделенные ими области различными цветами:

inequal(ineqs, xspec, yspec, options)

Параметры этой функции следующие: ineqs — одно или несколько неравенств или равенств или список неравенств или равенств; xspec — xvar=min_x. .max_x; yspec — yvar=min_y. .max_y; o — необязательные параметры, например указывающие цвета линий, представляющих неравенства или равенства, и областей, образованных этими линиями и границами графика. Пример применения этой функции представлен на рис. 12,33.

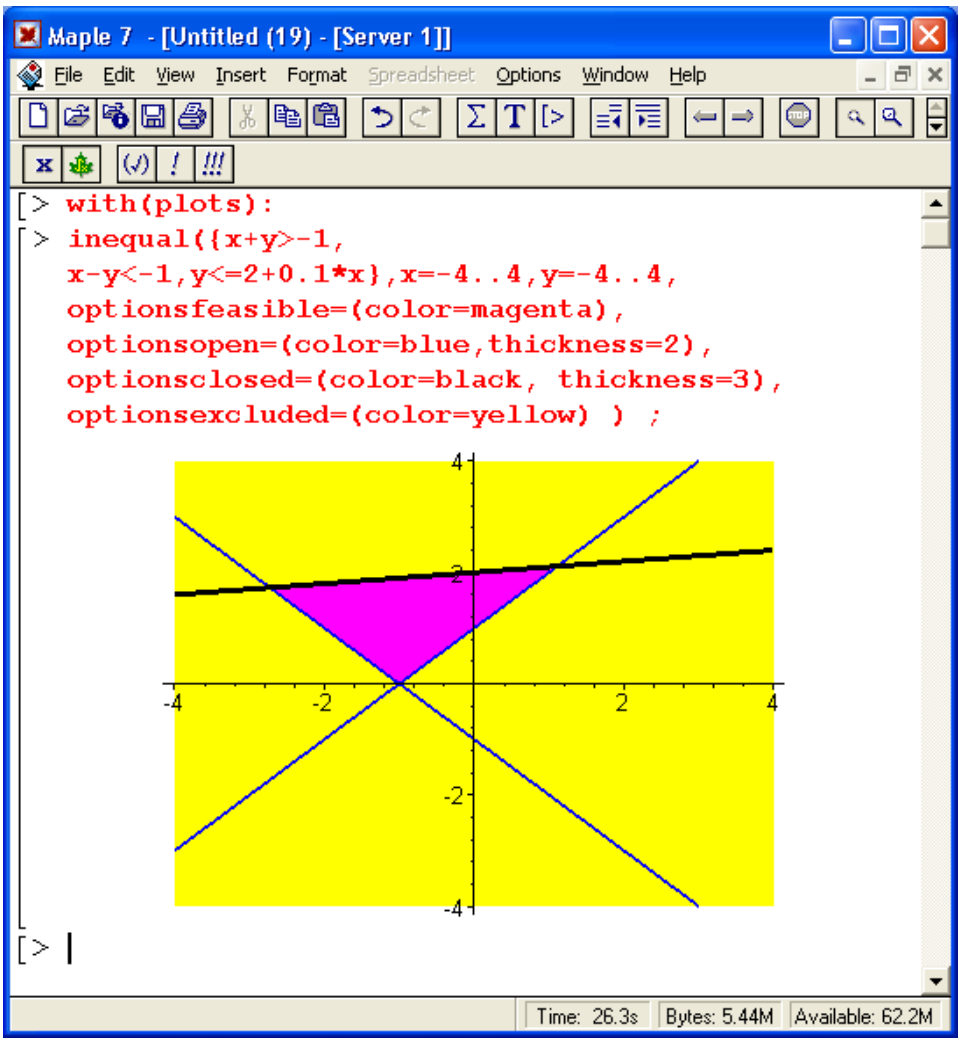


Рис. 12.33. Пример графической интерпретации решения системы неравенств

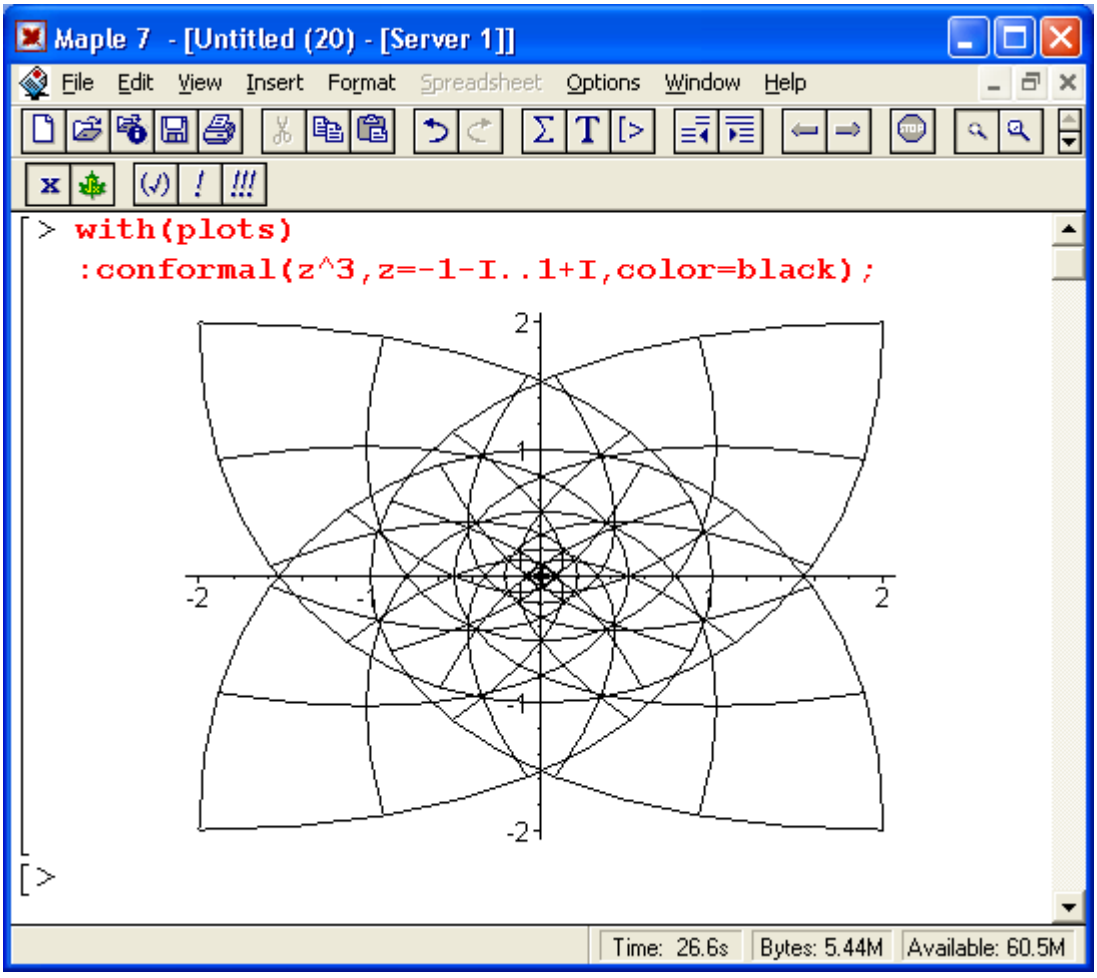
Обратите внимание на задание цветов: optionsfeasible задает цвет внутренней области, для которой удовлетворяются все неравенства (равенства), optionsopen и optionsdosed задают цвета открытых и закрытых границ областей графика, optionsexcluded используется для цвета внешних областей. График дает весьма наглядную интерпретацию действия ряда неравенств (или равенств).

4.4.Конформные отображения на комплексной плоскости

Объем данной книги не позволяет объяснить столь тонкое понятие, как конформные отображения на комплексной плоскости. Ограничимся лишь указанием на то, что в пакете plots имеется функция для таких отображений:

conformal(F,rl,r2,o);

где F — комплексная процедура или выражение; rl, г2 — области, задаваемые в виде a. .Ь или name=a. .b; o — управляющие параметры. Таким образом, для построения нужного графика достаточно задать нужное выражение и области изменения г! и г2. Пример построения конформных изображений для трех выражений дан на рис. 12.34.



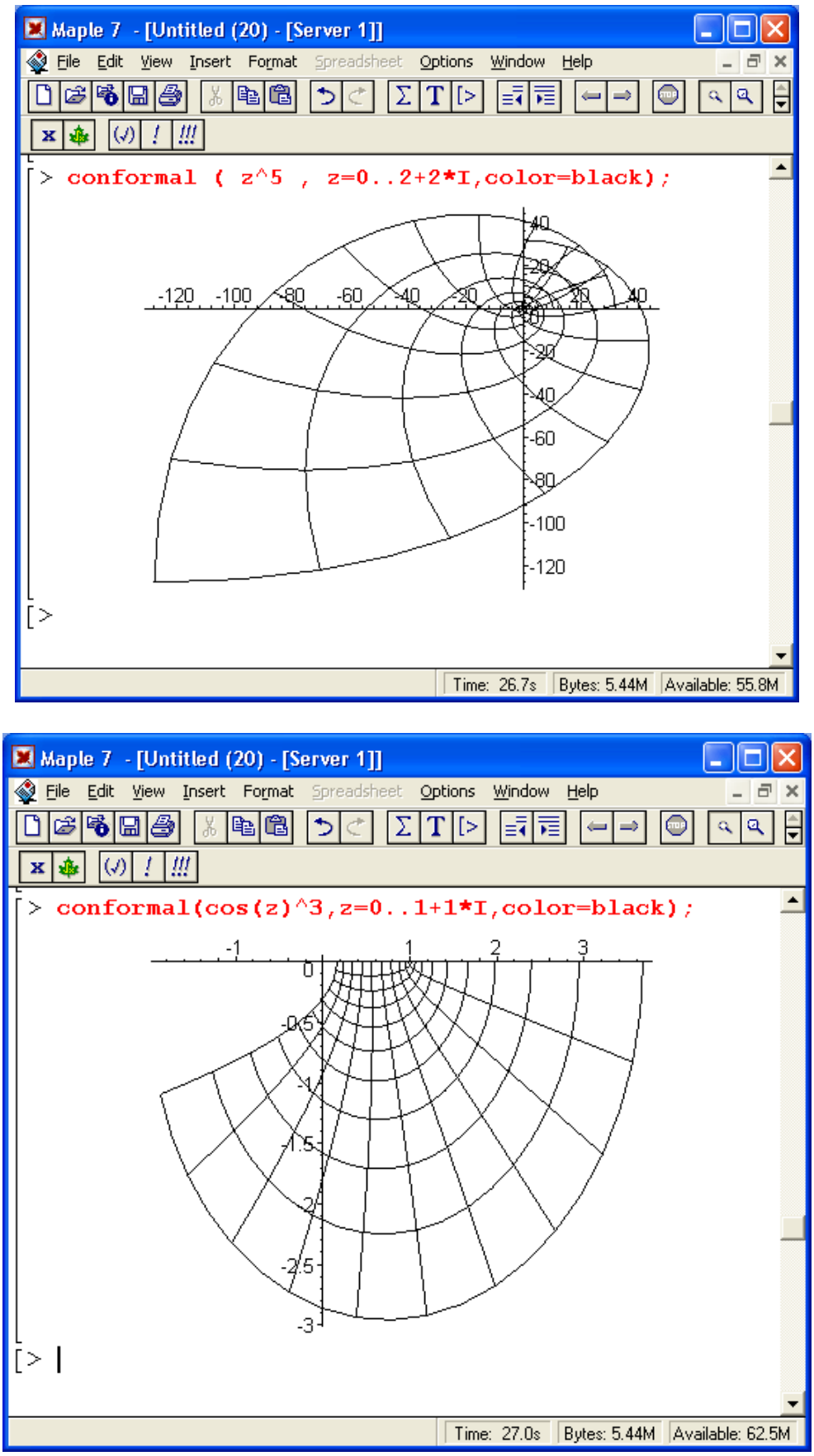


Рис. 12.34. Конформное отображение на комплексной плоскости графиков трех зависимостей

Средства конформного отображения в Maple 7, к сожалению, остаются рудиментарными и вряд ли достаточными для специалистов в этой области математики.

4.5.Графическое представление содержимого матрицы

Многие вычисления имеют результаты, представляемые в форме матриц. Иногда такие результаты можно наглядно представить графически, например в виде гистограммы. Она представляет собой множество столбцов квадратного сечения, расположенных на плоскости, образованной осями строк (row), и столбцов (column) матрицы. При этом высота столбцов определяется содержимым ячеек матрицы.

Такое построение обеспечивает графическая функция matrixplot из пакета plots. На рис. 12.35 показано совместное применение этой функции с двумя функциями пакета linalg, формирующими две довольно экзотические матрицы A и B.

