МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО
СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ имени АБУ РАЙХАНА БЕРУНИ

Б.Р. ТУЛАЕВ

ОСНОВЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

СИСТЕМЫ АВТОМАТИЗИРОВАННОЙ РАЗРАБОТКИ ЧЕРТЕЖЕЙ

УЧЕБНОЕ ПОСОБИЕ

ТАШКЕНТ 2010

УДК 681.3

Основы автоматизированного проектирования: Системы автоматизированной разработки чертежей: Учебное пособие. / Б.Р. Тулаев. – Ташкент: ТашГТУ, 2010.

В учебном пособии изложены принципы автоматизированной разработки чертежей, описаны концепции систем, не вдаваясь в конкретные детали, связанные с работой в конкретных пакетах. Для эффективной работы с существующим программным обеспечением и создания программ, автоматизирующих процесс проектирования, пользователь должен иметь представление не только о среде, в которой он работает, но и о принципах, лежащих в её основе. Фундаментальное знание помогает студенту быстро изучить любую конкретную систему с конкретной средой и использовать её максимально эффективно.

Учебное пособие рассчитано на студентов, обучающихся по направлениям отрасли знаний 500000 – «Инженерия, отрасли обработки и строительства».

Печатается по решению научно-методического совета Ташкентского государственного технического университета имени Абу Райхана Беруни.

Рецензенты: д.т.н., доц. Базаров Б.И. (ТАДИ);

д.т.н., проф. Мамаджанов А.М. (ТашГТУ)

© Ташкентский государственный технический университет, 2010.

ПРЕДИСЛОВИЕ

Бурный рост вычислительной мощности компьютеров и широкое распространение программного обеспечения проектирования и производства привели к тому, что инженеры могут использовать системы автоматизированного проектирования (САПР) для решения повседневных задач. Международная конкуренция, увеличение числа опытных специалистов и повышенные требования к качеству заставляют руководителей предприятий автоматизировать проектирование и производство. Как следствие этого, преподаватели высшей школы чувствуют потребность изменить программу своих курсов, относящихся к проектированию, чтобы научить студентов пользоваться САПР и дать им представления об основных принципах, лежащих в основе этих систем.

Цель этого учебного пособия – изложить эти принципы, описать концепции систем, не вдаваясь в конкретные детали, связанные с работой в конкретных пакетах. Некоторым может показаться, что достаточно научить студентов пользоваться существующими системами, или даже одной системой, наиболее популярный, потому что студент инженерных направлений станет пользователем, а не разработчиком САПР. Дело, однако, в том, что для эффективной работы с существующим программным обеспечением и создания программ, автоматизирующих процесс проектирования, пользователь должен иметь представление не только о среде, в которой он работает, но и о принципах, лежащих в её основе. Фундаментальное знание помогает студенту быстро изучить любую конкретную систему с конкретной средой и использовать её максимально эффективно. Без теоретической подготовки студент встретит серьезные затруднения с терминологией системной документации, а еще большие сложности у него вызовет анализ сообщений об ошибках.

Учебное пособие написано для студентов, обучающихся в высших технических учебных заведениях. Для работы с ним достаточно знать основы программирования, математического

анализа, матричной и векторной алгебры; никаких знаний о собственно САПР у студента не предполагается.

В конце каждой главы приводятся задачи, назначение которых – проверить качество усвоения материала студентами.

ГЛАВА 1. ВВЕДЕНИЕ В САПР

Современные предприятия не смогут выжить во всемирной конкуренции, если не будут выпускать новые продукции *лучшего качества, более низкой стоимости* и за меньшее время. Поэтому они стремятся использовать огромные возможности удобного графического интерфейса, для того чтобы автоматизировать и связать друг с другом задачи проектирования и производства, которые раньше были весьма утомительными и совершенно не связанными друг с другом. Таким образом, сокращается время и стоимость разработки и выпуска продукции. Для этой цели используются технологии *автоматизированного проектирования* (*computer-aided design – CAD*), *автоматизированного производства* (*computer-aided manufacturing – CAM*) и *автоматизированной разработки или конструирования* (*computer-aided engineering – CAE*).

Автоматизированное проектирование *CAD* представляет собой технологию, состоящую в использовании компьютерных систем для облегчения создания, изменения, анализа и оптимизации проектов [14]. Таким образом, любая программа, работающая с компьютерной графикой, так и любое применение, используемое в инженерных расчетах, относится к системам автоматизированного проектирования. Другими словами, множество средств *CAD* проектируется от геометрических программ для работы с формами до специализированных приложений для анализа и оптимизации [17]. Самая основная функция *CAD* – определение геометрии конструкции (детали механизма, архитектурные элементы, электронные схемы, планы зданий и т.п.), поскольку геометрия определяет все последующие этапы жизненного цикла продукта. Для этой цели обычно используются системы разработки чертежей и геометрического моделирования. Вот почему эти системы обычно и считаются системами автоматизированного проектирования. Более того, геометрия, определённая в этих системах, может использоваться в качестве основы для дальнейших операций в системах *CAE* и *CAM*. Это одно из наиболее значительных преимуществ *CAD*,

позволяющее экономить время и сокращать количество ошибок, связанных с необходимостью определять конструкцию с нуля каждый раз, когда она требуется в расчетах. Можно, следовательно, утверждать, что системы автоматизированной разработки рабочих чертежей и системы геометрического моделирования являются наиболее важными компонентами автоматизированного проектирования.

Автоматизированное производство *CAM* – это технология, состоящая в использовании компьютерных систем для планирования, управления и контроля операций производства через прямой или косвенный интерфейс с производственными ресурсами предприятия. Одним из наиболее зрелых подходов к автоматизации производства является числовое программное управление (ЧПИ, *numerical control – NT*). ЧПУ заключается в использовании запрограммированных команд для управления станком, который может шлифовать, резать, фрезеровать, штамповать, изгибать и иными способами превращать заготовки в готовые детали. В наше время компьютеры способны генерировать большие программы для станков с ЧПУ на основании геометрических параметров изделий из базы данных *CAD* и дополнительных сведений, предоставляемых оператором.

Еще одна важная функция систем автоматизированного производства – программирование роботов, которые могут работать на гибких автоматизированных участках, выбирая и устанавливая инструменты и обрабатываемые детали на станках с ЧПУ. Роботы могут также выполнять свои собственные задачи, например, заниматься сваркой, сборкой и переносом оборудования и деталей по цеху.

Планирование процессов также постепенно автоматизируется. План процессов может определять последовательность операций по изготовлению устройства от начала и до конца на всём необходимом оборудовании. Хотя полностью автоматизированное планирование процессов практически невозможно, план обработки конкретной детали вполне может быть сформулирован автоматически, если уже имеются планы обработки аналогичных деталей. Для этого была

разработана технология группировки, позволяющая объединять схожие детали в семейства. Детали считаются подобными, если они имеют общие производственные особенности (гнёзда, пазы, фаски, отверстия и т.д.). Для автоматического обнаружения схожести деталей необходимо, чтобы база данных *CAD* содержала сведения о таких особенностях. Эта задача осуществляется при помощи объектно-ориентированного моделирования или распознавания элементов.

Автоматизированное конструирование *(computer-aided engineering -- CAE)* – это технология, состоящая в использовании компьютерных систем для анализа геометрии *CAD*, моделирования и изучения поведения продукта для усовершенствования и оптимизации его конструкции. Средства *CAE* могут осуществлять множество различных вариантов анализа. Программы для кинематического(их) расчета(ов), например, способны определять траектории движения и скорости звеньев в механизмах. Программы динамического анализа с большими смещениями могут использоваться для определения нагрузок и смещений в сложных составных устройствах типа автомобилей. Программы верификации и анализа логики и синхронизации имитируют работу сложных электронных цепей.

Существует множество программных средств для оптимизации конструкций. Хотя средства для оптимизации могут быть отнесены к классу *CAE*, обычно их рассматривают отдельно. Ведутся исследования возможности автоматического определения формы конструкции путем объединения оптимизации и анализа [13]. В этих подходах исходная форма конструкции предлагается простой, как, например, у прямоугольного двумерного объекта, состоящего из небольших элементов различной плотности, позволяющие достичь определенной цели с учетом ограничений на напряжения. Целью часто является минимизация веса. После определения оптимальных значений плотности рассчитывается оптимальная форма объекта. Она получается отбрасыванием элементов с низкими значениями плотности.

Замечательное достоинство методов анализа и оптимизации конструкций заключается в том, что они позволяют конструктору

увидеть поведение конечного продукта и выявить возможные ошибки до создания и тестирования реальных прототипов, избежав определенных затрат.

Таким образом, технологии *CAD*, *CAM* и *CAE* заключаются в автоматизации и повышении эффективности конкретных стадий жизненного цикла продукта.

Вопросы и задачи

1. Опишите различие между проектной и аналитической моделями.
2. Почему аналитическая модель отличается от проектной?
3. Какие аналитические операции выполняются в рамках процесса разработки?
4. Как используются средства *CAD* в процессе разработки?
5. Какой вариант использования средств *CAD* в процессе производства?
6. Как используются средства *CAM* в процессе производства?
7. Какое главное преимущество использования средств *CAE* в процессе разработки?

ГЛАВА 2. КОМПОНЕНТЫ САПР

Для реализации компьютерно-ориентированного подхода к проектированию нужно специально аппаратное и программное обеспечение. Ключевым аспектом является интерактивное управление формой, поэтому аппаратное и программное обеспечение для интерактивного манипулирования формами относится к числу основных компонентов, составляющих системы *CAD/CAM/CAE*. Графические устройства и периферийные устройства ввода-вывода вместе с обычным вычислительным модулем составляет аппаратное обеспечение системы *CAD/CAM/CAE* (рис. 2.1). Ключевыми программными компонентами являются пакеты, манипулирующие формами или анализирующие их под управлением пользователя в двух или в трех измерениях, одновременно обновляя базу данных.

Рис. 2.1. Компоненты системы *CAD/CAM/CAE*

2.1. Аппаратное обеспечение

Графическое устройство состоит из дисплейного процессора, устройства отображения или дисплейного устройства (монитора) и одного или несколько устройств ввода. Дисплей

(монитор) представляет собой экран, на который выводится графическое изображение, однако вывод конкретного изображения на экран выводится дисплейным процессором. Другими словами, дисплейный процессор получает сигналы, которыми кодируются графические команды, генерируются электронные пучки и направляет их в нужное место монитора, порождая желаемое изображение.

В состав графического устройства обычно входит один или несколько устройств ввода. Помимо клавиатуры к ним относятся мышь, спейсбол и цифровой планшет с пером и роликом (рис. 2.2). Эти устройства ввода призваны способствовать интерактивному манипулированию формами, давая пользователю возможность вводить графические данные в компьютер непосредственно. Каждое графическое устройство обычно подключается к устройствам вывода, например, к плоттеру или цветному лазерному принтеру. Эти устройства могут использоваться несколькими графическими устройствами совместно.

а б в

Рис. 2.2. Примеры устройства ввода:
а – мышь; *б* – цифровой планшет с пером и роликом; *в* – спейсбол

Векторные графические устройства, появившиеся в середине 60-х годов XX века, состоит из дисплейного процессора, дисплейного буфера памяти и электронно-лучевой трубки (рис. 2.3).