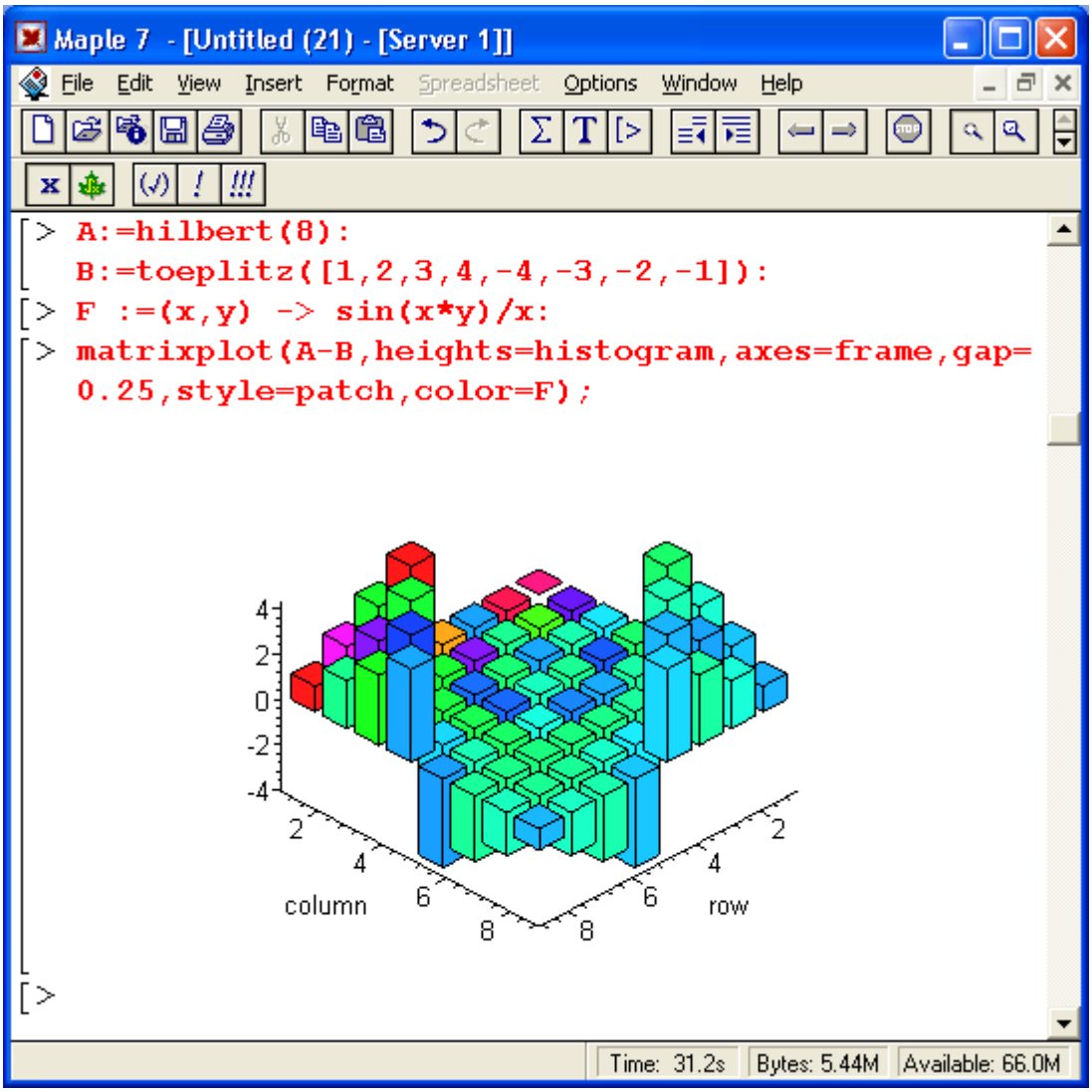


Рис. 12. 35. Графическое представление матрицы

На рис. 12.35 показана графическая визуализация матрицы, полученной как разность матриц A и B. Для усиления эффекта восприятия применяется функциональная закраска разными цветами. Для задания цвета введена процедура F.

4.6.Визуализация ньютоновских итераций в комплексной области

Теперь займемся довольно рискованным экспериментом — наблюдением ньютоновских итераций с их представлением на комплексной плоскости. На рис. 12.36 задана функция /(г) комплексного аргумента. Проследить за поведением этой функции на комплексной плоскости в ходе ньютоновских итераций позволяет графическая функция complexplot3d из пакета plots.

Наблюдаемая картина весьма необычна и свидетельствует о далеко не простом ходе итерационного процесса.

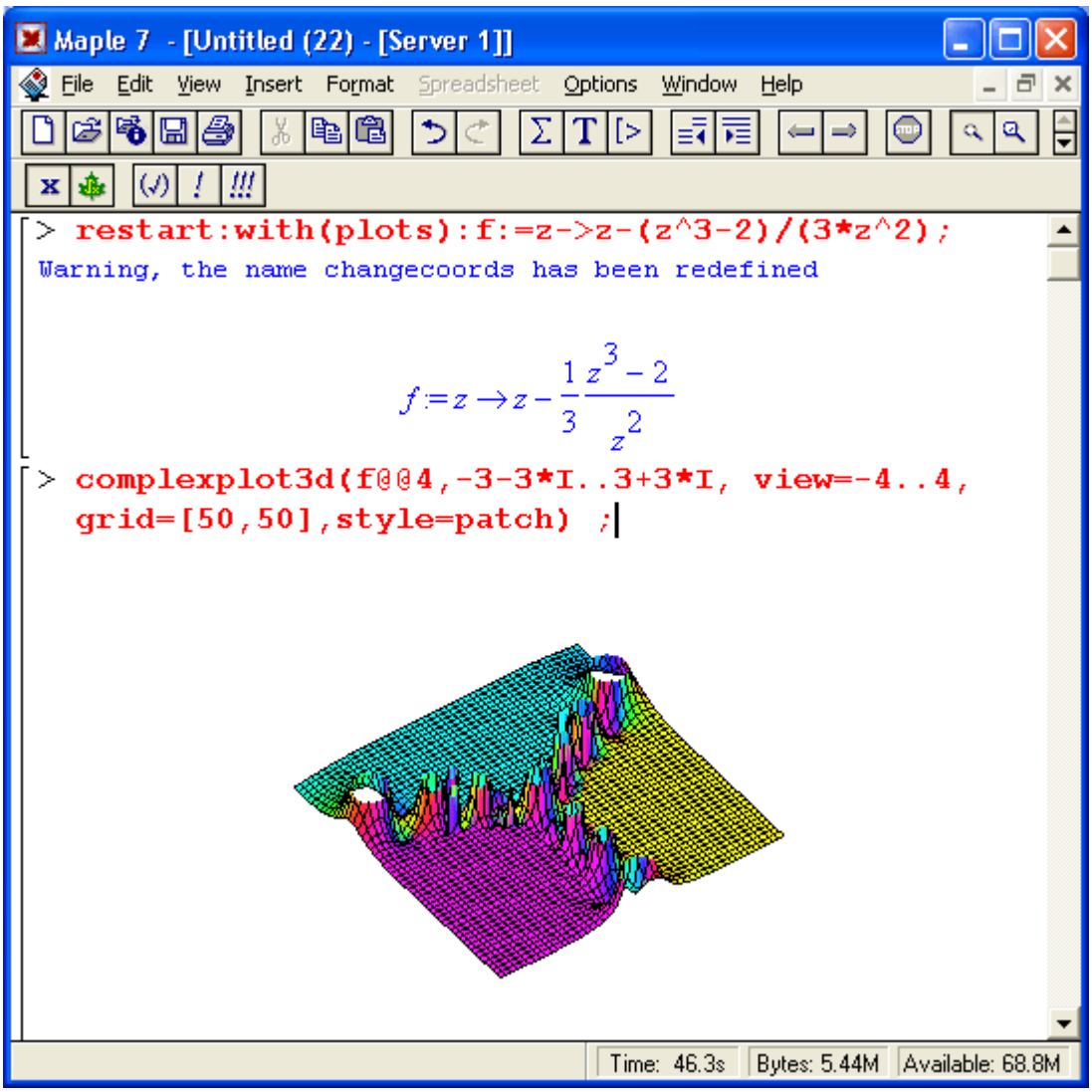


Рис. 12.36. Наблюдение за процессом ньютоновских итераций в трехмерном пространстве

ВНИМАНИЕ

Риск работы с этим примером заключается в том, что в системе Maple 7 он иногда вызывает фатальные ошибки, ведущие к прекращению работы с системой. Обычно при запуске этого примера сразу после загрузки системы Maple такого не происходит, но, когда память загружена, сбой вполне возможен. Объективности ради надо заметить, что в системах Maple 6 и 7 подобное поведение системы не было замечено. Тем не менее рекомендуется записывать подобные примеры на диск перед их запуском.

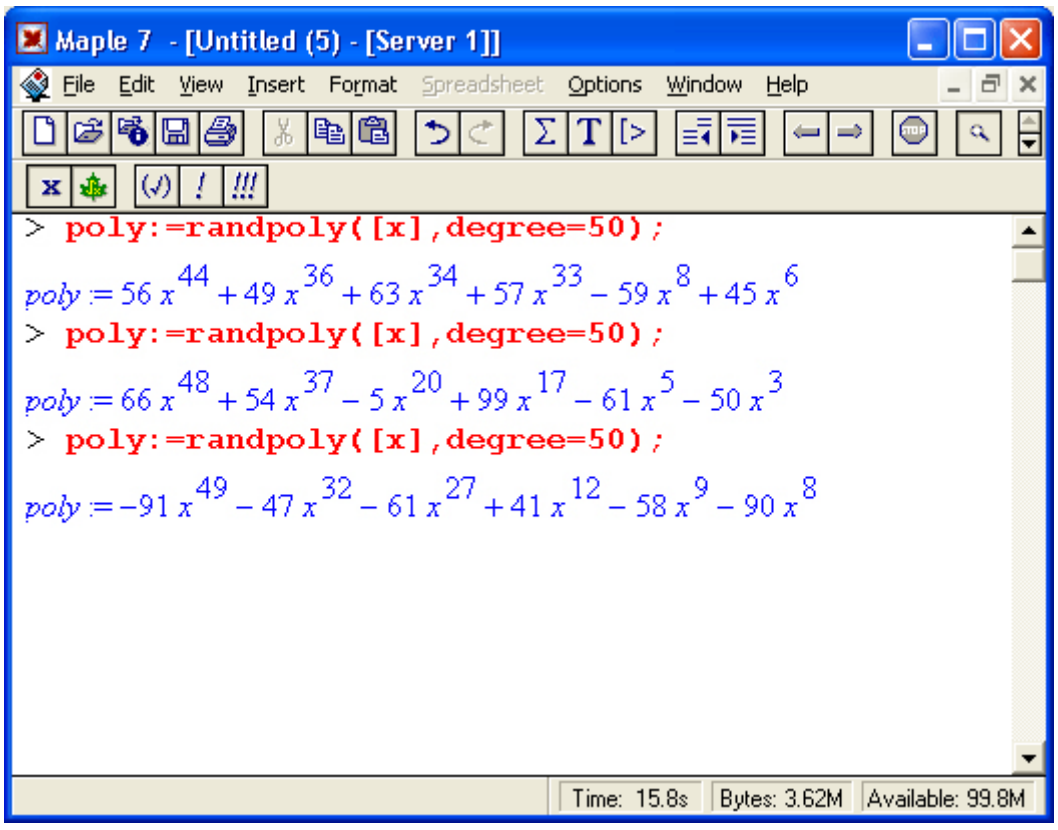
4.7.Визуализация корней случайных полиномов.

Наряду с традиционной для математических и статистических программ возможностью генерации случайных чисел Maple 7 предоставляет довольно экзотическую возможность генерации случайных полиномов с высокой максимальной степенью. Для этого используется функция:

randpoly(var,o)

Она возвращает случайный полином переменной var, причем максимальная степень полинома ptax может указываться параметром o вида degree=nmax.

Приведем примеры генерации случайного полинома с максимальной степенью 50:



С помощью функции allvalues можно построить список SA корней случайного полинома. А с помощью команды вида:

> with(plots):

complexplot(SA.x=-1.2..1.2.style=point):

построить комплексные корни полученного случайного полинома в виде точек • на комплексной плоскости. Один из таких графиков (их можно построить множество) показан на рис. 12.37.

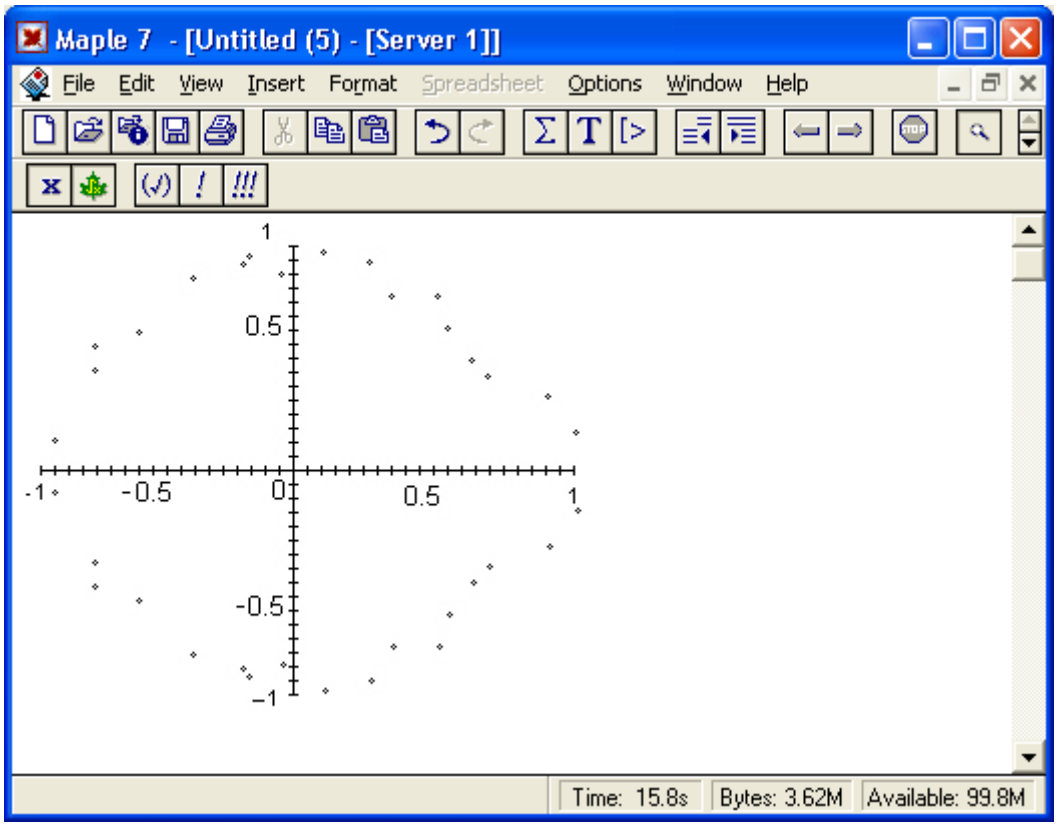


Рис. 12.37. Расположение корней случайного полинома на комплексной плоскости

Можно заметить любопытную закономерность — точки, представляющие корни случайного полинома, укладываются вблизи окружности единичного радиуса с центром в начале координат. Однако этот пример, приводимый в ряде книг по Maple, показывает, что порою вычисления могут давать довольно неожиданные результаты. Кстати говоря, аналитически можно вычислять корни полинома с максимальной степенью не более четырех.

4.8.Визуализация поверхностей со многими экстремумами

Maple 7 дает прекрасные возможности для визуализации поверхностей, имеющих множество пиков и впадин, другими словами, экстремумов. Рисунок 12.38 показывает задание «вулканической» поверхности с глубокой впадиной, окруженной пятью пиками. Здесь полезно обратить внимание на способ задания такой поверхности f(a, b, c) как функции трех переменных a, b и c.

Он обеспечивает индивидуальное задание координат каждого экстремума и его высоты (отрицательной для впадины).