Рис. 2.3. Компоненты векторного графического устройства

Растровые графические устройства появились в середине 70-х годов XX века в результате широкого распространения телевизионных технологий. С тех пор они стали основным видом графических устройств благодаря высокому соотношению «качество-цена». Основные принципы их функционирования коротко можно описать следующим образом.

Дисплейный процессор принимает графические команды от приложения, преобразует их в точечное изображение или растр, после чего сохраняет растр в раздел памяти, который называется *буфером кадра (frame buffer)* (рис. 2.4). Получить наглядное представление о растровом изображении можно, если вплотную приблизиться к телевизору. Размеры точек определяются установленным разрешением. Растровые графические устройства должны хранить в своей памяти изображение в виде растра, в

отличие от векторных, хранящих лишь дисплейные файлы. Поэтому требования к памяти у этих двух видов устройств отличаются, как и методы обновления изображения на экране.

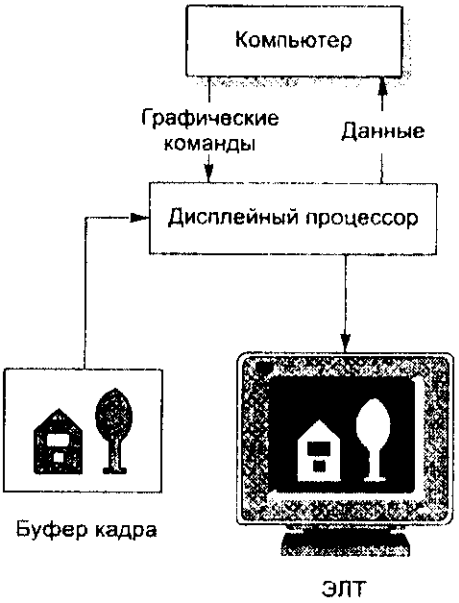


Рис. 2.4. Компоненты растрового графического устройства

2.2. Конфигурация аппаратных средств

Графические устройства, описанные в предыдущем параграфе, редко используются поодиночке. Чаще всего они объединяются в кластер того или иного рода, рассчитанный на обслуживание множества пользователей. Существует три основных варианта конфигурации такого кластера.

Первая конфигурация состоит из мейнфрейма (*mainframe*) и множества графических устройств (рис. 2.5). Графические устройства подключаются к мейнфрейму точно так же, как алфавитно-цифровые терминалы в обычных вычислительных центрах. К нему же подключаются и устройства вывода, такие как

принтеры и плоттеры. Этот подход до сих пор используется производителями автомобилей, у которых имеются огромные базы данных, обрабатываемые централизованно. Однако он обладает некоторыми недостатками. Он требует больших начальных вложений в аппаратное и программное обеспечение, да и обслуживание эксплуатационной системы тоже стоит недёшево. Обслуживание мейнфрейма всегда включает в себя расширение системной памятки и жёсткого диска, что для мейнфрейма обходится гораздо дороже, чем для небольших компьютеров. Более того, обновление операционной системы тоже оказывается непростой задачей. Программы *CAD/CAM/CAE* требуют довольно частой замены в связи с выходом новых, гораздо более мощных версий и альтернатив, а также из-за ошибок при первичном выборе программного обеспечения. Программы *CAD/CAM/CAE* для мейнфреймов стоят намного дороже, чем аналогичные программы для меньших компьютеров. Еще одним серьезным недостатком централизованных вычислений является нестабильность времени отклика системы. В конфигурации с мейнфреймом приложения пользователей, относящихся к разным графическим устройствам, конкурируют друг с другом за вычислительные ресурсы мейнфрейма. Поэтому время отклика для любого конкретного графического устройства зависит от того, какие задачи были запущены с другого устройства. Иногда отклика может быть слишком большим для интерактивной работы с графикой, особенно когда другие пользователи решают сложные вычислительные задачи.

<table>
<tr><th colspan="4">Мейнфрейм</th></tr>
<tr><td>Графическое устройство</td><td>Графическое устройство</td><td>Графическое устройство</td><td rowspan="2">Плоттер</td></tr>
<tr><td>Интерактивные устройства ввода</td><td>Интерактивные устройства ввода</td><td>Интерактивные устройства ввода</td></tr>
</table>

Рис. 2.5. Мейнфрейм с графическими устройствами

Вторая конфигурация состоит из автоматизированных рабочих мест проектировщика (рабочих станций – *workstations*), объединенных в сеть (рис. 2.6). К этой же сети подключаются устройства вывода – плоттеры и принтеры. Рабочая станция – это графическое устройство с собственными вычислительными ресурсами. Такой подход в настоящее время используется очень широко, потому что в области изготовления рабочих станций прогресс идет огромными темпами, да и вообще имеется тенденция к распределению вычислений. Производительность рабочих станций удваивается каждый год при сохранении их цены. Данный подход обладает и другими преимуществами. Пользователь может работать с любой станцией сети, выбирает её в соответствии со своей задачей, причем системные ресурсы не будут зависеть от задач других пользователей. Еще одно преимущество – отсутствие необходимости в крупных первичных вложениях. Количество рабочих станций и программных пакетов может увеличиваться постепенно, по мере роста потребности в ресурсах *CAD/CAM/CAE*. Это очень выгодно, потому что стоимость оборудования постоянно падает.

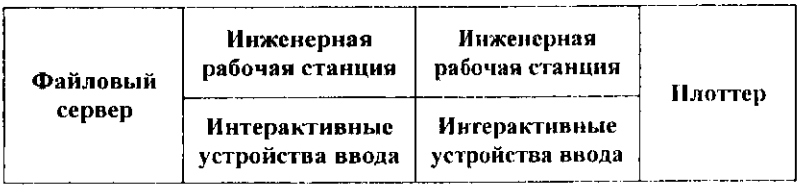


Рис. 2.6. Рабочие станции, присоединенные к сети

Третья конфигурация аналогична второй за тем исключением, что вместо рабочих станций используются персональные компьютеры с операционными системами *Windows*... и *NT*. Конфигурации на базе персональных компьютеров популярны в небольших компаниях, особенно, если выпускаемые продукты состоят из небольшого количества деталей ограниченной сложности, а также в компаниях, использующих системы *CAD* главным образом для построения чертежей. По мере

того как различие между персональными компьютерами и рабочими станциями сглаживается, стирается и различие между вторым и третьим типами конфигурации.

2.3. Программные компоненты

Любая программа, используемая в жизненном цикле продукта для сокращения времени и стоимости разработки продукта, а также для повышения его качества, может быть отнесена к классу *CAD/CAM/CAE*. В основе лежат программы *CAD*, позволяющие конструктору создавать формы и манипулировать ими на мониторе в интерактивном режиме, сохраняя результаты в базе данных. Однако, в принципе любая программа, облегчающая процесс разработки, может быть названа программой *CAD*. Например, специализированное приложение, предназначенное для автоматизации проектирования конкретной детали и механизма, также считается приложением *CAD*.

Любая программа, используемая в процессе производства продукта, считается средством *CAM*. Таким образом, к *CAM* относятся программы для планирования, управления и контроля производственных операций через прямой или косвенный компьютерный интерфейс с производственными ресурсами предприятия. Это может быть программа, формирующая план процесса производства детали, или программа, которая пишет программу для станка с ЧПУ, моделирует движение резца и контролирует работу станка в процессе обработки поверхностей детали.

Программы *CAE* используются для анализа геометрии конструкции и позволяют разработчику моделировать и изучать поведение продукта для улучшения и оптимизации проекта. Типичным примером является программа для расчета напряжений, деформации или теплопередачи в детали методом конечных элементов.

Форма, созданная в системе *CAD*, может использоваться для приложений, рассчитывающих траекторию движения фрезы или выполняющих трёхмерный анализ напряжений, только в том случае, если она строилась как трёхмерная. По этой причине

конструкторы часто начинают работать сразу с трёхмерной моделью детали в системах геометрического моделирования.

2.4. САПР на базе Windows

Когда рынок программного обеспечения *CAD/CAM/CAE* достиг зрелости, ситуация на нём радикально изменилась. Инженеры и производители свыклись с тем, что им нужно нечто большее, нежели средства построения двумерных чертежей. Они давно ждали появления гибких систем построения трёхмерных чертежей, которые могли бы использоваться на всех этапах, от проектирования до производства. До недавних пор промышленные приложения доминировали на рынке традиционных средств *CAD* высшего класса. К счастью, персональные компьютеры со временем стали невероятно быстрыми и мощными, а многие разработчики программного обеспечения начали выпускать хорошие программные продукты, использующие преимущества превосходной графической среды *Microsoft Windows 95* и *NT*. Первые программные продукты этой категории были выпущены в 1995 г., а первые версии большинства продуктов датированы 1996 г.

Все новые продукты обладали следующими общими особенностями. *Во-первых*, они разрабатывались с максимальным использованием функций *Windows 95* и *Windows NT* и потому их интерфейсы получались схожими с интерфейсами других программ *Microsoft*. Фактически эти программы были похожи на другие продукты *Microsoft* для автоматизации конторского труда, поэтому пользователи чувствовали себя комфортно в знакомой среде и быстро осваивали их. Кроме того, новые программы поддерживали *функции внедрения и связывания объектов (object Linking and Embedding – OLE)*, характерные для офисных пакетов *Microsoft*. Таким образом, любое изображение трёхмерной детали или устройства, созданное в пакете геометрического моделирования, может использоваться другими программами *Microsoft*.

Во-вторых, в новых системах использовался компонентный подход, согласно которому важнейшие компоненты программного

обеспечения выбираются из доступных программ, сосредоточивая своё внимание на деталях, относящихся непосредственно к проектированию. Такой подход позволяет сократить время на разработку, причем пользователю будут предоставлены именно те возможности, которые понадобятся ему в процессе проектирования конкретного продукта.

В-третьих, новые системы основаны на объектно-ориентированной технологии, три аспекты которой рассмотрим особо. С точки зрения программирования объектно-ориентированная технология означает написание модульных программ таким образом, чтобы обеспечить независимое повторное использование модулей. Каждая функция может быть написана на этом языке таким образом, что она будет действовать как независимое целое. Объектно-ориентированная технология определяет также интерфейс между системой и пользователем. Объектная ориентированность пользовательского интерфейса означает, что каждый элемент интерфейса самостоятельно реагирует на изменения в ситуации и на действия пользователя. Это значительно облегчает работу пользователя с системой. Объектно-ориентированная технология используется и для эффективного хранения данных. В обычных системах *CAD* данные о детали обычно хранятся в нескольких файлах: один файл используется для геометрической формы, другой – для сетки конечных элементов, третий – для траектории движения фрезы станка с ЧПУ и т.д. В объектно-ориентированных системах все данные, относящиеся к одной детали, хранятся в одном файле. При сохранении одинаковых данных в разных файлах происходит избыточное дублирование, а в объектно-ориентированных системах этого удаётся избежать, что приводит к значительной экономии памяти.

В-четвёртых, системы поддерживают либо параметрическое, либо вариационное моделирование. Оба подхода позволяют пользователю определять форму, задавая ограничения, а не характеристики отдельных элементов этой формы. Единственное различие в том, что в одном случае ограничения учитываются одновременно, а в другом последовательно.