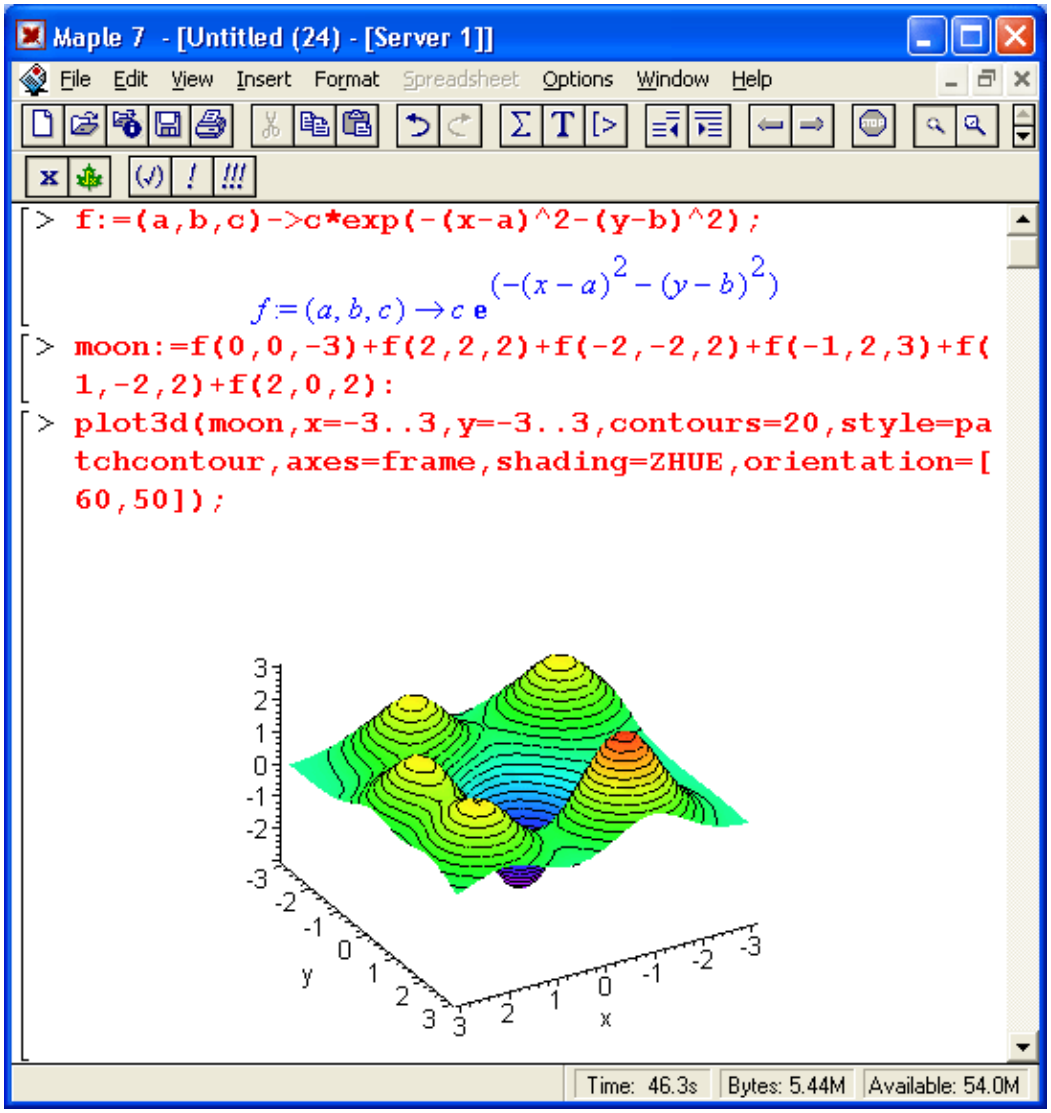


Рис. 12.38. Построение графика поверхности с множеством экстремумов

Наглядность этого графика усилена за счет применения функциональной окраски и контурных линий, нанесенных на саму поверхность. Все эти возможности обеспечивают параметры основной функции plot3d.

А на рис. 12.39 представлен еще один способ задания поверхности — с помощью функции двух угловых переменных f(theta, phi).

При построении этого рисунка также используются функциональная окраска и построение контурных линий.

4.9.Визуализация построения касательной и перпендикуляра

В ряде геометрических построений нужно отроить касательную и перпендикуляр к кривой, отображающей произвольную функцию f(x) в заданной точке x =a. Рисунок 12.40 поясняет, как это можно сделать. Линии касательной T(x) и перпендикуляра N(x) определены аналитически через производную в заданной точке.

Во избежание геометрических искажений положения касательной и перпендикуляра при построении графика функцией plot надо использовать параметр scaling=constrained.

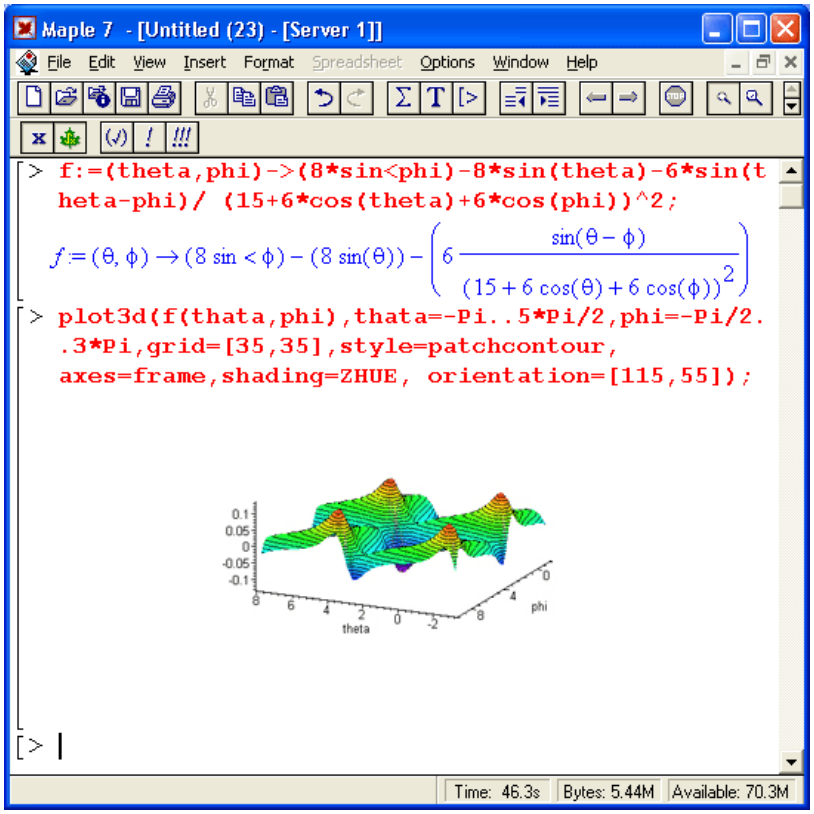


Рис. 12.39. Построение графика поверхности, заданной функцией двух угловых переменных

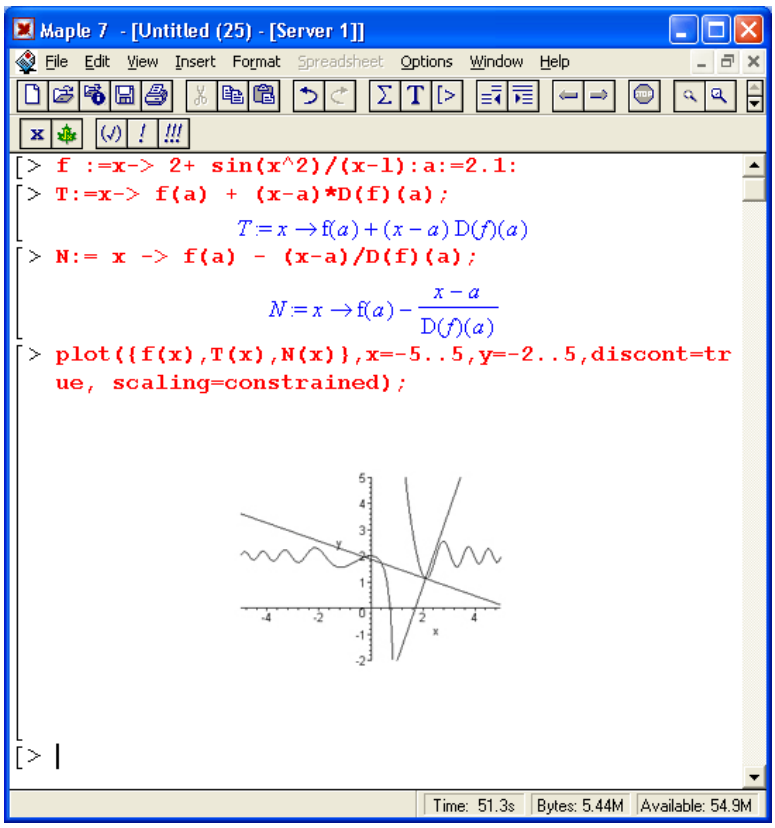


Рис. 12.40. Построение касательной и перпендикуляра к заданной точке графика функции f(x)

4.10.Визуализация вычисления определенных интегралов

Часто возникает необходимость в геометрическом представлении определенных интегралов в виде алгебраической суммы площадей, ограниченных кривой подынтегральной функции f(x), осью абсцисс x и вертикалями x =a их =b (пределами интегрирования). При этом желательно обеспечение закраски верхней и нижней (отрицательной и положительной) площадей разными цветами, например зеленым для верхней площади и красным для нижней. Как известно, численное значение определенного интеграла есть разность этих площадей.

К сожалению, в Maple 7 нет встроенной функции, явно дающей такое построение. Однако ее несложно создать. На рис. 12.41 представлена процедура a_plot, решающая эту задачу. Параметрами процедуры являются интегрируемая функция/(д:) (заданная как функция пользователя), пределы интегрирования a и b и пределы слева am и справа bm, задающие область построения графика f(x).

Рисунок 12.41 дает прекрасное представление о сущности интегрирования для определенного интеграла. Приведенную на этом рисунке процедуру можно использовать для подготовки Эффектных уроков по интегрированию разных функций.

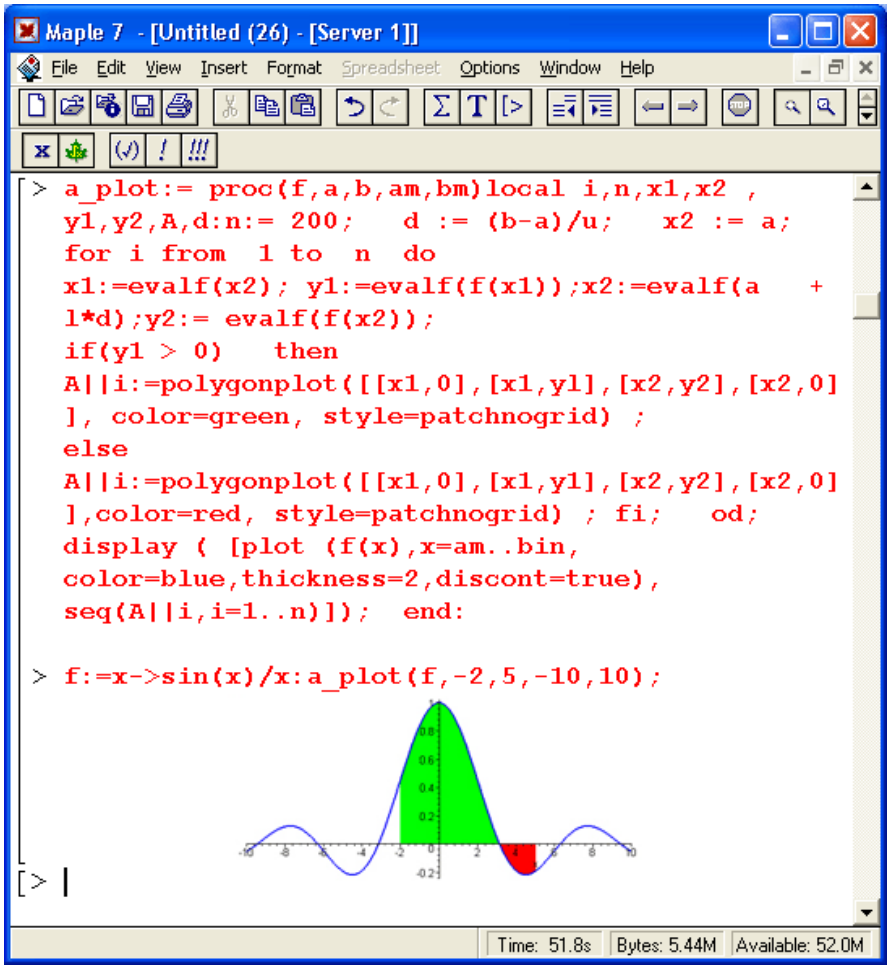


Рис. 12.41. Графическое представление определенного интеграла

4.11.Визуализация теоремы Пифагора

Еще один пример наглядного геометрического представления математических понятий — визуализация известной теоремы Пифагора (рис. 12.42).

В этом примере используется функция построения многоугольников. Наглядность построений усиливается выбором разной цветовой окраски треугольников и квадрата.