Примером непосредственной работы с элементами формы является определение прямоугольника как двух наборов параллельных отрезков, находящихся на конкретном расстоянии друг от друга. Однако тот же прямоугольник может быть определен при помощи ограничений, например, заданием условия перпендикулярности смежных отрезков и расстояния между отрезками. Многие системы поддерживают возможность параметрического или вариационного моделирования, воспринимают очевидные ограничения, такие как перпендикулярность и параллельность, непосредственно из начального эскиза пользователя, позволяя уменьшить объем вводимых данных. В этом случае от пользователя требуется только ввести размеры, после чего он может изменять форму, варьируя эти размеры. Такая функция системы называется *моделированием по размерам (dimension-driven modeling)*. Это означает, что геометрия определяется размерами детали и может быть легко изменена переопределением этих размеров. В таких системах пользователь может также указывать отношения между размерами, которые будут сохраняться при изменении самих размеров. Это один из механизмов, позволяющих сохранить идеи разработчика в конечном продукте. Однако, задать все ограничения, определяющие геометрию, может быть достаточно затруднительно, особенно для сложной детали. В таких ситуациях системам, поддерживающим параметрическое или вариационное моделирование, требуются дополнительные сведения для проектирования.

Наконец, в системы встраивается поддержка совместного проектирования через Интернет. Эта поддержка позволяет удалённым пользователям работать над одной и той же деталью, имея перед глазами её модель на своих экранах. Разработчики могут также проверять проект в целом, сравнивая свои детали с деталями других разработчиков.

Вопросы и задачи

1. Перечислите компоненты графического устройства и объясните назначение каждого их них.
2. Перечислите преимущества аппаратной конфигурации, состоящей из объединенных в сеть рабочих станций.
3. Перечислите доступные Вам программы для построения двумерных чертежей.
4. Перечислите доступные Вам системы геометрического моделирования.
5. Перечислите доступные Вам системы *CAM*. Коротко опишите функции каждого программного продукта.
6. Перечислите доступные Вам системы *CAE*. Коротко опишите функции каждого программного продукта.
7. Перечислите общие особенности всех новых продуктов.

ГЛАВА 3. ОСНОВНЫЕ КОНЦЕПЦИИ ГРАФИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Интерактивное манипулирование формами составляет значительную часть работы с системами CAD/CAM/CAE. Следовательно, важнейшей составляющей таких систем является программное обеспечение, создающее графическое изображение на экране монитора. Поэтому нам придется изучить терминологию и основные понятия графического программирования.

3.1. Графические библиотеки

Термин *программирование на компьютере* (*computer programming*) раньше означал написание «сочинения» на языке компьютерных команд в соответствии с предопределенными правилами грамматики. В ответ на вводимые числа выполняемое «сочинение» порождало ожидаемые числа и символы на терминале или в файле данных. В наши дни на входе и выходе «сочинений» все чаще находится графическая информация. Такое программирование называется *графическим* (*graphics programming*), а область его применения – *компьютерной графикой* (*computer graphics*).

Помимо основного программного обеспечения, необходимого для обычного программирования (операционная система, редактор и компилятор), графическое программирование требует наличия специальных графических программ. Графические программы делятся на два класса: драйверы устройств и графические библиотеки.

Драйвер устройства может рассматриваться как набор аппаратно-зависимых кодов, непосредственно управляющих процессором графического устройства таким образом, что электронный пучок оказывается направленным в нужное место (см. главу 2). Драйверы обязательно являются аппаратно-зависимыми, то есть жестко привязанными к конкретным графическим процессорам. Примерно то же самое можно сказать об ассемблере, конкретный вид которого может выполняться только на процессорах одной и той же модели. То же происходит, если

графическая программа использует драйвер устройства непосредственно (рис. 3.1). Такую графическую программу при переходе на другое графическое устройство придется переписывать с использованием новых команд драйвера. Более того, команды драйвера устройства весьма примитивны, поэтому такая программа была бы очень длинной, если бы она должна была решать какую-либо сложную задачу. К тому же программа эта получилась бы плохо читаемой.

Программисты предпочитают писать программы на языках высокого уровня. Графическое программирование не могло стать исключением, особенно если представить все неудобства, связанные с использованием команд драйвера низкого уровня. Поэтому с графическими устройствами стали поставляться библиотеки, получившие название *графических* (*graphics libraries*). Графическая библиотека, как, например, и математическая, представляет собой набор подпрограмм, предназначенных для решения определенных задач. Конкретная подпрограмма может изображать на экране прямую, круг или иной объект. Графическая библиотека основывается на командах драйвера устройства (рис. 3.2). Каждая подпрограмма создается с использованием поддерживаемого набора команд драйвера. Например, подпрограмма, изображающая круг, может быть составлена из отдельных команд драйвера, рисующих на экране точки или короткие отрезки.

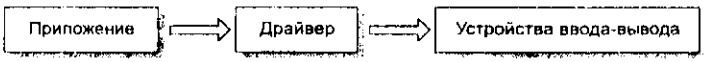


Рис. 3.1. Непосредственное использование драйвера устройств

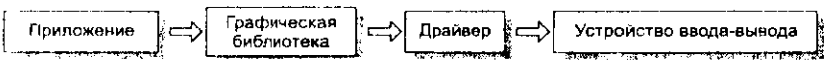


Рис. 3.2. Использование графической библиотеки

Подпрограммы графической библиотеки могут использоваться точно так же, как подпрограммы математической. Нужная подпрограмма вызывается из основной программы

аналогично тому, как вызываются функции синуса и косинуса, когда программисту требуется вычислить их значения. Одна из проблем использования подпрограмм графической библиотеки связана с тем, что их названия и способы вызова (входные и выходные аргументы) у каждой библиотеки свои. Это не создавало бы трудностей, если бы одна графическая библиотека могла работать со всеми существующими устройствами, что теоретически было бы возможным, если бы все существующие драйверы устройств поддерживали ее. Однако по некоторым причинам производители программного обеспечения не хотят или не могут создать графическую библиотеку, которая могла бы работать со всеми драйверами, а потому у каждой библиотеки имеется свой круг поддерживаемых драйверов. Следовательно, такая библиотека может работать лишь с ограниченным набором графических устройств, а графические программы, рассчитанные на работу со множеством устройств, приходится переписывать с использованием нескольких библиотек.

3.2. Системы координат

Для вывода изображения объекта на экран графического устройства необходимо решить две основные задачи:

- указать положение всех точек объекта в пространстве;
- определить положение их образов на мониторе.

Для задания положения точек в пространстве и на мониторе используются системы координат. Важно понимать, как связаны между собой различные системы координат. Особенно это важно для проектирования трехмерного объекта на плоский экран. Проекция на экране строится по тем же законам, что и проекция реального объекта на сетчатке человеческого глаза.

Первой среди систем координат мы рассмотрим *систему координат устройства* (*device coordinate system*), которая определяет положение точки на экране. Эта система состоит из горизонтальной оси *u* и вертикальной оси *v* (рис. 3.3). Обратите внимание, что начало отсчета, может выбираться произвольно. Осей *u* и *v* достаточно для задания положения любой точки экрана, поэтому третья ось, перпендикулярная первым двум, не

определяется. Положение любой точки задается двумя целыми числами *u* и *v*, равными числу пикселов между началом координат и точкой по осям *u* и *v*. Однако одна и та же точка может задаваться разными парами *u* и *v* в зависимости от положения начала координат, направления осей и масштаба. Эти параметры для разных графических устройств устанавливаются достаточно произвольно (см. рис. 3.3). Поэтому аппаратные координаты в графической программе могут потребовать изменения при смене графического устройства.

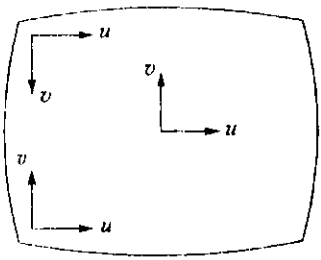


Рис. 3.3. Системы координат устройства

Виртуальная система координат устройства (*virtual device coordinate system*) позволяет избежать описанной выше проблемы. Виртуальная система координат устройства фиксирует точку отсчета, направление и масштаб осей для всех рабочих станций. Слово «виртуальный» означает, что данная система отсчета существует только в воображении программиста. Обычно начало отсчета располагается в левом нижнем углу экрана, ось *u* откладывается вправо, а ось *v* – вверх. Обе координаты могут изменяться в диапазоне от нуля до единицы. Точка, положение которой задается в виртуальной системе координат, на любом экране будет попадать в одно и то же место. Это дает программисту возможность единообразно определять формы, не заботясь о конкретных системах координат устройств. Графическая программа передает виртуальные координаты подпрограмме драйвера устройства, которая преобразует их в координаты конкретного устройства.

Виртуальная и обычная системы координат устройства позволяют задавать положение точки на плоском экране. Займемся теперь системами координат для работы с трехмерным пространством. Основных трехмерных систем координат всего три:

- *внешняя система координат* (*world coordinate system*);
- *система координат модели* (*model coordinate system*);
- *система координат наблюдателя* (*viewing coordinate system*).

Внешняя, или мировая система координат (*world coordinate system*), – это опорная система, используемая для описания интересующего нас мира. Внешней она является по отношению к объектам этого мира. Например, такая система может использоваться для описания расположения и ориентации парт, стульев и доски, если интересующий нас мир представляет собой класс.

Следующим шагом является описание формы каждого объекта мира. Форма объекта определяется координатами всех или некоторых характеристических точек объекта по отношению к системе координат, связанной с ним, – *системой* координат модели (*model coordinate system*). Координаты точек объекта, определенные таким образом, не изменяются даже тогда, когда объект перемещается или вращается в пространстве. Они действительно зависят только от формы объекта. Система координат модели перемещается вместе с тем объектом, к которому она привязана. Поэтому форма каждого объекта определяется в его собственной системе координат модели. Расположение и ориентация любого объекта задаются относительным положением и ориентацией модельной системы координат данного объекта по отношению к внешней системе координат. Относительное расположение и ориентация систем координат определяются матрицей преобразования, о которой будет рассказано в разделе 3.7. Наличие внешней системы координат и модельных систем для всех объектов полностью определяет мир, то есть расположение и форму всех объектов данного мира. Другими словами, применение матриц

преобразования позволяет получить координаты любой точки любого объекта во внешней системе.

Следующий шаг – проектирование трехмерных объектов или их точек на монитор подобно тому, как они проектируются на сетчатку человеческого глаза. В компьютерной графике используется два вида проекций: перспективная и параллельная (рис. 3.4).

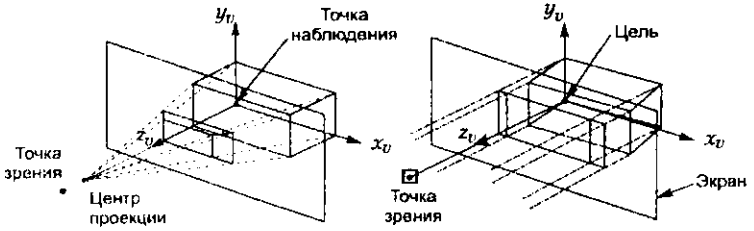


Рис. 3.4. Два вида проекций: *а* – перспективная; *б* – параллельная

Оба вида требуют задания двух точек: точки зрения и точки наблюдения. *Точка зрения* (*viewpoint*) – это глаз наблюдателя. *Точка наблюдения* (*view site*) – это точка объекта, определяющая направление «луча зрения». Вектор, проведенный от точки зрения к цели, задает направление наблюдения.

В *перспективной проекции* (*perspective projection*) все точки рассматриваемого объекта соединяются с центром проекции, который обычно лежит на линии, соединяющей точку зрения и цель[1]. Точки пересечения этих линий с экраном образуют проекцию. Экран располагается между точкой зрения и целью. В *параллельной проекции* (*parallel projection*) линии от всех точек объекта проводятся в направлении наблюдателя параллельно направлению наблюдения, а точки пересечения этих линий с экраном формируют проекцию. Экран, как и в перспективной проекции, располагается перпендикулярно направлению проектирования. Такая проекция называется ортогональной[2].

[1] В противном случае проекция называется косоугольной.

[2] В косоугольной проекции ориентация экрана произвольна.

Точки проекции, получаемые любым из описанных методов, легко могут быть рассчитаны, если координаты точек проецируемого объекта даны в системе координат $x_v y_v z_v$ (рис. 3.4). Например, координаты точек параллельной проекции объекта попросту равняются соответствующим координатам X_v и Y_v точек объекта. Система координат $x_v y_v z_v$ называется *наблюдательской* (*viewing coordinate system*), поскольку она облегчает расчет проекции наблюдения. Наблюдательская система координат строится таким образом, чтобы обладать перечисленными ниже характеристиками. Начало этой системы координат располагается в рассматриваемой точке, ось z_v направлена из начала координат в точку зрения, а ось y_v параллельна вертикальной оси экрана (см. рис. 3.4).

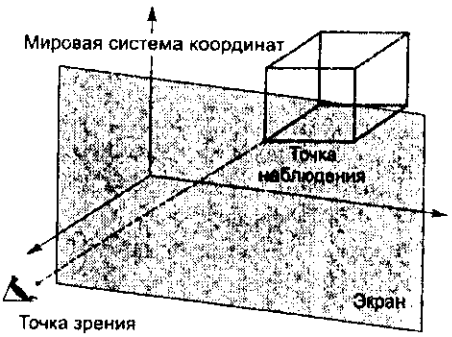


Рис. 3.5. Точка зрения и точка наблюдения

Третья ось, x_v определяется как векторное произведение первых двух. У большинства людей вертикальное направление в пространстве естественным образом ассоциируется с вертикальным направлением на экране, поэтому ось y_v считается проекцией вертикального вектора из пространства на экран. В большинстве графических библиотек пользователю приходится задавать вектор вертикали в пространстве (*up vector*) в мировых координатах. Положение точки зрения и точки наблюдения также задается в мировых координатах (рис. 3.5).

После определения наблюдательской системы координат и вычисления координат точек всех интересующих нас объектов остается только вычислить положение их проекций на экране. Мы уже знаем, что для параллельной проекции это сделать очень легко. Поэтому мы займемся описанием процедуры вычисления координат в перспективной проекции. Рассмотрим виды сверху и сбоку (рис. 3.4, *а*, рис. 3.6). Интересующая нас точка на рисунке выделена, ее координаты мы обозначим как X_v, Y_v, Z_v.

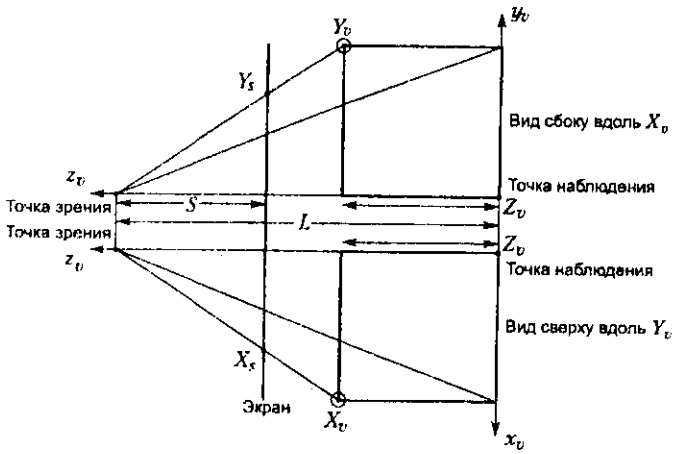


Рис. 3.6. Расчет проекции

Из подобия треугольников следует, что

$$X_S = \frac{S}{L - Z_v} X_v; \qquad (3.1)$$

$$Y_S = \frac{S}{L - Z_v} Y_v. \qquad (3.2)$$

В формулах (3.1) и (3.2) X_S *и* Y_S – расстояния до проекции выбранной точки. Расстояния измеряются в горизонтальном и вертикальном направлениях от точки, где ось z_v пересекается с экраном. Далее L – расстояние между точкой наблюдения и центром проекции, а S – расстояние между центром проекции и

экраном. Формулы (3.1) и (3.2) показывают, что точка с большими значениями Z_v будет иметь большие значения X_s и Y_s, благодаря чему далекий отрезок кажется меньше; чем близкий той же длины. Расстояния X_s и Y_s преобразуются в виртуальные координаты устройства с учетом желаемого положения центра изображения и его размеров (на мониторе).

Взаимоотношения перечисленных выше систем координат иллюстрирует рис. 3.7.

Как уже говорилось, системы координат связаны друг с другом матрицами преобразования. Так, положение и ориентация каждой из модельных систем координат по отношению к мировой задаются соответствующими матрицами преобразований. Наблюдательская система координат также может быть определена относительно мировой при помощи матрицы преобразования, если задать положение точек зрения и наблюдения, а также вектор вертикали. Процедура расчета точек проекции с использованием матриц преобразования выглядит следующим образом. Сначала координаты проецируемой точки преобразуются из модельных в мировые при помощи матрицы преобразования, определяющей переход от модели, к которой относится точка, к мировой системе координат. Эта операция (рис. 3.8) называется *преобразованием модели* (*model transformation*). Затем координаты этой точки преобразуются из мировой системы координат в наблюдательскую. Эта операция (см. рис. 3.8) называется *преобразованием наблюдения* (*viewing transformation*). Наконец, координаты в наблюдательской системе координат преобразуются в значения X_s и Y_s по формулам (3.1) и (3.2), а затем – в виртуальные координаты устройства. Эта операция (рис. 3.8) называется *преобразованием проекции* (*projection transformation*). Наконец, виртуальные координаты устройства преобразуются в обычные подпрограммой драйвера. Результат показан на рис. 3.8.

Все эти преобразования обычно выполняются внутри графической библиотеки, а программисту приходится только указывать сведения, необходимые для проведения преобразований. Например, трансляции и повороты объектов учитываются при преобразовании модели, положение точки зрения, точки

наблюдения и вектора вертикали – при преобразовании наблюдения, а тип проекции, расположение центра проекции и экрана – при преобразовании проецирования. Однако графические библиотеки примитивного уровня могут потребовать от программиста самостоятельного написания кода для всех этих преобразований.

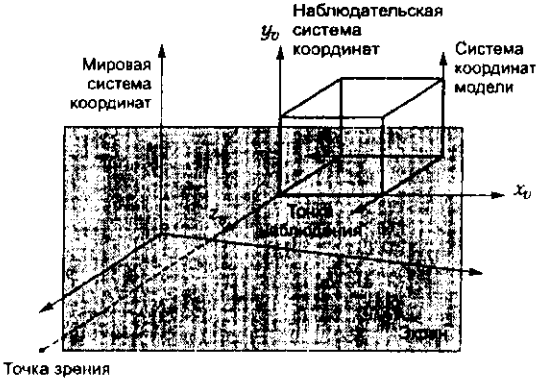


Рис. 3.7. Соотношение систем координат

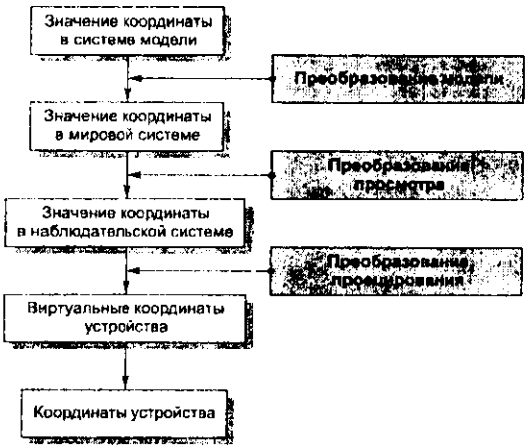


Рис. 3.8. Преобразования между системами координат

3.3. Окно и видовой экран

Термин *окно* (*window*) в сетевой компьютерной среде обозначает область экрана монитора рабочей станции, посредством которого пользователь взаимодействует с вычислительными ресурсами, подключенными к той же сети. В компьютерной графике этот термин имеет иное значение. Окно – это область пространства, проецируемая на монитор. Объекты, находящиеся вне окна, на мониторе не появляются. В этом смысле оно подобно окну дома, через которое человеку, сидящему внутри дома, видна лишь часть внешнего мира. Вероятно, эта аналогия была основанием для выбора соответствующего термина. Окно обычно определяется как прямоугольник, лежащий на экране и заданный значениями X_v, и Y_v, в системе координат просмотра (рис. 3.9 и рис. 3.10). Видимая область пространства, называемая *просматриваемым объемом* (*viewing volume*), зависит от типа проекции. Для параллельной проекции эта область имеет форму параллелепипеда, а для перспективной – форму пирамиды.

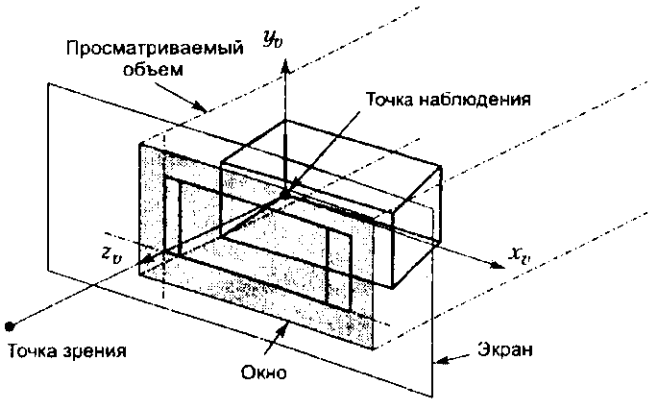


Рис. 3.9. Окно и просматриваемый объем для параллельной проекции

Просматриваемый объем при проектировании может давать довольно сложное изображение, поскольку в него могут попадать ненужные объекты, расположенные вблизи наблюдателя или вдали

от него. Иногда бывает удобно ограничить этот объем двумя плоскостями, из которых одна располагается ближе, а другая – дальше (рис. 3.11). Для параллельной проекции ближняя и дальняя плоскости определяются так же, как и для перспективной.