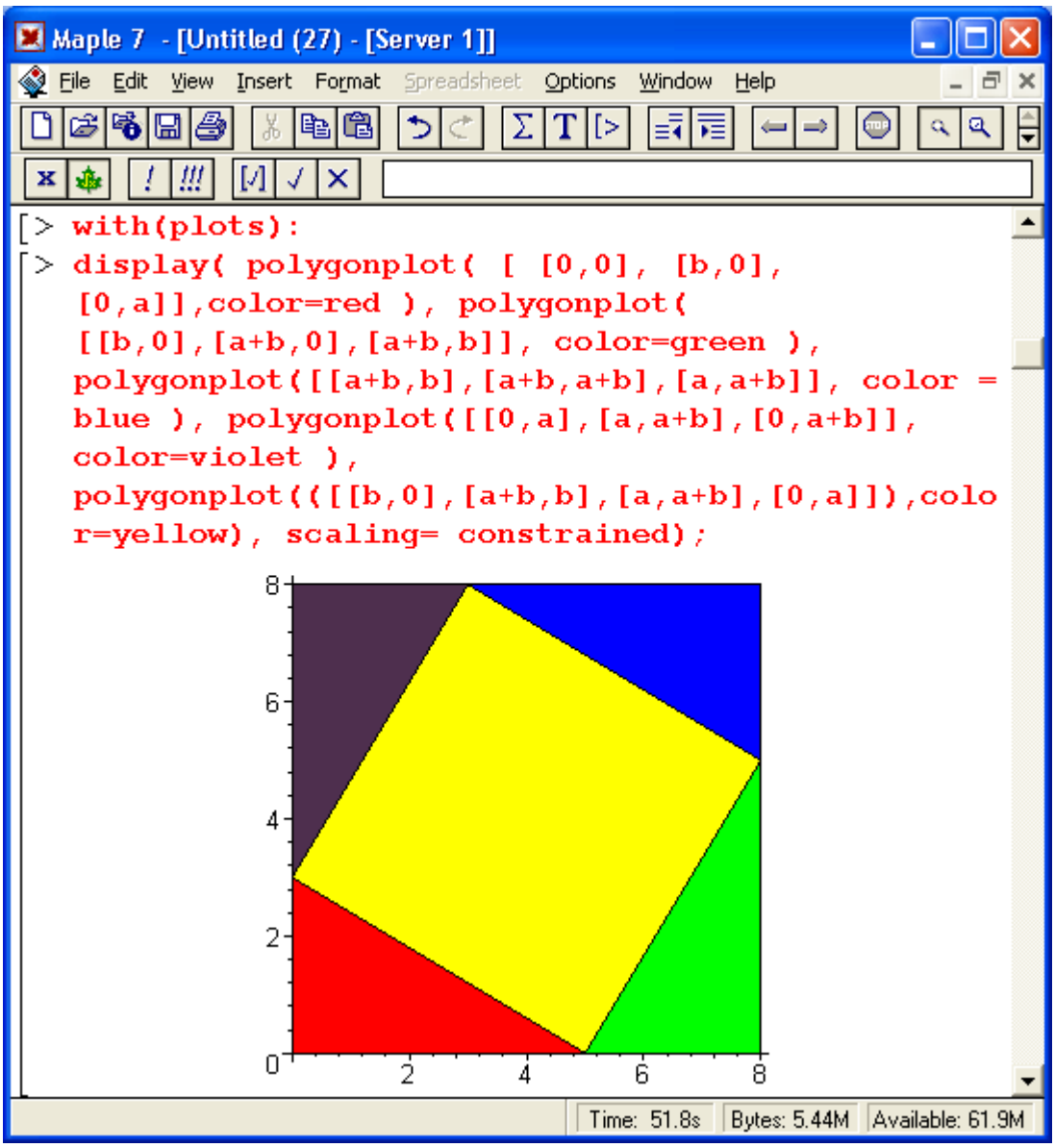
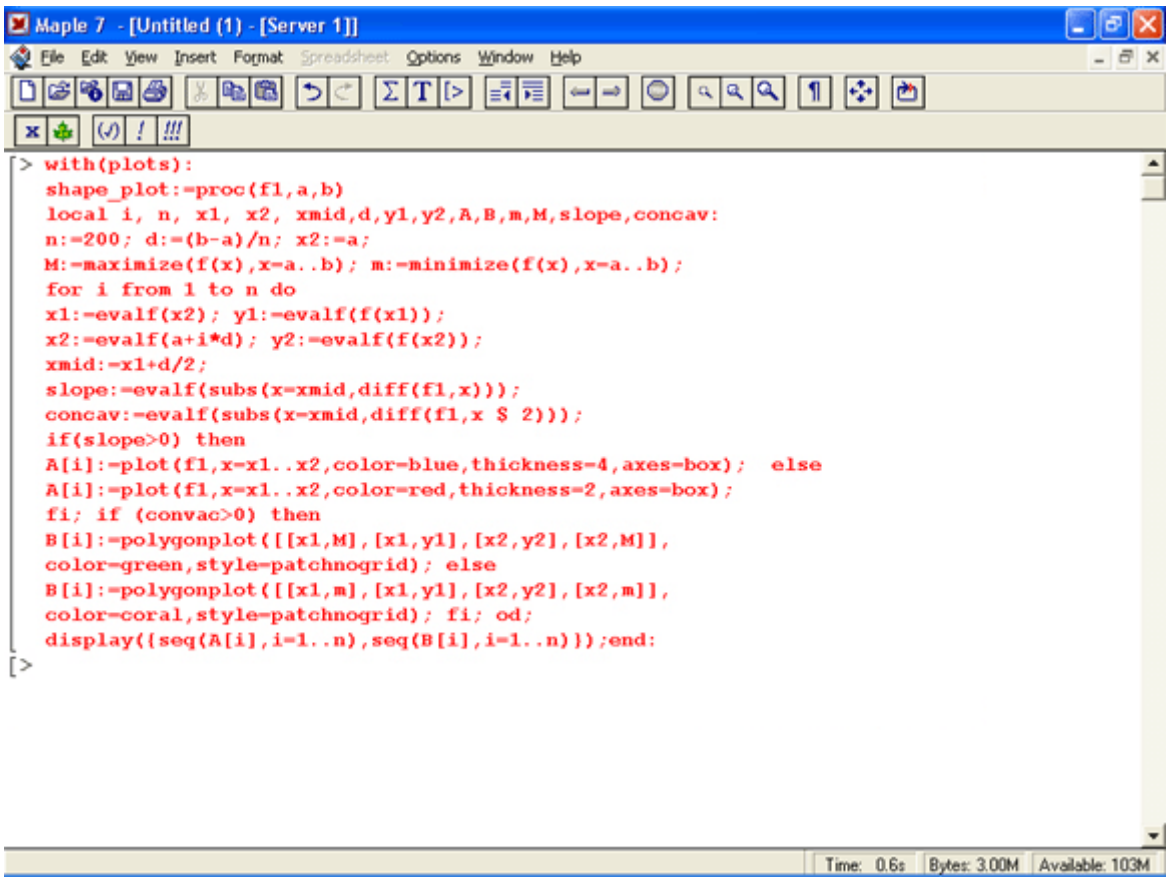


Рис. 12.42. Графическая иллюстрация к теореме Пифагора

4.12.Визуализация дифференциальных параметров кривых

Дифференциальные параметры функции f(x), описывающей некоторую кривую, имеют большое значение для анализа ее особых точек и областей существования. Так, точки с нулевой первой производной задают области, где кривая нарастает (первая производная положительна) или убывает (первая производная отрицательна) с ростом аргументах. Нули второй производной задают точки перегиба кривой.

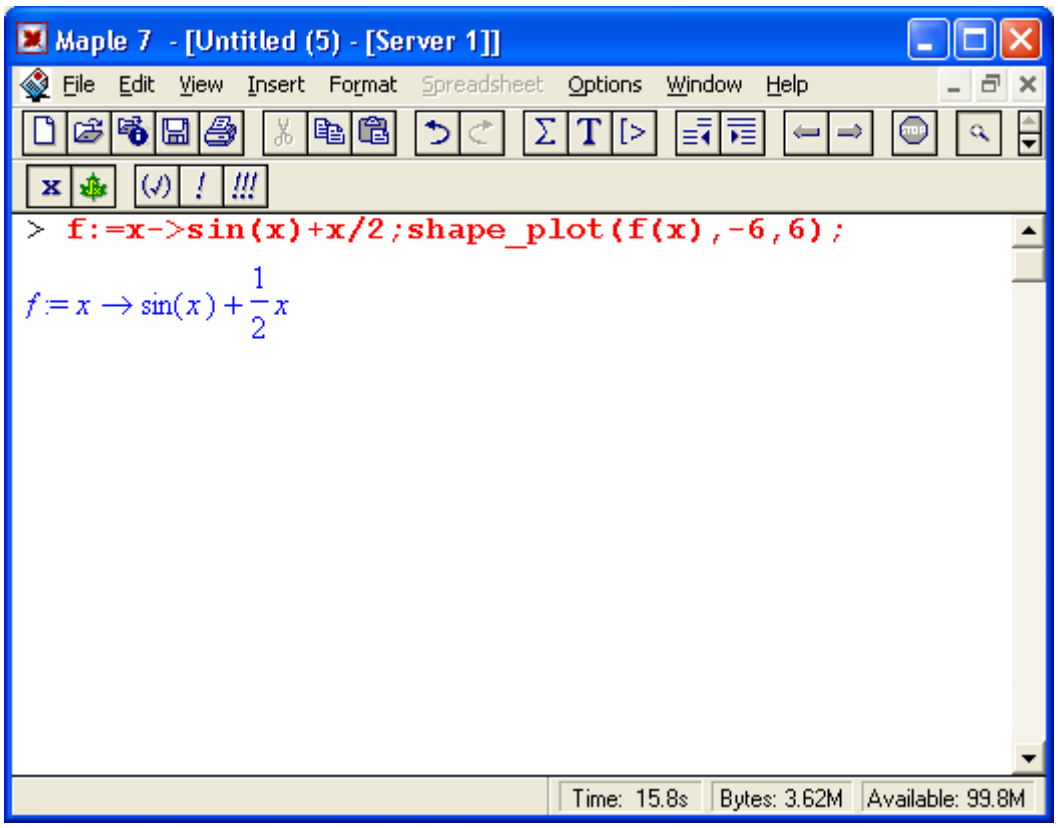
Следующая графическая процедура служит для визуализации поведения кривой /, = /(.г) на отрезке изменениях от а до b:



В этой процедуре заданы следующие цвета (их можно изменить):
Таблица 12.1. Цвета при визуализации в процедуре shape_plot

Изменение /(x)	Цвет
Возрастание	Синий
Убывание	Красный
Площадь	Цвет
Над минимумом	Зеленый
Под максимумом	Коралловый

Например, для функции:



построенный график будет иметь вид, представленный на рис. 12.43 (естественно, в книге цвета — лишь оттенки серого).

Рисунок 12.43 дает наглядное представление о поведении заданной функции. Рекомендуется опробовать данную процедуру на других функциях. Следует отметить, что, поскольку процедура использует функции ntiroimize и maximize, она может давать сбои при исследовании сложных функций, содержащих специальные математические функции или особенности. Иногда можно избежать такой ситуации, исключив особенность. Например, для анализа функции sin(x)/x можно записать ее в виде:

>f:=x->if x=0 then 1 else sin(x)/x

end if;

shape_plot(f(x),-10,10);

Исполнение приведенной выше строки ввода дает график, представленный на рис. 12.44.

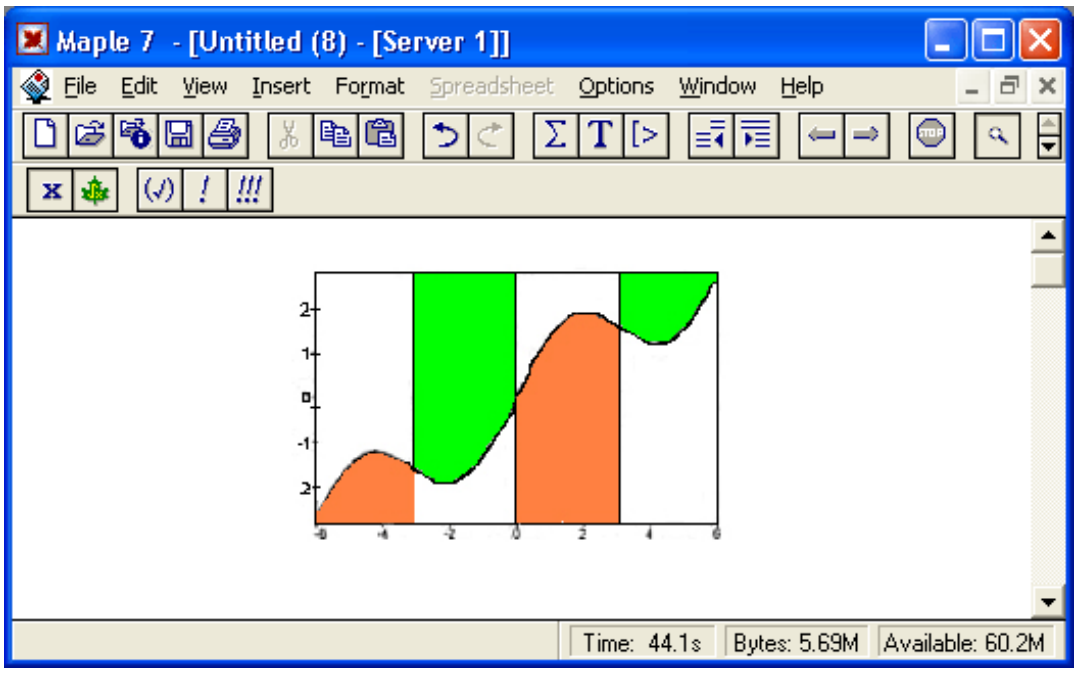


Рис. 12.43. Визуализация поведения функции f(x)

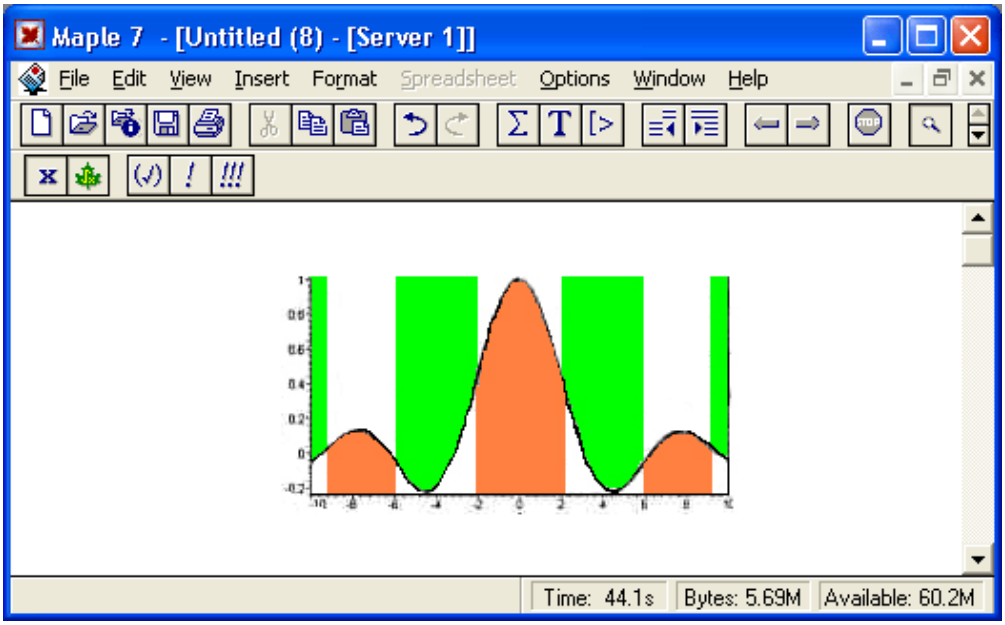


Рис. 12.44. Визуализация поведения функции sin(x)/x

Данная процедура дает хорошие результаты при анализе функций, представленных полиномами. Вы можете сами убедиться в этом.