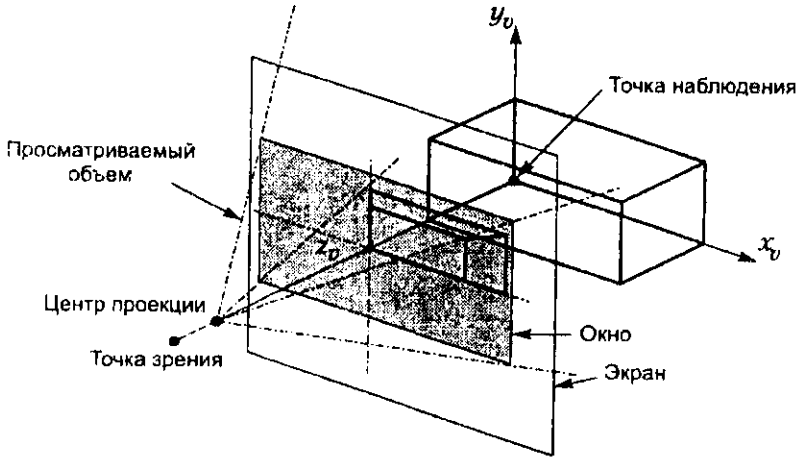


Рис. 3.10. Окно и просматриваемый объем для параллельной проекции

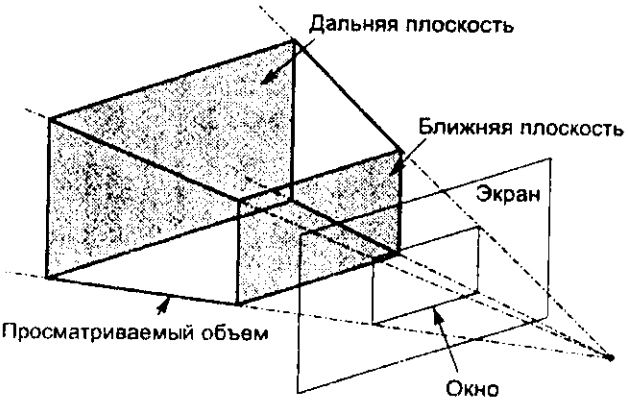


Рис. 3.11. Ближняя и дальняя плоскости

Видовой экран (*view port*) – это область экрана, где будет отображаться проецируемое изображение (рис. 3.12). В эту область проецируется просматриваемый объем, определяемый «обычным» окном. Отображение состоит из трансляции и масштабирования, учитывающих расстояние между центром видового экрана и центром монитора, а также разницу размеров окна и видового экрана. Другими словами, значения X_s и Y_s, полученные по формулам (3.1) и (3.2), должны быть увеличены или уменьшены таким образом, чтобы центр окна попадал в центр видового экрана, а не в центр монитора. Кроме того, они должны быть подвергнуты масштабированию, чтобы четыре граничные точки окна стали четырьмя граничными точками видового экрана. Соотношение сторон у окна должно быть таким же, как и у видового экрана, в противном случае изображение будет искажено, и круг, например, превратится в эллипс.

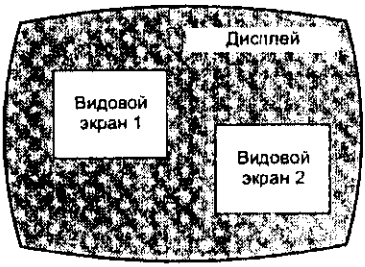


Рис. 3.12. Видовые экраны

Ниже приведен пример кода, задающего окно[1] и видовой экран с использованием графической библиотеки OpenGL. В OpenGL окно просмотра также определяется как трехмерный объем. Однако результат получается таким, как если бы третье измерение окна просмотра просто игнорировалось.

[1] Это «окно» означает область монитора, через которую пользователь взаимодействует с компьютером. Окно открывается и обрабатывается диспетчером операционной системы (например, клиентом X window или Microsoft Windows).

```
static GLint viewport[] = {0, 0, 400, 400};
   /* Видовой экран задается координатами окна[1]. Первый и второй аргументы
   определяют положение левого нижнего угла окна просмотра, а третий
   и четвертый - размер прямоугольника. */
static GLclamped depth_range[] = {0.0, 1.0};
   /* Первый и второй аргументы представляют собой поправки к минимальному
   и максимальному значениям, которые могут храниться в буфере глубины. */
static GLdouble viewing_volume[] = {-100.0, 100.0, -100.0, 100.0, -10.0, 100.0};
   /* Горизонтальный и вертикальный размеры окна задаются в координатах
   просмотра (аргументы 1-4). Пятый и шестой аргументы определяют
   расстояние от экрана до ближней и дальней плоскостей соответственно. */
glOrtho(viewing_volume[0], viewing_volume[1], viewing_volume[2], viewing_volume[3],
viewing_volume[4], viewing_volume[5]);
   /* Определяет тип проекции - параллельный - и создает матрицу
   ортографического параллельного просматриваемого объема, после чего
   умножает на него текущую матрицу. */
glViewport(viewport[0], viewport[1], viewport[2], viewport[3]);
   /* Определяет прямоугольник окна, открытого диспетчером, после чего
   отображает в это окно готовое изображение. Если glViewport не
   используется, окно просмотра по умолчанию считается совпадающим со всем
   открытым окном. */
glDepthRange(depth_range[0],depth_range[1]);
   /* Определяет кодирование z-координат при преобразовании просмотра.
   Значения z-координаты масштабируются этой командой так, чтобы
   они лежали в определенном диапазоне. */
```

3.4. Примитивы

Примитивы (*primitives*) – это элементы графики, которые могут отображаться графической библиотекой. В каждой библиотеке набор примитивов свой, поэтому в данном разделе мы рассмотрим только наиболее общие примитивы, поддерживаемые большинством графических библиотек.

3.4.1. Отрезок

Для отображения *отрезка прямой* (*линии – line*) необходимо задание координат двух его концов. В большинстве графических библиотек координаты концов могут задаваться в трехмерном пространстве; проецирование на плоскость экрана осуществляется автоматически. Можно указывать атрибуты отрезка: тип, толщину, цвет и другие. Типы отрезков, поддерживаемых большинством графических библиотек, изображены на рис. 3.13. Для систем автоматизированной разработки чертежей поддержка этих типов линий совершенно необходима, поскольку они часто используются в машиностроительных и архитектурных чертежах и электрических схемах.

В библиотеках GKS, PHIGS и OpenGL одной из базовых функций является *ломаная* (*polyline*), представляющая собой набор соединенных друг с другом отрезков. Координаты концов отрезков, составляющих ломаную,

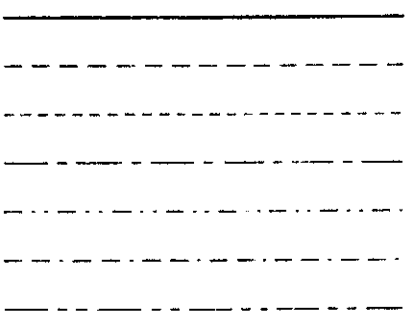


Рис. 3.13. Различные виды отрезков

задаются в виде матрицы. В случае ломаной, состоящей всего из одного отрезка, в матрицу помещаются координаты двух его концов.

$$[P]=\begin{bmatrix} X_1 & Y_1 & Z_1 \\ X_2 & Y_2 & Z_2 \\ \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n \end{bmatrix}.$$

Примеры использования функций построения ломаной из библиотек PHIGS и OpenGL приведены ниже.

```
PHIGS
Pint        num_of_points = 10;
     /* Количество точек в ломаной. */
Ppoint3     point3[] = {
                {0.0, 0.0, 0.0},
                {10.0, 20.0, 15.0},.....
                {1.0, 3.0, 6.5}};
     /* Координаты концов отрезков ломаной. */
Ppolyline3(num_of_points, point3);
     /* Рисует заданную ломаную. */
```

```
OpenGL
GLdouble   point[][3] = {
                  {0.0, 0.0, 0.0},
                  {10.0, 20.0, 15.0},.....
                  {1.0, 3.0, 6.5}};

     /* Координаты концов отрезков ломаной. */
glBegin(GL_LINE_LOOP);
    . glVertex3dv(&point[0][0]);
      glVertex3dv(&point[1][0]);


      glVertex3dv(&point[9][0]);
glEnd();
      /* Построение ломаной по заданным точкам (10 шт.) */
```

3.4.2. Многоугольник

Многоугольник – это то же самое, что и ломаная, за небольшим исключением: первая и последняя строки матрицы вершин $[P]$ должны быть одинаковыми (соответствующие им точки совпадают). Того же результата можно было бы достичь и с использованием функции построения ломаной, однако многоугольник, построенный при помощи специальной функции, распознается системой как объект, имеющий внутреннюю и внешнюю части. Внутренняя площадь многоугольника может быть заполнена штриховкой различных видов (рис. 3.14).

Атрибутами многоугольника могут быть цвет внутренней области (цвет заполнения), а также тип, ширина и цвет ломаной, ограничивающей эту область. Хотя функция построения многоугольников может использоваться и для построения кругов и прямоугольников, в большинстве графических библиотек существуют специальные функции, требующие гораздо меньше входных параметров (например, центр и радиус круга или два конца диагонали прямоугольника). Тем не менее, внутри библиотек эти функции реализованы через функцию построения многоугольников.

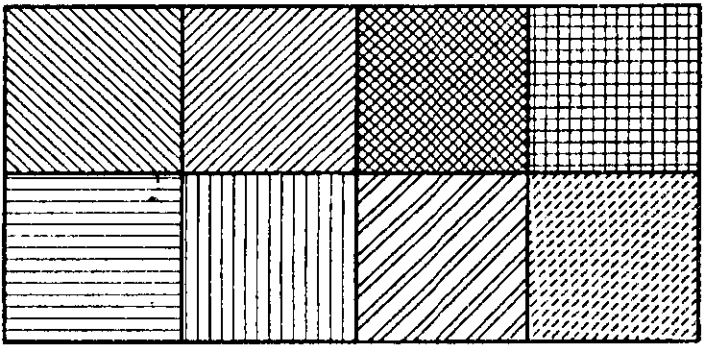


Рис. 3.14. Примеры различных заливок

3.4.3. Маркер

Маркеры обычно используются для выделения точек на графиках. Маркеры, доступные в большинстве графических библиотек, показаны на рис. 3.15.

Рис. 3.15. Примеры маркеров

Тип маркера указывается в качестве атрибута. Полимаркер, как и отрезок, является стандартным объектом в GKS и PHIGS. OpenGL не поддерживает маркеры явно, однако предоставляет механизм сохранения маркеров в растровых файлах и выведения их на экран. Благодаря этому графическая программа, построенная на OpenGL, гораздо лучше переносится на различные платформы. Приведенный ниже образец кода демонстрирует выведение маркера * (звездочка) в PHIGS и OpenGL.

```
PHIGS
pint                num_of_point = 10;
      /* Количество маркеров для отображения. */
Ppoint3      point3[] = {
               {0.0, 0.0, 0.0},
               {10.0, 20.0, 15.0}, ... ,
               {1.0, 3.0, 6.5}};
      /* Координаты маркеров. */
pset_marker_type(PMK_STAR);
      /* Тип маркера - звездочка. */
ppolymarker3(num_of_point, point3);
      /* Построение маркеров по имеющимся данным. */

OpenGL
GLubyte  asterisk[13] = {0x00, 0x00, 0x30, 0x18, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c,
0x0c, 0x18, 0x30};
      /* Задание растрового изображения звездочки. Первой рисуется нижняя
      строка, затем следующая и т. д. */
GLsizei      width = 8;
      /* Ширина растра в пикселах. */
GLsizei      height = 13;
      /* Высота растра в пикселах. */
GLfloat      origin_x = 0.0, origin_y = 0.0;
      /* Задание начала отсчета системы координат растра. Положительные
      значения смещают начало отсчета вверх и вправо, отрицательные - вниз
      и влево. */
GLfloat      incre_x = 10.0, incre_y = 0.0;
      /* Поправки к положению растра после выведения. */
GLfloat      white[3] = {1.0, 1.0, 1.0};
      /* Задание цвета маркера. */
GLfloat      position[2] = {20.5, 20.5};
      /* Координаты маркеров. */
glPixelStorei(GL_UNPACK_ALIGNMENT 1);
      /* Установка режима хранения пикселов, влияющего
      на работу glBitmap. */
glColor3fv(white);
      /* Установка белого цвета маркера. */
glRasterPos2fv(position);
      /* Установка текущего положения растра. */
glBitmap(width, height, origin_x, origin_y, incre_x, incre_y, asterisk);
      /* Выводит растровую картинку, заданную в последнем аргументе,
      представляющем собой указатель на эту картинку. Положение и размер
      растра указываются первыми четырьмя аргументами. */
```

3.4.4. Текст

Большинство графических библиотек поддерживают два вида текста:

- текст для пояснений (экранный или двумерный текст)
- трехмерный текст.

Текст для пояснений всегда располагается в плоскости экрана, поэтому его форма не искажается вне зависимости от угла,

на который он повернут. Трехмерный текст может быть расположен на любой плоскости в трехмерном пространстве. Его положение и ориентация задаются в мировых координатах. Для текста любого вида необходимо задание таких параметров, как шрифт, отношение высоты к ширине и угол наклона букв, а также положение и направление строки текста. Текст может быть представлен символами двух видов: аппаратными и программными. Программный шрифт строится соответствующими графическими программами, заранее сохраняемыми в памяти компьютера. Построение его занимает больше времени, чем построение символов аппаратного шрифта, но зато форма может быть гораздо более замысловатой. Символы аппаратного шрифта состоят из отдельных отрезков, формирующих буквы.

Приведенный ниже листинг демонстрирует построение текстовой строки ABC с помощью библиотек PHIGS и OpenGL. Символы необходимо сохранить в виде растровых картинок, прежде чем их можно будет использовать в OpenGL, точно так же, как и маркеры.

```
PHIGS
Pvector3     direction[2] = {
                  {0.0, 0.0, 0.0},
                  {10.0, 10.0, 10.0}};
     /* Направление строки текста задается двумя радиус-векторами. */
Ppoint3      position = {5.5, 5.0, 5.0};
     /* Положение строки текста. */
ptext3(&position,direction,"ABC");
     /* Построение строки текста с указанными параметрами. */

OpenGL
GLubyte    a[13] = {0x00, 0x00, 0x3f, 0x60, 0xcf, 0xdb, 0xd3, 0xdd, 0xc3, 0x7e, 0x00,
0x00, 0x00};
GLubyte    b[13] = {0x00, 0x00, 0xc3, 0xc3, 0xc3, 0xc3, 0xff, 0xc3, 0xc3, 0xc3, 0x66,
0x3c, 0x18};
GLubyte    c[13] = {0x00, 0x00, 0xf3, 0xc7, 0xc3, 0xc3, 0xc7, 0xfe, 0xc7, 0xc3, 0xc3,
0xc7, 0xfe};
     /* Растровые символы A, B и C. Растр задается снизу вверх
     построчно. */
GLsizei    width = 8;
     /* Ширина растра в пикселах. */
GLsizei    height = 13;
     /* Высота растра в пикселах. */
GLfloat origin_x = 0.0, origin_y=0.0;
     /* Задание начала отсчета системы координат растра.
     Положительные значения смещают начало отсчета вверх и вправо,
     отрицательные - вниз и влево. */
```

```
GLfloat      incre_x = 10.0, incre_y = 0.0;
      /* Поправки к положению растра после выведения. */
GLfloat    white[3] = {1.0, 1.0, 1.0};
      /* Задание цвета текста. */
GLfloat    position[2] = {20.5, 20.5};
      /* Координаты символов. */
glPixelStorei(GL_UNPACK_ALIGNMENT,1);
      /* Установка режима хранения пикселов, влияющего
      на работу glBitmap. */
glColor3fv(white);
      /* Установка белого цвета символов. */
glRasterPos2fv(position);
      /* Установка текущего положения растра. */

glBitmap(width, height, origin_x, origin_y, incre_x, incre_y, asterisk);
      /* Выводит растровую картинку, заданную в последнем аргументе,
      представляющем собой указатель на эту картинку. Положение и размер
      растра указываются первыми четырьмя аргументами. */
```

3.5. Ввод графики

Как уже отмечалось, графической программе может требоваться поддержка ввода графических элементов, таких как точки, отрезки и многоугольники, а не только чисел и текстовых строк. Например, если пользователь хочет вычислить площадь многоугольника на экране или увеличить его размеры, он должен сначала указать интересующий его многоугольник среди прочих объектов, видимых на экране. Для ввода графика используется два вида физических устройств: локатор (устройство ввода координат) и кнопка. *Локатор* (*locator*) передает графической программе информацию о своем положении, то есть о положении курсора. *Кнопка* (*button*) сообщает о действиях пользователя (включении и выключении) в месте текущего положения курсора. В наши дни наиболее популярным устройством графического ввода является мышь, которая выполняет обе функции. Шарик в нижней части корпуса мыши позволяет вводить координаты, а кнопки наверху корпуса передают программе действия пользователя.

Устройство графического ввода может работать в трех режимах: опрос, запрос и выбор. В режиме *опроса* (*sampling*) осуществляется постоянное считывание состояния устройства ввода, прежде всего положения локатора. Например, если вы свободно рисуете на экране, перемещая мышь, она работает в

режиме опроса. Перемещение мыши приводит к непрерывному перемещению курсора по экрану.

В режиме *запроса* (*requesting*) положение локатора считывается только при отправке запроса, которая обычно производится при нажатии на кнопку мыши. Чтобы прояснить различие между режимами опроса и запроса, рассмотрим процесс построения многоугольника путем графического задания координат его вершин при помощи мыши. В этом случае мы перемещаем мышь до тех пор, пока курсор не окажется в нужном месте, после чего нажимаем кнопку. Курсор перемещается по экрану согласно движениям мыши, которая находится при этом в режиме опроса. Координаты вершин передаются графической программе в режиме запроса. У этих режимов есть общее свойство: графической программе передаются координаты мыши или курсора.

В режиме *выбора* (*picking*) устройство графического ввода идентифицирует элемент экрана, на который указывает курсор в момент нажатия кнопки. Графические элементы можно идентифицировать по именам, присвоенным им программистом во время составления программы. Режим выбора очень удобен при редактировании чертежа, уже имеющегося на экране (например, для удаления многоугольников или изменения координат их вершин).

3.6. Дисплейный файл

Дисплейный файл (*display list*) – это группа команд графической библиотеки, сохраненная для последующего выполнения. Большинство команд графических библиотек могут либо помещаться в дисплейный файл, либо выполняться немедленно. Дисплейный файл обеспечивает удобство и эффективность упорядочения и обработки команд библиотеки. Рассмотрим, например, перемещение изображения дома по экрану. Предположим, что рисунок состоит из нескольких сотен отрезков. Если бы эти отрезки существовали по отдельности, нам пришлось бы написать команду перемещения несколько сотен раз – для каждого из них. Однако если команды построения отрезков,

образующих рисунок, объединены в дисплейный файл, команду перемещения достаточно написать только один раз. Чтобы поместить графические элементы в дисплейный файл, нужно открыть этот файл перед первой командой, которая должна в него попасть, и закрыть его после последней команды (см. пример ниже).

```
OpenGL
glNewList(AREA_FILL, GL_COMPILE_AND_EXECUTE);
/* Открытие дисплейного файла с именем AREA_FILL. */
      glBegin(GL_POLYGON);
         glVertex2fv(point1);
         glVertex2fv(point2);
         .
         .
         .
      glEnd();
      /* Определение многоугольника. */
glEndList();
      /* Закрытие дисплейного файла. */
```

Дисплейный файл OpenGL ориентирован на оптимизацию производительности, в частности, при работе по сети, но не за счет производительности на отдельном компьютере. Оптимизация обеспечивается благодаря тому, что дисплейный файл хранится в виде списка команд, а не в виде динамической базы данных. Другими словами, созданный дисплейный файл изменить уже нельзя. Если бы его можно было изменять, производительность упала бы из-за накладных расходов на поиск внутри списка и управление памятью. Изменение отдельных частей дисплейного файла потребовало бы перераспределения памяти, что могло бы привести к ее фрагментации. Дисплейные файлы, как правило, работают так же быстро, как и обычные последовательности команд, не объединенные в группы. В случае OpenGL дисплейные файлы могут значительно повысить производительность, в особенности при передаче подпрограмм OpenGL по сети, поскольку файлы эти хранятся на сервере, благодаря чему сокращается сетевой трафик.

К созданному дисплейному файлу могут быть применены следующие операции:

- множественное выполнение (multiple execution) – один и

тот же файл можно выполнять много раз;

- иерархическое выполнение (hierarchical execution) – иерархическим называется дисплейный файл (родительский), вызывающий другие дисплейные файлы (дочерние). Иерархические дисплейные файлы удобны для объектов, состоящих из отдельных компонентов, особенно если некоторые компоненты входят в объект в нескольких экземплярах;
- удаление (deletion) – дисплейный файл может быть удален.

3.7. Матрица преобразования

Как говорилось в разделе 3.2, проецирование точек на объект в трехмерном пространстве требует преобразования координат из одной системы в другую. Сначала нужно перевести координаты точек объекта из модельной системы в мировую. Текущее положение объекта обычно задается через повороты и смещения относительно исходного положения, в котором модельная система координат совпадала с мировой. Следовательно, мировые координаты точек объекта можно получить трансляцией и поворотом соответствующих точек из их исходного положения, в котором их модельные координаты совпадали с мировыми. Большинство графических библиотек выполняют эти преобразования самостоятельно, а программисту остается задать только смещение и поворот для интересующего его объекта. Однако проектировщику все равно нужно знать законы преобразований, чтобы рисовать объекты в нужных местах без проб и ошибок, в особенности, если эти объекты перемещаются достаточно сложным образом. В этом разделе мы рассмотрим матрицы преобразования, действующие на точки объекта при его перемещении и повороте.

Получив мировые координаты всех точек объекта в его текущем положении, мы должны вычислить координаты этих точек в наблюдательской системе. Перевод координат из одной системы в другую называется *отображением (mapping)*. Отображение между мировой и наблюдательской системами

координат обычно также осуществляется графической библиотекой самостоятельно, по заданным программистом координатам точки зрения, точки наблюдения и направлению вектора вертикали (в мировых координатах). Матрица преобразования для операции отображения рассматривается в разделе 3.7.3.

3.7.1. Трансляция

При трансляции объекта на величины a, b и c в направлениях x, y и z соответственно по отношению к начальному положению, в котором модельная система координат совпадала с мировой (рис. 3.16), мировые координаты точек объекта в новом положении (X_w, Y_w, Z_w) вычисляются следующим образом:

$$\begin{aligned} X_w &= X_m + a; \\ Y_w &= Y_m + b; \\ Z_w &= Z_m + c. \end{aligned} \tag{3.3}$$

В этой формуле числа X_m, Y_m, Z_m являются также модельными координатами точки.