4.13.Иллюстрация итерационного решения уравнения f (x) = x

Классическим методом решения нелинейных уравнений является сведение их к виду x =f(x) и применение метода простых итераций $x_k = s(x_{k-1})$ при заданном значенииx_0. Приведем пример такого решения:

>f :=x ->3*1n(x+l);

f:=x-> 3ln(x+1)

>x||0 := 0.5:

x0:=5

>x0 := .5;

x0:=.5

>for k from 1 to 16 do x||k := evalf(f(x||(k-l))): od;

xl := 1.216395324

x2 := 2.387646445

x3 := 3.660406248

x4:= 4.617307866

x5:= 5.177557566

x6:= 5.462768931

x7:= 5.598173559

x8:= 5.660378631

x9 := 5.688529002

xl0:= 5.701181910

x11 := 5.706851745

x12 := 5.709388956

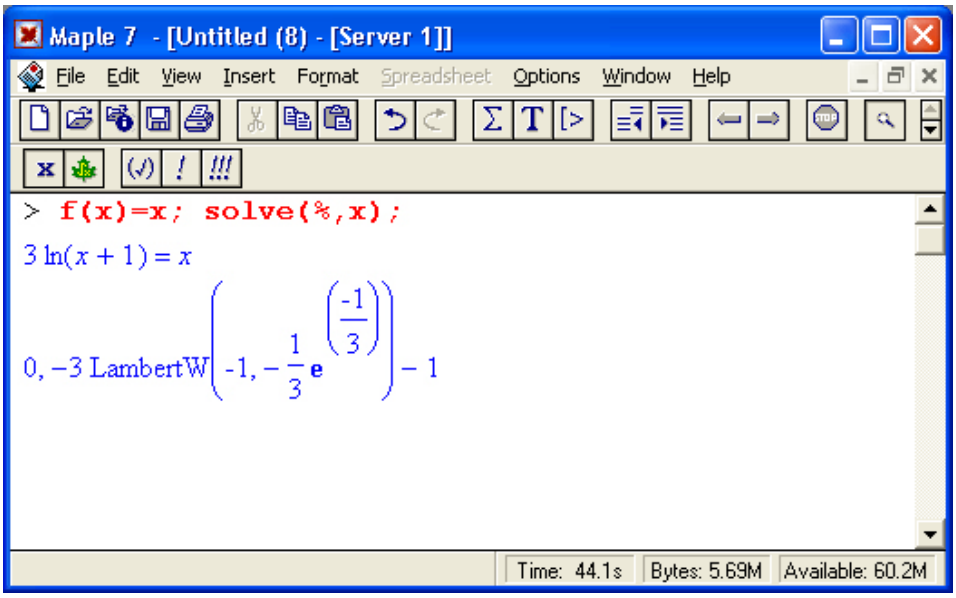
x13:= 5.710523646

x14 — 5.711030964

xl5:= 5.711257755

x16:= 5.711359134

Нетрудно заметить, что значения x_k в ходе итераций явно сходятся к некоторому значению. Проведем проверку решения, используя встроенную функцию solve:

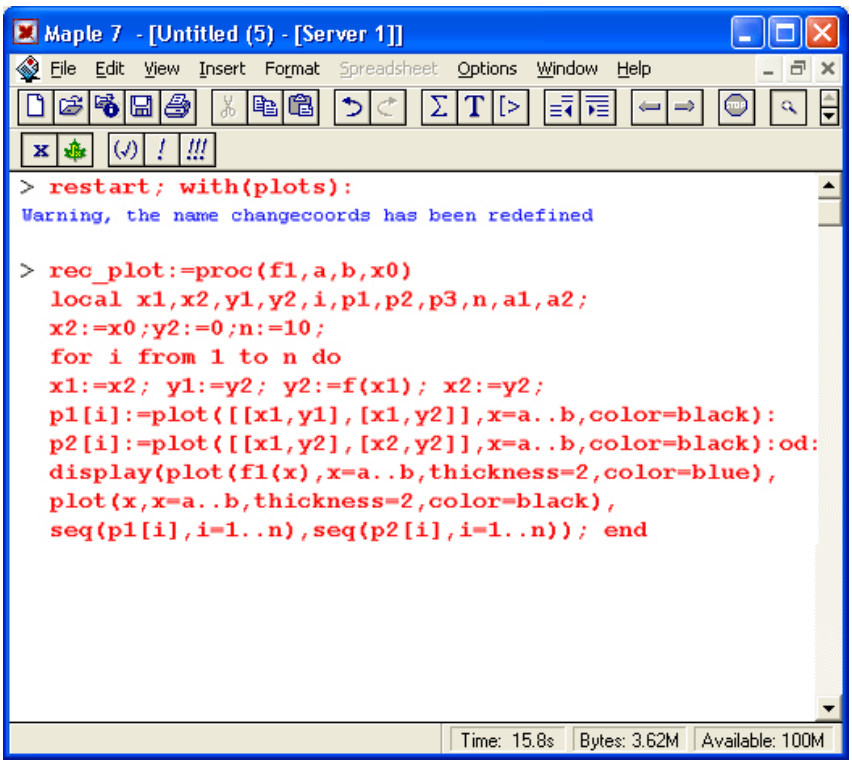


Результат выглядит необычно — помимо довольно "очевидного корнях x= 0 значение другого корня получено в виде специальной функции Ламберта. Впрочем, нетрудно найти и его численное значение:

```
> evalf(%);
```

0., 5.711441084

Однако как сделать процесс решения достаточно наглядным? Обычно для этого строят графики двух зависимостей — прямой x и кривой f(x) — и наносят на них ступенчатую линию перемещения точки x_k. Специальной функции для графиков подобного рода Maple 7 не имеет. Однако можно составить специальную процедуру для их построения .Ее листинг, заимствованный из примера, описанного в пакете обучения системе Maple 7 - PowerTools, представлен ниже:



Параметрами этой процедуры являются: f1 — функция f(x); a и b — пределы изменениях при построении графика; x0 — значение x, с которого начинаются итерации. Исполнив команду:

>rec_p1ot(f(x), 0, 8, x0):

можно наблюдать график, иллюстрирующий итерационный процесс. Он представлен на рис. 12.45.

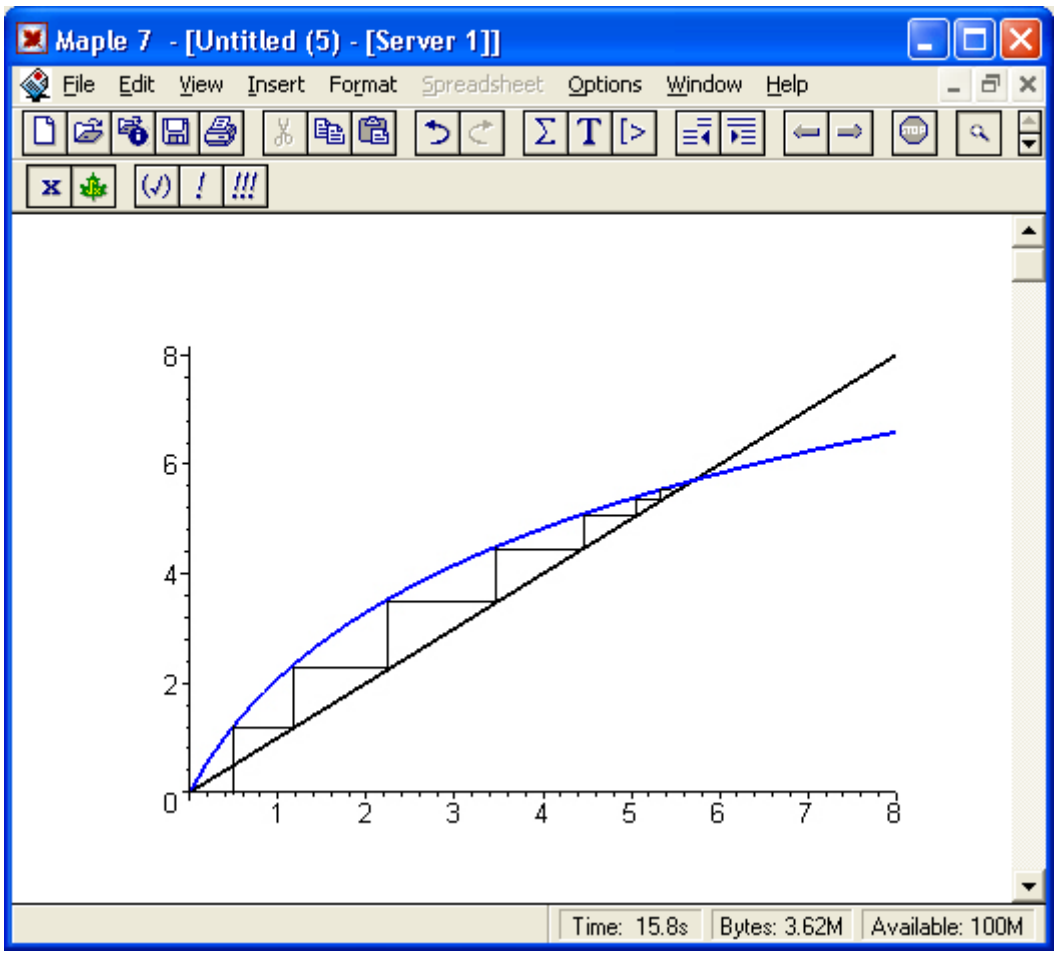


Рис. 12.45. Иллюстрация процесса итераций

Нетрудно заметить, что для данной функции процесс итераций хотя и не очень быстро, но уверенно сходится к точке пересечения прямой y = x и кривой y=f(x). Вы можете, меняя зависимость f(x), провести исследования сходимости уравнений x = f(x).

4.14.Построение сложных фигур в полярной системе координат

Некоторые виды математической графики имеют определенную художественную ценность и фигурируют в символике различных стран и общественных организаций. Остановимся на нескольких таких примерах применительно к графике в полярной системе координат. Представим фигуры, образованные множеством линий на плоскости.

Рисунок 12.46 демонстрирует одну из таких фигур. Это семейство из 10 кардиоид разного размера. Параметр scallIng=constrained обеспечивает правильное отображение фигур — каждая кардиоида вписывается в огибающую ее невидимую окружность. Размер кардиоид задается значением параметра a.

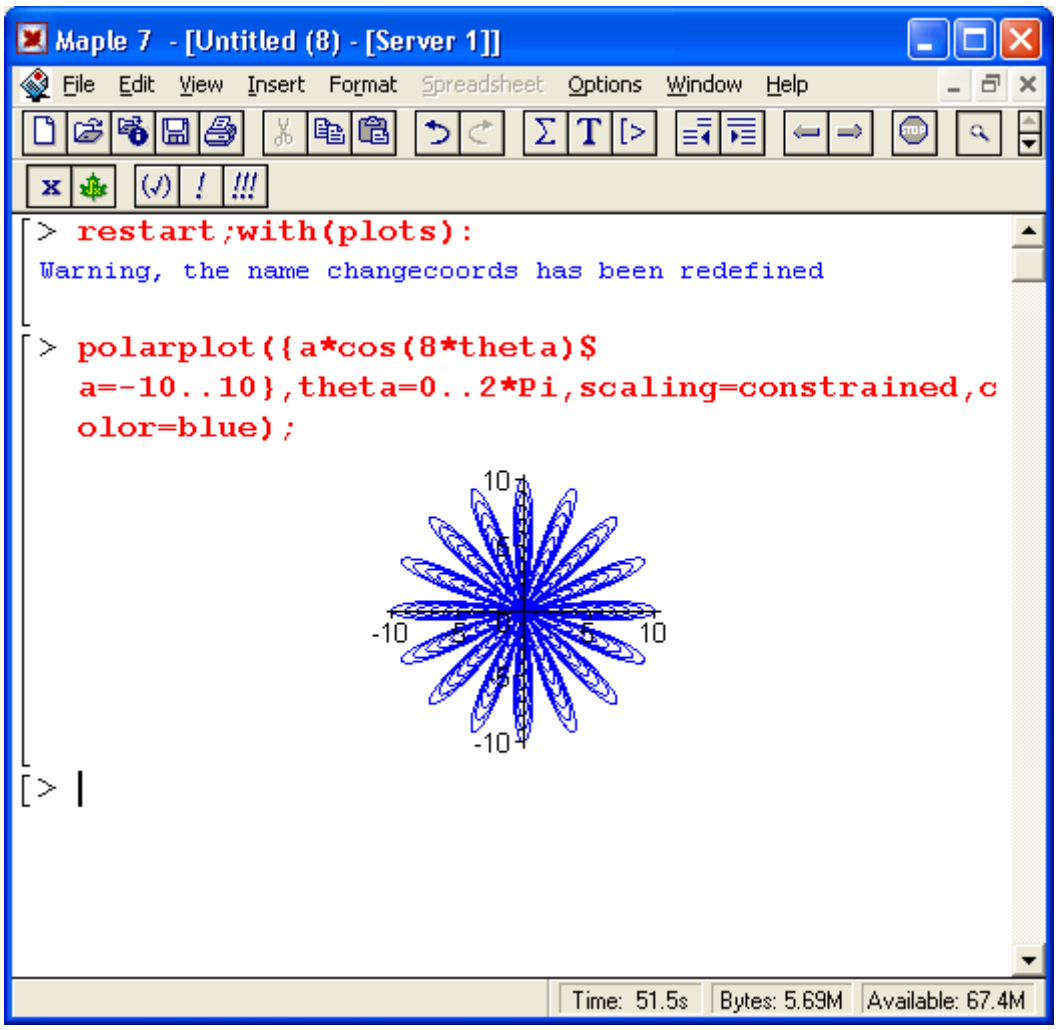


Рис.12.46. Семейство кардиоид на одном графике

Еще одно семейство кардиоид, на сей раз шестилепестковых, представлено на рис. 12.47. Здесь также изменяемым параметром каждой фигуры является ее размер, заданный параметром a.

Фигуре, представленной на рис. 12.48, трудно дать определенное название. Назовем ее волнообразной спиралью.