Формула (3.2) может быть записана в матричной форме[1]:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix}$$

$$\Downarrow \tag{3.4}$$

$$Trans(a,b,c)$$

Легко убедиться, что формулы (3.4) и (3.3) эквивалентны друг другу: для этого достаточно записать (3.4) в развернутом виде. Операцию сложения в (3.3) удалось записать через умножение в

[1] Координаты могут быть также представлены в виде строки. Тогда матрица преобразования записывается после вектора координат. В этом случае матрица преобразования представляет собой транспонированную матрицу из формулы (3.4). В формуле (3.4) мы следуем соглашениям о записи OpenGL.

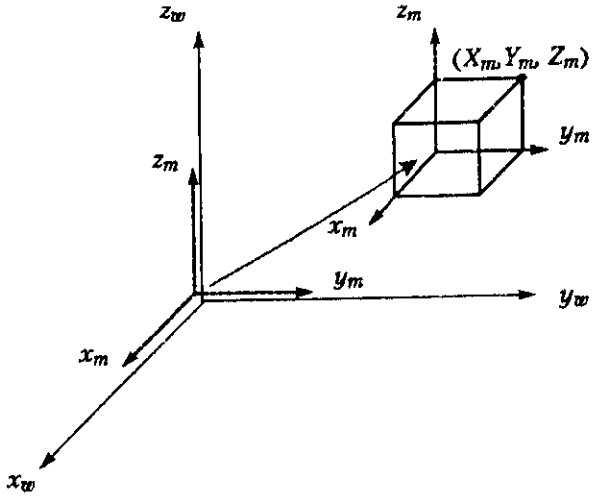


Рис. 3.16. Трансляция объекта

(3.4) благодаря использованию однородных координат, в которых трехмерный вектор записывается через четыре скаляра вместо трех[1]. Матрица, которую мы получили, называется *матрицей однородного преобразования* (*homogeneous transformation matrix*). В данном случае преобразование является трансляцией. Если бы преобразование (в частности, трансляцию) нужно было применить к точке в двумерном пространстве, однородная матрица преобразования редуцировалась бы до матрицы размерностью 3x3 удалением третьей строки и третьего столбца из матрицы размерностью 4x4. Новая матрица действовала бы на вектор координат размерностью 3x1, полученный из вектора 4x1 удалением 2-координаты.

[1] Любой вектор $(x, y, z)^T$ трехмерного пространства может быть записан в соответствующих однородных координатах в виде $(xw, yw, zw, w)^T$, где верхний индекс T обозначает операцию транспонирования. Поскольку значение w может быть произвольным, для каждого вектора существует множество вариантов записи в однородных координатах. В формуле (3.4) используется значение w 1.

3.7.2. Вращение

Пусть объект поворачивается на угол 0 вокруг оси x мировой системы координат вместе со своей модельной системой, которая, как и в предыдущем случае, изначально совпадает с мировой (рис. 3.17). Мировые координаты точки объекта в новом положении (Xw, Yw, Zw) могут быть получены из исходных мировых координат этой точки (X_m, Y_m, Z_m), совпадающих с ее текущими координатами в модельной системе.

Соотношение между (Xw, Yw, Zw) и (X_m, Y_m, Z_m) становится очевидным после проецирования рис. 3.17 на плоскость yz. Результат проецирования показан на рис. 3.18.

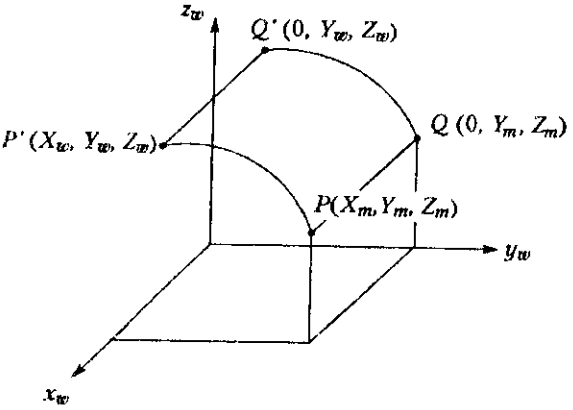


Рис. 3.17. Вращение вокруг оси x

Из рис. 3.18 можно легко получить следующие равенства:

$$X_w = X_m \tag{3.5}$$

$$\begin{aligned} Y_w &= l\cos(\theta+\alpha) = l(\cos\theta\cos\alpha - \sin\theta\sin\alpha) = \\ &= l\cos\alpha\cos\theta - l\sin\alpha\sin\theta = Y_m\cos\theta - Z_m\sin\theta \end{aligned} \tag{3.6}$$

$$\begin{aligned} Z_w &= l\sin(\theta+\alpha) = l(\sin\theta\cos\alpha + \cos\theta\sin\alpha) = \\ &= l\cos\alpha\sin\theta + l\sin\alpha\cos\theta = Y_m\sin\theta + Z_m\cos\theta \end{aligned} \tag{3.7.}$$

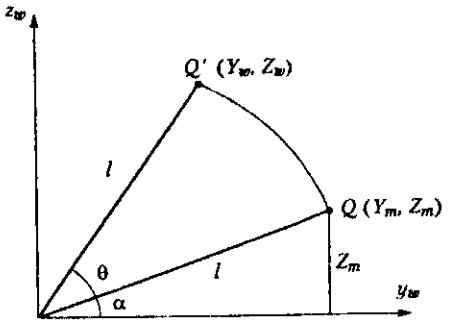


Рис. 3.18. Проекция на плоскость *yz*

Равенства (3.5), (3.6) и (3.7) могут быть записаны в матричной форме:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} \tag{3.8}$$

Матрица в правой части формулы (3.8) – это однородная матрица преобразования вращения вокруг оси *x*, которая кратко обозначается $Rot(x,\theta)$. Подобно матрице трансляции, для двумерного объекта однородная матрица вращения редуцируется до размера 3x3.

Однородные матрицы вращения вокруг осей *y* и *z* получаются аналогичным образом и записываются так:

$$Rot(y,\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

$$Rot(z,\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

Мы получили матрицы преобразования, описывающие поворот вокруг мировых осей координат. Можно показать, что поворот вокруг любой произвольной оси раскладывается на повороты вокруг осей x, y и z. Таким образом, матрица преобразования для произвольной оси получается перемножением матриц (3.8) – (3.10).

Как уже отмечалось, матрицы преобразования, описываемые в этом разделе, обычно вычисляются соответствующими подпрограммами графических библиотек. Приведенный ниже код иллюстрирует использование подпрограмм PHIGS и OpenGL.

```
PHIGS
ptranslate3(Pvector* offset3, Pint* error_ind, Pmatrix3 result3);
   /*  offset3: вектор трансляции
       error_ind: код ошибки
       result3: вычисленная матрица преобразования[1] */
protate_x(Pfloat angle, Pint* error_ind, Pmatrix3 result3);
protate_y(Pfloat angle, Pint* error_ind, Pmatrix3 result3);
protate_z(Pfloat angle, Pint* error_ind, Pmatrix3 result3);
   /*  angle: угол поворота
       error_ind: код ошибки
       result3: вычисленная матрица преобразования */

OpenGL
glTranslated(GLdouble offset_x, GLdouble offset_y, GLdouble offset_z);
       /* Умножает текущую матрицу на матрицу трансляции объекта со
       смещениями offset_x, offset_y и offset_z по соответствующим осям. */
glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z);
       /* Умножает текущую матрицу на матрицу поворота объекта на угол
       angle против часовой стрелки вокруг луча, проведенного из начала
       координат в точку (x, y, z). */
```

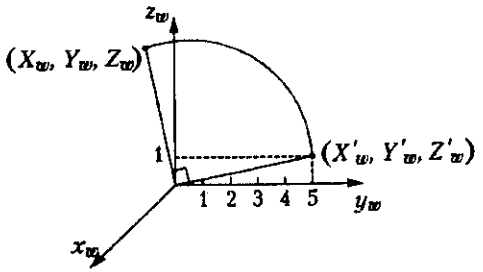
[1] В PHIGS матрица преобразования ставится после вектора – строки.

Рассмотрим несколько примеров, поясняющих применение матриц преобразований.

Пример 3.1

Объект в трехмерном пространстве транслируется на 5 единиц в направлении y мировой системы координат, после чего

поворачивается на 90° вокруг оси x той же системы координат. Если координаты точки объекта в модельной системе имеют значения (0, 0, 1), какими будут мировые координаты этой точки после трансляции и вращения?



Решение

Координаты (X'_w, Y'_w, Z'_w) после преобразования трансляции могут быть вычислены следующим образом:

$$\begin{bmatrix} X'_w & Y'_w & Z'_w \end{bmatrix}^T = Trans(0,5,0)\begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & 5 & 1 & 1 \end{bmatrix}^T.$$

После этого применяется преобразование вращения:

$$\begin{bmatrix} X_w & Y_w & Z_w & 1 \end{bmatrix}^T = Rot(x,90°)\begin{bmatrix} 0 & 5 & 1 & 1 \end{bmatrix}^T$$

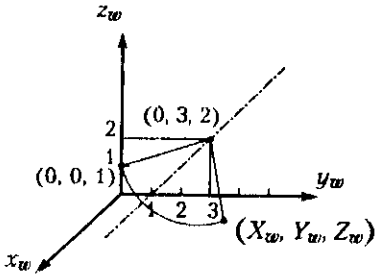
Следовательно, координаты точки после преобразований будут иметь значения (0,-1, 5). Обратите внимание, что предыдущие выражения можно объединить:

$$\begin{bmatrix} X_w & Y_w & Z_w & 1 \end{bmatrix}^T = Rot(x,90°)Trans(0,5,0)\begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}^T$$

Последнее выражение значительно более удобно, особенно при вычислении координат множества точек. В последнем случае матрицы трансляции и вращения перемножаются заранее и дают эквивалентную матрицу преобразования, которая действует на все интересующие нас точки. Процесс вычисления эквивалентной матрицы преобразования путем перемножения отдельных матриц преобразования в соответствующей последовательности называется *конкатенацией* (*concatenation*). Возможность выполнения конкатенации – одно из преимуществ использования однородной системы координат, в которой трансляция записывается через матричное умножение, а не через сложение.

Пример 3.2

Объект в пространстве поворачивается на 90° вокруг оси, параллельной оси x мировой системы координат и проходящей через точку с мировыми координатами (0, 3, 2). Если точка объекта имеет модельные координаты (0, 0, 1), какими будут мировые координаты той же точки после поворота?



Решение

Мы изучили только повороты относительно осей, проходящих через начало координат, поэтому нам придется сместить объект вместе с осью вращения. Ось вращения должна проходить через начало координат, причем положение объекта относительно этой оси должно сохраниться. Трансляция объекта вместе с осью на вектор (0, -3, -2) даст нам совпадение оси вращения с осью x мировых координат. Затем мы повернем объект вокруг оси x на 90°, после чего сместим его обратно на вектор (0, 3, 2), чтобы вернуться к исходному положению.