По образу и подобию приведенных фигур читатель может опробовать свои силы в создании новых красочных фигур в полярной системе координат. Некоторые из них поразительно напоминают снежинки, картинки в калейдоскопе и изображения морских звезд. Если убрать параметр color=black, .введенный ради черно-белой печати картинок в книге, то можно усилить красочность фигур за счет их разноцветной окраски.

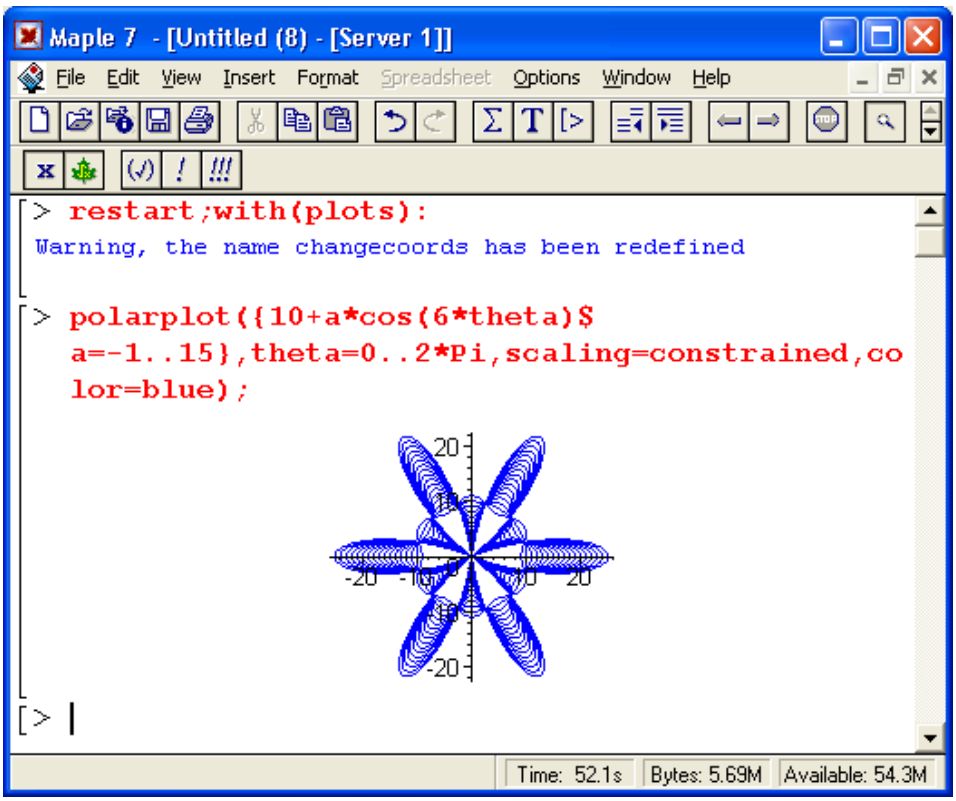


Рис. 12.47. Семейство шестилепестковых кардиоид

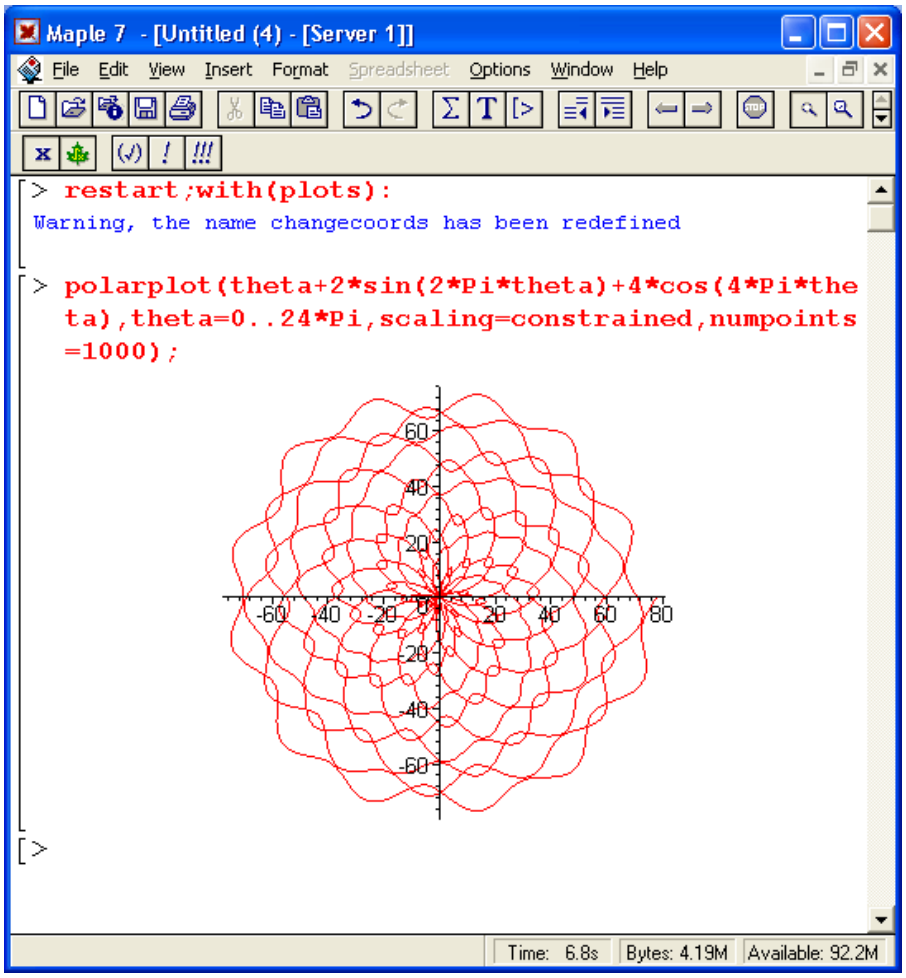
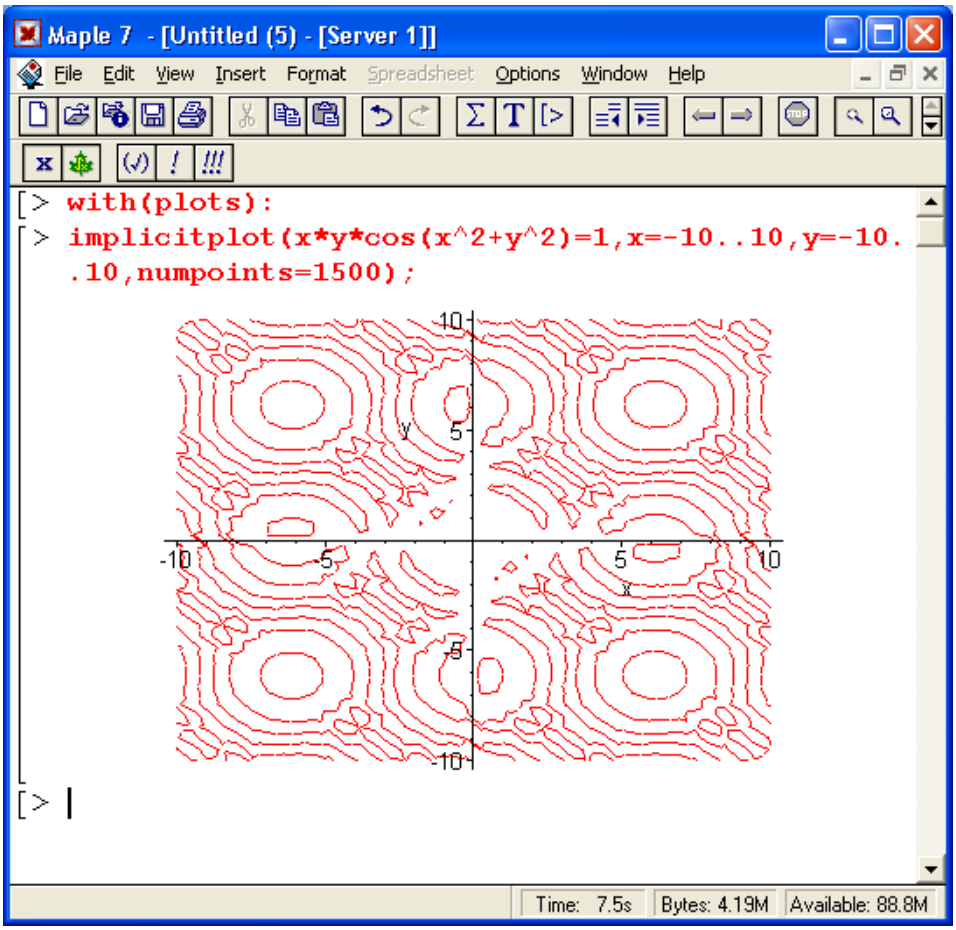


Рис. 12.48. Фигура— волнообразная спираль

4.15.Построение сложных фигур импликативной графики

Импликативные функции (см. урок 7) нередко имеют графики весьма любопытного вида. Ограничимся парой примеров построения таких графиков, представленных на рис. 12.49. Эти фигуры напоминают контурные графики функции двух переменных.

Приведенные примеры дают весьма наглядное представление о больших возможностях визуализации решений самых различных задач в системе Maple V. Можно значительно расширить их, эффектно используя описанные ранее приемы анимации изображений. В целом надо отметить, что графические возможности Maple 7 дают новый уровень качества графики современных математических систем, о котором с десяток лет тому назад можно было только мечтать.



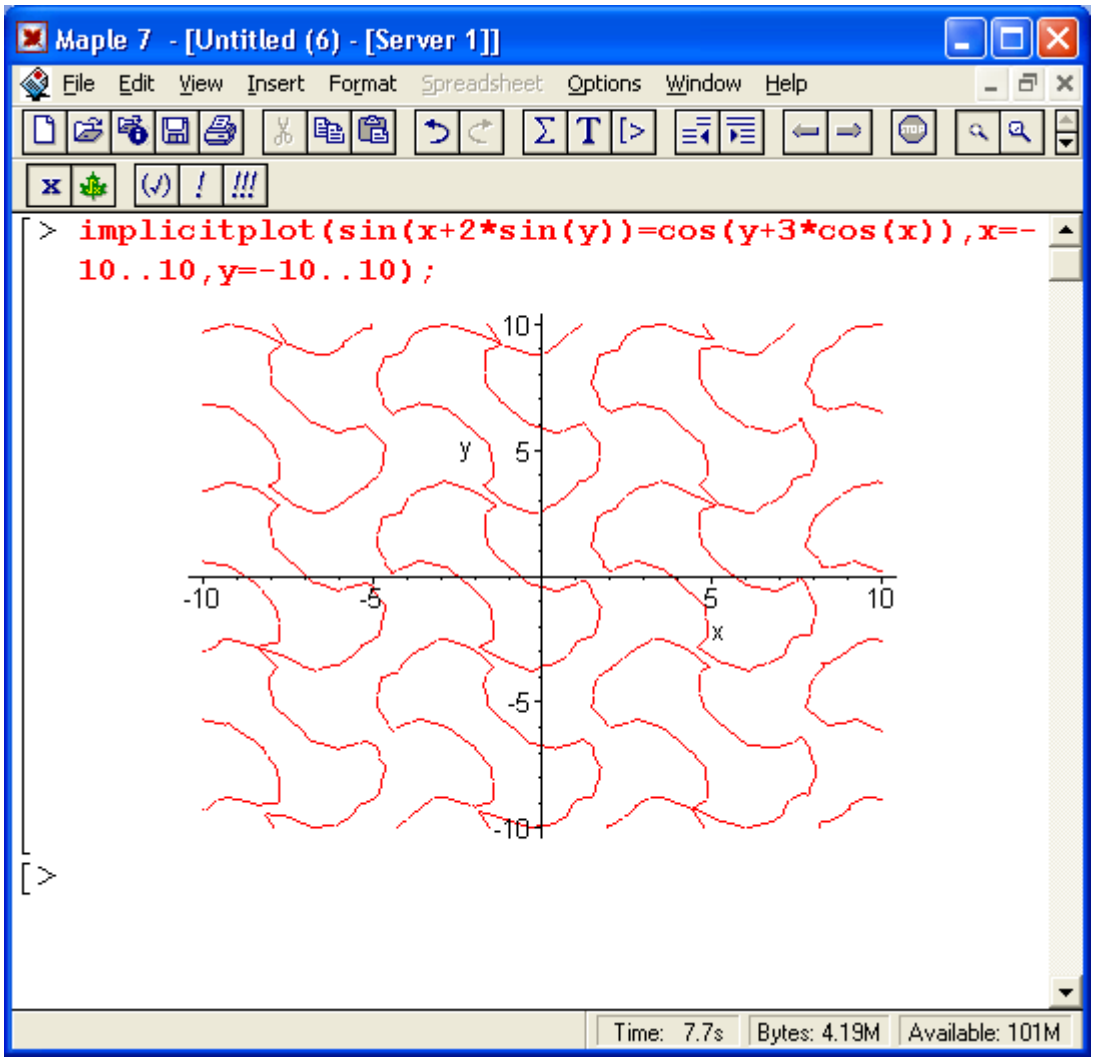


Рис. 12.49. Построение сложных фигур, заданных импликативными функциями

5.Расширенная техника анимации

5.1.Анимирование разложения импульса в ряд Фурье

Анимирование изображений является одним из самых мощных средств визуализации результатов моделирования тех или иных зависимостей или явлений.

Порою изменение во времени одного из параметров зависимости дает наглядное представление о его математической или физической сути.

Здесь мы расширим представление об анимации и рассмотрим не вполне обычный пример — наблюдение в динамике за

гармоническим синтезом некоторой произвольной функции f(x) на отрезке изменения л; от 0 до 1. Значения функции f(x) могут быть одного знака или разных знаков. В этом примере можно наблюдать в динамике синтез заданной функции рядом Фурье с ограниченным числом синусных членов (гармоник) — до 1, 2, 3..JV. На рис. 12.50 представлен документ, реализующий такое разложение и затем синтез для пилообразного линейно нарастающего импульса, описываемого выражением f(x) = -1 + 2 *x. На графике строится исходная функция и результат ее синтеза в динамике анимации.

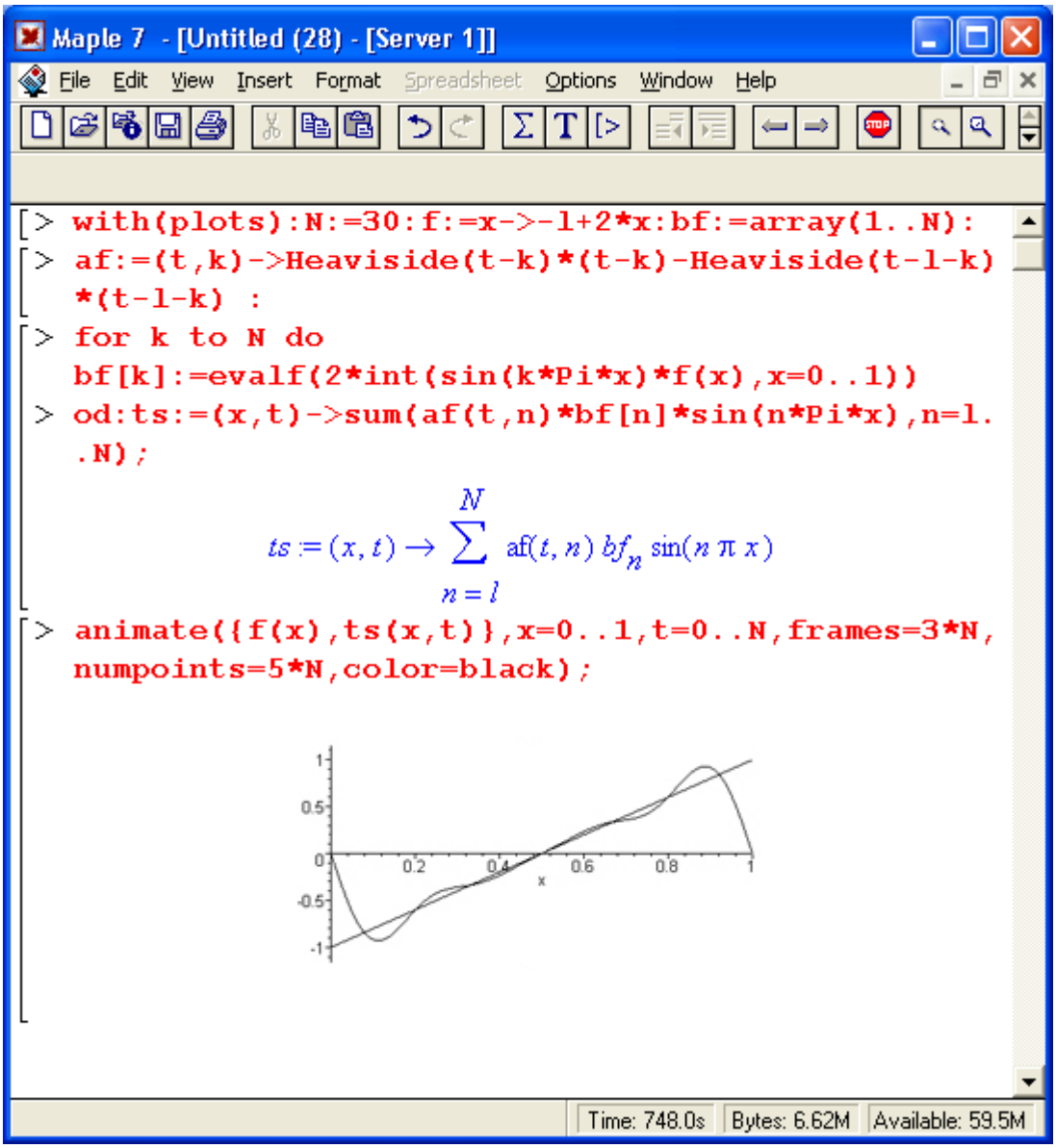


Рис. 12.50. Один из первых стоп-кадров анимации разложения импульса в ряд Фурье

Рисунок 12.51 показывает завершающий стоп-кадр анимации, когда число гармоник N равно 30. Нетрудно заметить, что такое число гармоник в целом неплохо описывает большую часть

импульса, хотя в. его начале и в конце все еще заметны сильные отклонения.

Для f(x) = 1 строится приближение для однополярного импульса с длительностью 1 и амплитудой 1, при f(x) =x — приближение для пилообразного линейно нарастающего импульса, при f(x) =x^2 — приближение для нарастающего по параболе импульса, при f(x)=signum(x-l/2) — приближение для симметричного прямоугольного импульса-меандра и т. д. Фактически можно наблюдать анимационную картину изменения формы импульса по мере увеличения числа используемых для синтеза гармоник. Выбор используемого числа гармоник осуществляет амплитудный селектор — функция a= f(t,k), основанная на применении функции Хевисайда.

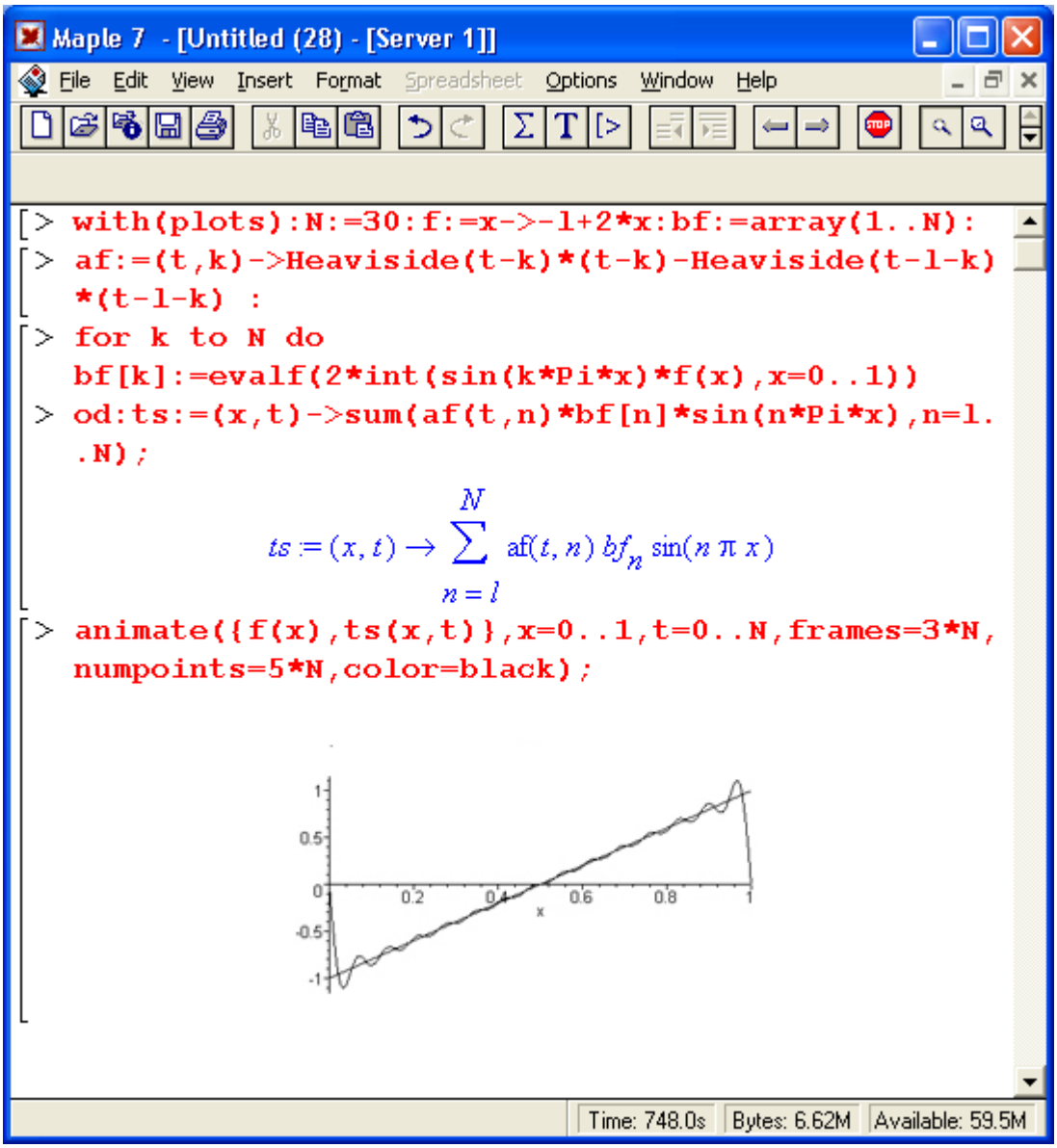


Рис. 12.51. Второй (завершающий) кадр анимации

Самым интересным в этом примере оказывается наблюдение за зарождением и эволюцией эффекта Гиббса — так называют волнообразные колебания на вершине импульса, связанные с ограничением числа гармоник при синтезе сигнала. С ростом числа гармоник эффект Гиббса не исчезает, просто обусловленные им выбросы вблизи разрывов импульса становятся более кратковременными. Амплитуда импульсов может достигать 18% от амплитуды перепадов сигнала, что сильно ухудшает приближение импульсных сигналов рядами Фурье и вынуждает математиков разрабатывать особые меры по уменьшению эффекта Гиббса.

Можно ли наблюдать одновременно все фазы анимации? Можно! Для этого достаточно оформить анимационную картину, созданную функцией animate, в виде отдельного графического, объекта например g, после чего можно вывести все его фазы оператором display. Это и иллюстрирует рис. 12.52. На этот раз задано f(x) = signum(x-l/2) и N = 25. Таким образом рассматриваются симметричные прямоугольные импульсы - меандр. У каждого рисунка координатные оси с делениями удалены параметром axes=none.

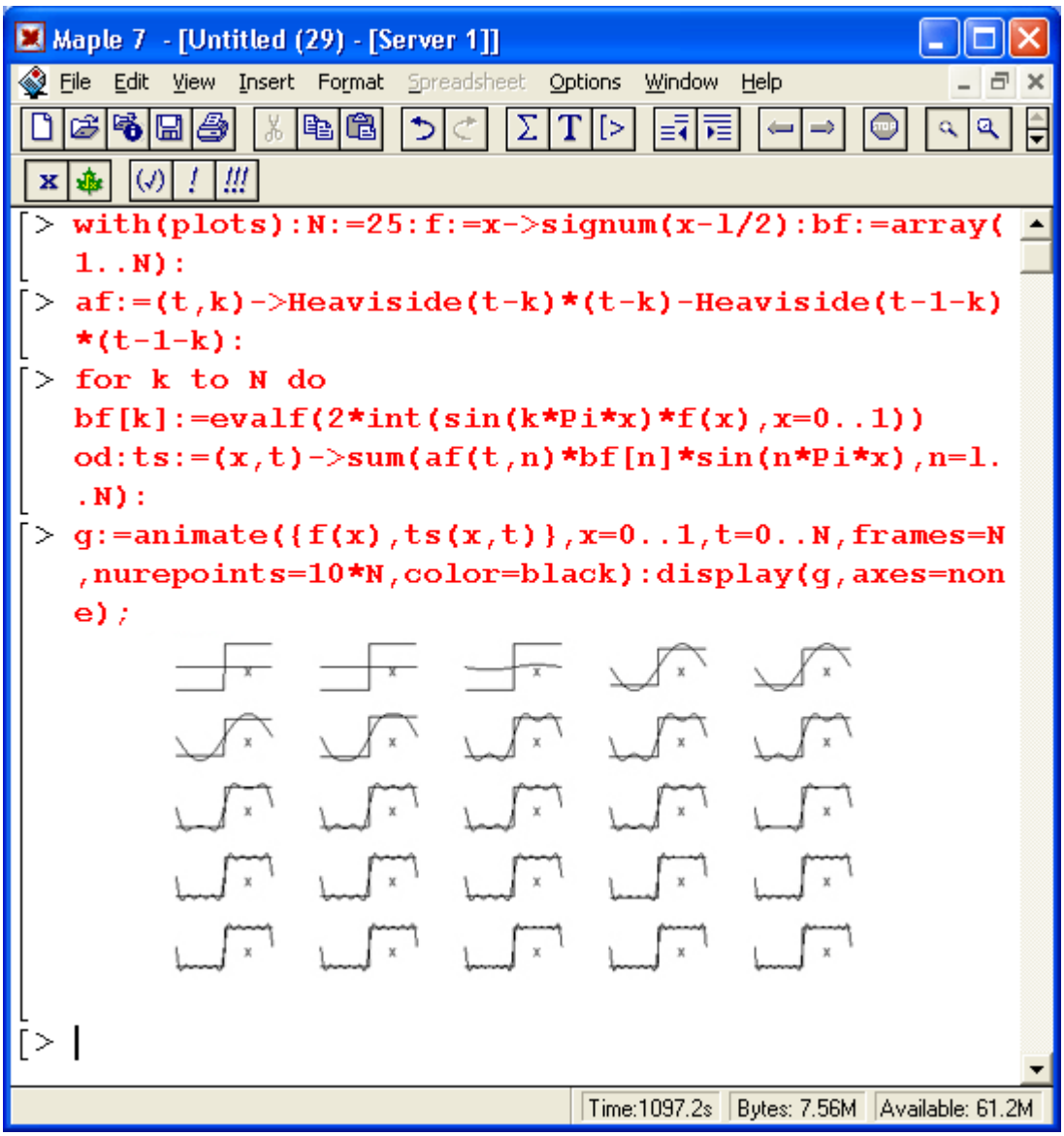


Рис. 12.52. Иллюстрация получения всех кадров анимации двумерного графика

Любопытно отметить, что при определенных числах гармоник связанная с колебательными процессами неравномерность вершины импульса резко уменьшается. Наблюдение этого явления и является наиболее интересным и поучительным при просмотре данного примера.

При внимательном просмотре рис. 12.52 заметно, что после некоторого периода установления фазы анимационной картинки практически повторяются. Это связано с известным обстоятельством — установившийся спектр меандра содержит только нечетные гармоники. Поэтому, к примеру, вид спектрального разложения при 22 гармониках будет тот же, что и при 21 гармонике, при 24 гармониках тот же, что при 23, и т. д.

Однако эта закономерность проявляется только при установившемся (стационарном) спектре.

5.2.Наблюдение надрав анимации поверхности

Наблюдение за развитием поверхности производит на многих (особенно на студентов) большое впечатление. Оно позволяет понять детали создания сложных трехмерных графиков и наглядно представить их математическую сущность. Рассмотрим анимацию поверхности на примере рис. 12.18.

Как и для случая анимации двумерного графика, большой интерес представляет построение всех фаз анимации на одном рисунке. Делается это точно так же, как в двумерном случае. Это иллюстрирует рис. 12.53. На нем представлены 8 фаз анимации трехмерной поверхности cos(t*x*y/3), представленной функцией трех переменных t, x и y. При этом изменение первой переменной создает фазы анимации поверхности.

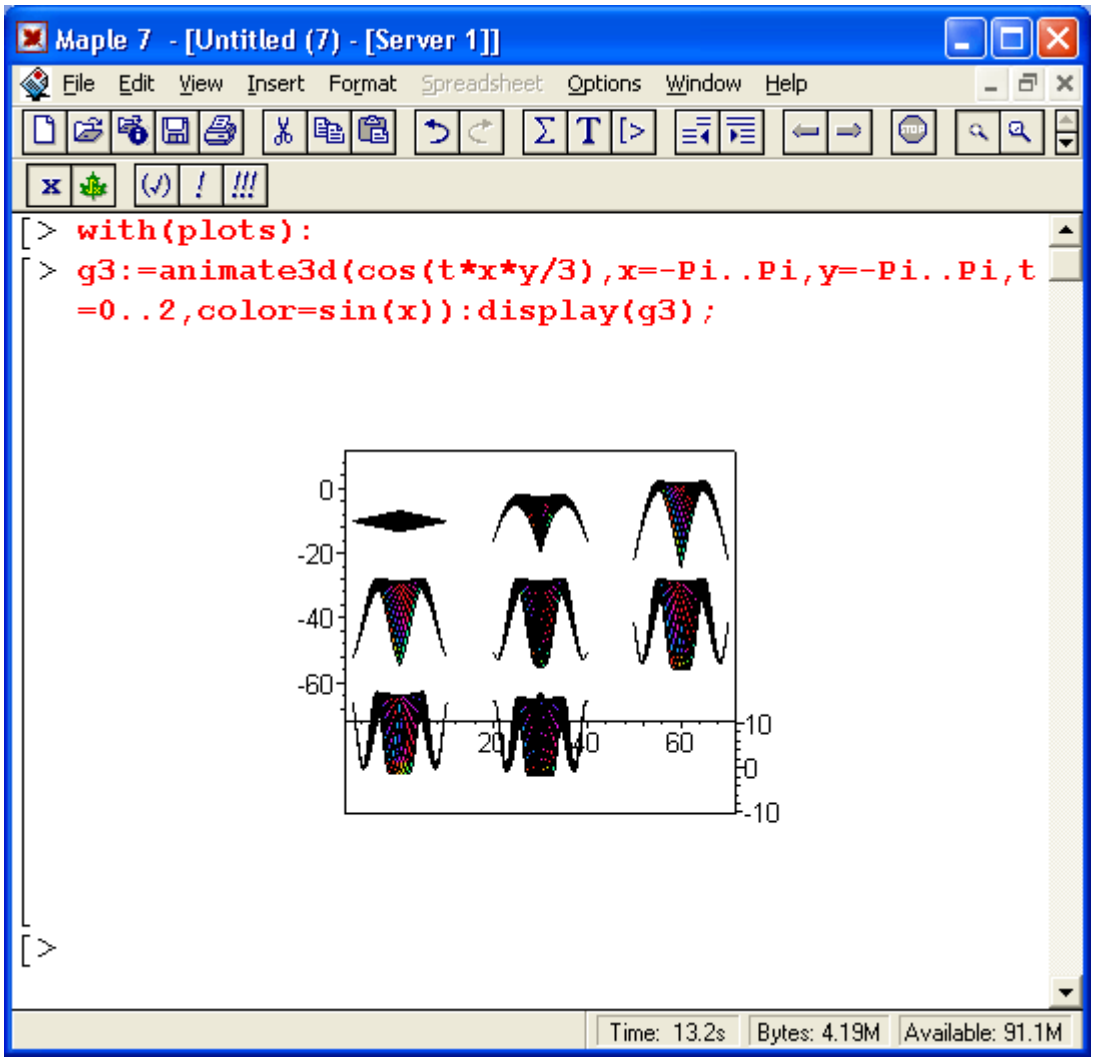


Рис. 12.53. Фазы анимации трехмерной поверхности

Применение анимации дает повышенную степень визуализации решений ряда задач, связанных с построением двумерных и трехмерных графиков. Следует отметить, что построение анимированных графиков требует дополнительных и достаточно существенных затрат оперативной памяти. Поэтому злоупотреблять числом стоп-кадров таких графиков не стоит.

5.3.Новая функция для построения стрелок arrow

В пакет plots системы Maple 7 введена новая функция построения стрелок arrow. Она задается в виде arrow(u,[v,]opts) или arrow(U,opts)

Построение стрелок задается одномерными массивами координат начала стрелок и их направления и и v или двумерным массивом U, которые могут быть представлены векторами, списками или множествами. Вид стрелок задается параметром opts, который может иметь значения shape, length, width, head_width, head_length или plane и задает вид стрелок (форму, длину, ширину и т. д.). Детали задания параметров можно найти в справке по данной функции. Рисунок 12.54 дает наглядное представление о ее возможностях.

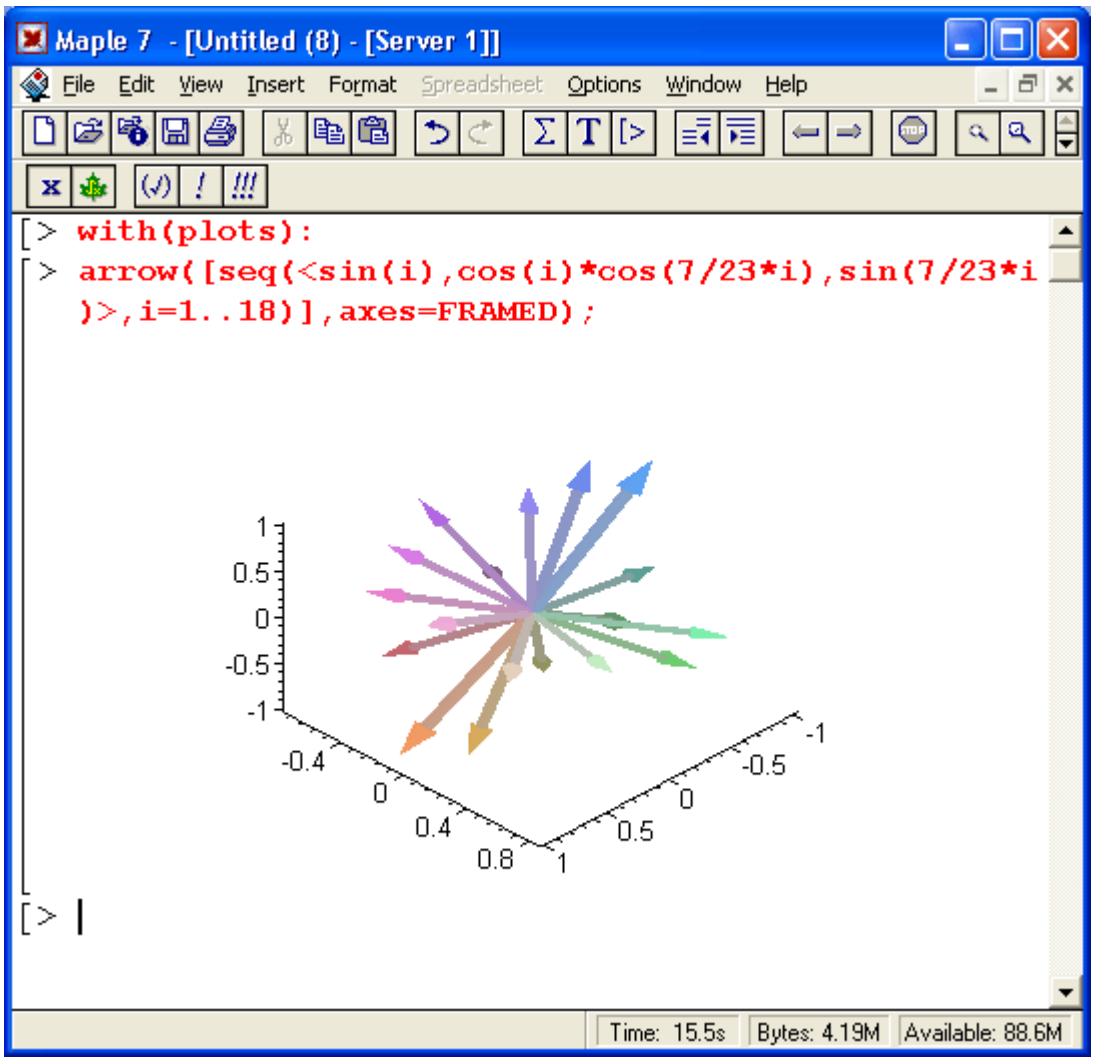
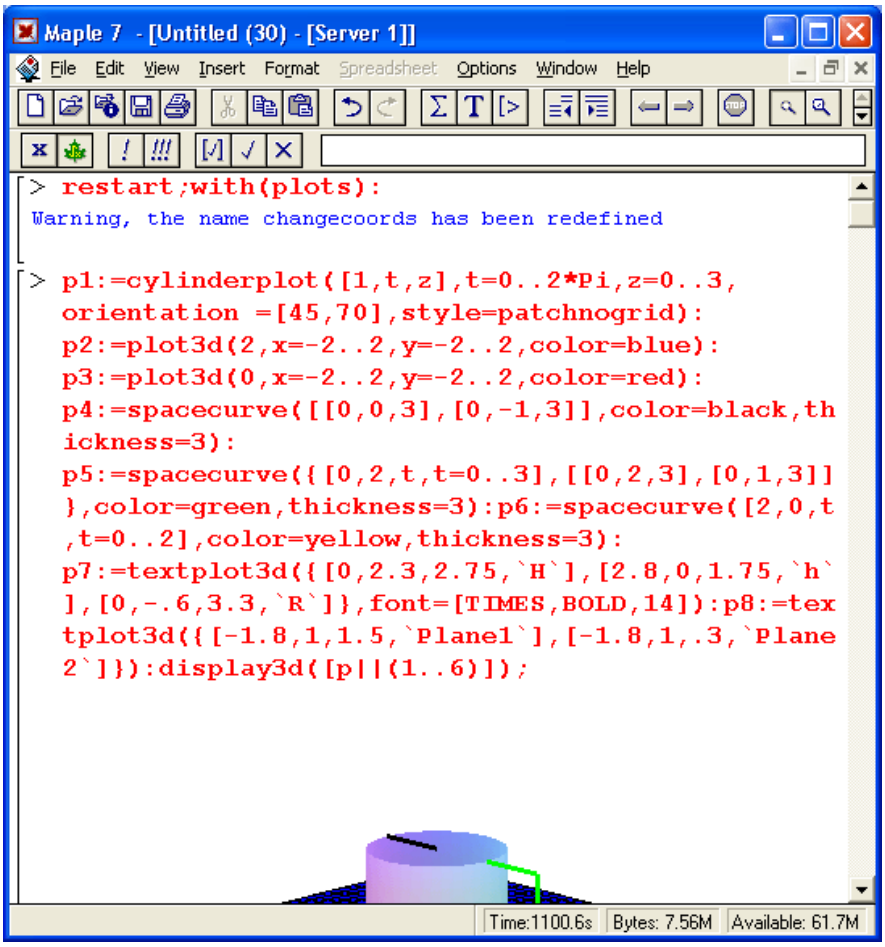


Рис. 12.54. Построение стрелок с помощью функции arrow

5.4.Построение сложных комбинированных графиков

В заключение этой главы рассмотрим построение с помощью системы Maple 7 достаточно сложных комбинированных графиков, содержащих различные графические и текстовые объекты. Пример построения такого графика представлен на рис. 12.55.



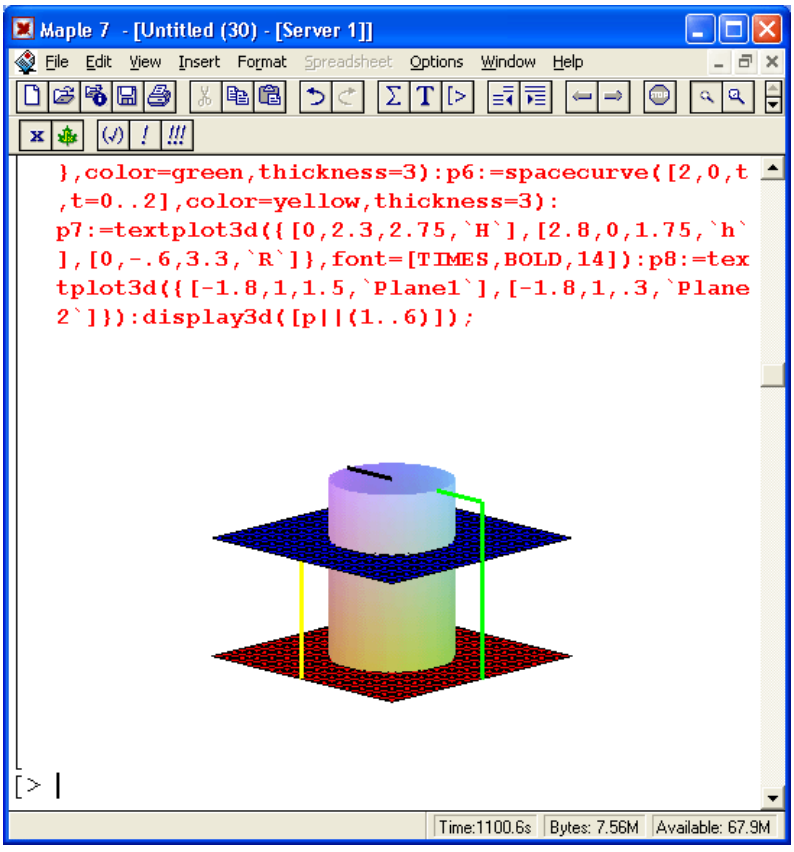


Рис. 12.55. Пример построения сложного объекта, состоящего из 8 графических и текстовых объектов

Представленный на рис. 12.55 объект задает построение восьми графических объектов от p! до p8. Среди них цилиндр, две пересекающие его плоскости и иные (в том числе текстовые) объекты. Обратите внимание на способ вывода этих объектов функцией display3d. Этот пример показывает, что с помощью графических программных средств Maple 7 можно строить достаточно замысловатые графики, которые могут использоваться для визуализации тех или иных геометрических и иных объектов.

5.5. Что нового мы узнали?

В этом уроке мы научились:

- Пользоваться графикой пакета plots.
- Пользоваться техникой анимации графиков пакета plots.
- Применять графику пакета plottools.
- Использовать расширенные средства графической визуализации.
- Использовать расширенную технику создания графической анимации.
- Использовать новую функцию Maple 7 для построения стрелок.
- Строить сложные комбинированные графики.