Эти операции могут быть записаны следующим образом:

$$[X_w\ Y_w\ Z_w\ 1]^T = Trans(0,3,2)Rot(x,90°)Trans(0,-3,-2)[0\ 0\ 1\ 1]^T$$

Обратите внимание на последовательность матриц в этой формуле. Результат легко проверить, применяя преобразования одно за другим, как в примере 3.1.

Раскрывая выражение, получим:

$$[X_w\ Y_w\ Z_w\ 1]^T = \begin{bmatrix}1&0&0&0\\0&1&0&3\\0&0&1&2\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&\cos 90°&-\sin 90°&0\\0&\sin 90°&\cos 90°&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&-3\\0&0&1&-2\\0&0&0&1\end{bmatrix} = [0\ \ 4\ \ -1\ \ 1]^T$$

Результат соответствует приведенной выше иллюстрации.

3.7.3. Отображение

Отображение (*mapping*) состоит в вычислении координат точки в некоторой системе координат по известным координатам той же точки в другой системе координат.

Рассмотрим две системы координат (рис. 3.19). Предположим, что координаты (X_2, Y_2, Z_2) точки P в системе координат x_2, y_2, z_2 должны быть вычислены по координатам (X_1, Y_1, Z_1) той же точки в системе x_1, y_1, z_1. Далее, предположим, что вычисление производится применением матрицы преобразования $T_{1\text{-}2}$ к известным координатам:

$$\begin{bmatrix} X_2 & Y_2 & Z_2 & 1 \end{bmatrix}^{\mathrm{T}} = T_{1\text{-}2} \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \end{bmatrix}^{\mathrm{T}}. \qquad (3.11)$$

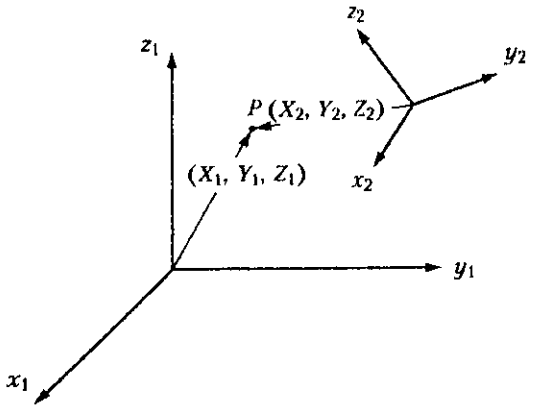


Рис. 3.19. Отображение из одной системы координат в другую

Записав матрицу $T_{1\text{-}2}$ в явном виде, мы получим из формулы (3.11) следующее выражение:

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix}. \qquad (3.12)$$

Чтобы найти неизвестные в уравнении (3.12), подставим в него конкретные значения X_1=0, Y_1=0 и Z_1=0, в результате чего получим:

$$X_2 = p_x,\ Y_2 = p_y,\ Z_2 = p_z. \qquad (3.13)$$

Можно сказать, что p_x, p_y и p_z определяют координаты начала отсчета системы x_1, y_1, z_1 в системе координат x_2, y_2, z_2.

Теперь подставим в уравнение (3.12) значения X_1=1, Y_1=0, Z_1=0 и получим:

$$X_2 = n_x + p_x,\ Y_2 = n_y + p_y,\ Z_2 = n_z + p_z. \qquad (3.14)$$

Вычитая формулы (3.13) из (3.14), можно заключить, что n_x, n_y, n_z – компоненты x_2, y_2, z_2 единичного вектора, направленного вдоль оси x_1 системы координат x_1, y_1, z_1. Следовательно, коэффициенты n_x, n_y, n_z легко вычислить с учетом взаимной ориентации систем координат.

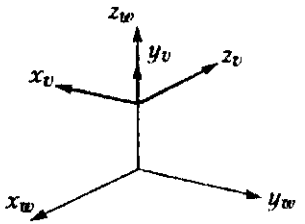
Аналогичным образом, o_x, o_y и o_z представляют собой компоненты x_2, y_2, z_2 единичного вектора оси y_1 а компоненты a_x, a_y и a_z – вектора оси z_1.

Следующий пример демонстрирует предлагаемую теорию в действии.

Пример 3.3

По заданному положению точки зрения (–10, 0, 1), точки наблюдения (0, 0, 1) и вектора вертикали (0, 0, 1) строится наблюдательская система координат (см. представленный ниже рисунок). Обратите внимание, что все координаты и компоненты векторов даны в мировой системе координат. Зная относительное положение системы координат просмотра и мировой системы координат, рассчитать:

- матрицу преобразования $T_{w\text{-}v}$;
- координаты точки с мировыми координатами (5, 0, 1) в наблюдательской системе координат.



Решение

Первые три числа первого столбца $T_{w\text{-}v}$ (то есть n_x, n_y и n_z) равны (0 0 –1), поскольку они представляют собой компоненты x_v, y_v и z_v оси x_v. Аналогично, o_x, o_y и o_z, которые представляют собой компоненты x_v, y_v и z_v оси y_v равны (–1 0 0), а a_x, a_y и a_z равны (0 1 0). p_x, p_y и p_z – это координаты x_w, y_w и z_w начала координат системы x_w, y_w и z_w, поэтому они равны 0, -1 и 0 соответственно. В итоге получаем матрицу $T_{w\text{-}v}$.

$$T_{w\text{-}v} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Координаты точки (5, 0, 1) в наблюдательской системе мы получим, применив к этому вектору координат только что вычисленную матрицу отображения $T_{w\text{-}v}$.

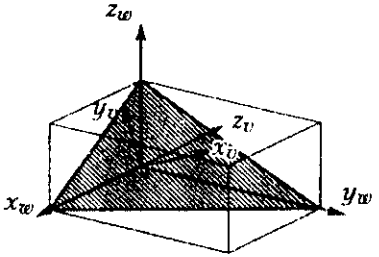
$$\begin{bmatrix} X_v \\ Y_v \\ Z_v \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -5 \\ 1 \end{bmatrix}$$

Итак, координаты точки (5, 0, 1) в наблюдательской системе имеют значения (0, 0, –5), о чем можно догадаться, посмотрев на приведенный рисунок.

Пример 3.4

Точки зрения и наблюдения имеют координаты (5, 5, 5) и (0, 0, 0) соответственно, а вектор вертикали выбирается равным (0, 0, 1). Проекция изометрическая. Необходимо вычислить матрицу

преобразования отображения $T_{w\text{-}v}$ и наблюдательские координаты точки с мировыми координатами (0, 0, 5).



Решение

Наблюдательская система координат может быть изображена так, как показано на рисунке выше. Заштрихованный треугольник на этом рисунке параллелен плоскости экрана. В этой плоскости лежат оси x_v и y_v.

Чтобы вычислить элементы $T_{w\text{-}v}$ нам нужно получить компоненты x_v, y_v и z_v осей x_w, y_w и z_w. Для этого обозначим единичные векторы осей x_v, y_v и z_v буквами i_v, j_v и k_v соответственно. Единичные векторы осей x_w, y_w и z_w будут называться просто i, j и k.

Единичный вектор k_v направлен из точки наблюдения в точку зрения, поэтому

$$k_v = \frac{1}{\sqrt{3}}i + \frac{1}{\sqrt{3}}j + \frac{1}{\sqrt{3}}k.$$

Как отмечалось в разделе 3.2, единичный вектор j_v должен быть коллинеарен проекции вектора вертикали на экран. Другими словами, его направление будет совпадать с направлением вектора, полученного вычитанием из вектора вертикали составляющей, перпендикулярной экрану. Обозначив вектор вертикали u_p, запишем выражение для j_v.

$$j_v = \frac{u_p - (u_p \cdot k_v)k_v}{\left|u_p - (u_p \cdot k_v)k_v\right|} = \frac{-\frac{1}{3}i - \frac{1}{3}j + \frac{1}{3}k}{\left|-\frac{1}{3}i - \frac{1}{3}j + \frac{1}{3}k\right|} = -\frac{1}{\sqrt{6}}i - \frac{1}{\sqrt{6}}j + \frac{2}{\sqrt{6}}k.$$

Последний единичный вектор i_v вычисляется через векторное произведение:

$$i_v = j_v \times k_v = \frac{1}{-\sqrt{2}} i + \frac{1}{\sqrt{2}} j.$$

Теперь вычислим n_x -- компоненту x_v оси x_w -- как произведение:

$$i \cdot i_v = \frac{1}{-\sqrt{2}}.$$

Аналогичным образом, компонента n_y равна

$$i \cdot j_v = \frac{1}{-\sqrt{6}}.$$

а компонента n_z равна

$$i \cdot k_v = \frac{1}{\sqrt{3}}.$$

Таким же путем получим второй и третий столбцы матрицы T_{w-v}. Компоненты p_x, p_y и p_z можно не вычислять, потому что в этом примере начала координат наблюдательской и мировой систем совпадают. Следовательно, матрица преобразования выглядит так:

$$T_{w-v} = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Координаты точки (0, 0, 5) в наблюдательской системе имеют значение

$$[X_v \;\; Y_v \;\; Z_v \;\; 1]^T = T_{w-v} \cdot [0 \;\; 0 \;\; 5 \;\; 1]^T = \left[0 \;\; \frac{5\sqrt{6}}{3} \;\; \frac{5\sqrt{6}}{3} \;\; 1\right]^T.$$

Экранные координаты точки в изометрической проекции получаются из наблюдательских координат непосредственно:

$$\left(0 \quad \frac{5\sqrt{6}}{3}\right).$$

Изометрическая проекция относится к параллельным, поэтому все точки на оси z_w проецируются на ось y экранных координат. Фактически задание вектора вертикали, равного (0, 0, 1), означает, что после проецирования ось z_w представляется на экране в виде вертикальной прямой.

3.7.4. Другие матрицы преобразования

Помимо матриц преобразования, рассмотренных в предыдущих разделах, часто используются матрицы масштабирования и зеркального отображения.

Для масштабирования объекта с коэффициентом s_x по оси x, s_y по оси y, s_z по оси z применяется следующая матрица преобразования:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \qquad (3.15)$$

Для двумерных объектов матрица масштабирования редуцируется до размера 3x3, как это было с матрицами трансляции и поворота. Эффекта масштабирования можно достичь, изменив размеры видового экрана или окна, не меняя значений координат.

Матрица преобразования (3.15) используется при масштабировании объекта относительно начала координат. Часто бывает необходимо масштабировать объект относительно одной из его точек P с координатами (X_p, Y_p, Z_p). В этом случае сначала к точке P применяется преобразование трансляции *Trans* ($-X_p$, $-Y_p$, $-Z_p$), которое перемещает эту точку в начало координат, затем применяется матрица масштабирования из (3.15), после чего объект возвращается в исходное положение действием *Trans* (X_p, Y_p, Z_p).

Отражение относительно зеркальной плоскости xy может быть достигнуто при помощи приведенной ниже матрицы

преобразования. Преобразование заключается в изменении знака координаты z.

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \qquad (3.16)$$

Матрицы преобразования для других отражений (относительно плоскостей xz и yz) выводятся аналогичным образом.

3.8. Удаление невидимых линий и поверхностей

Проекция на экран станет более наглядной, если будет содержать только видимые линии и поверхности. *Удаление невидимых линий* (*hidden-line removal*) заключается в блокировании отображения отрезков, скрытых от наблюдателя, а *удаление невидимых поверхностей* (*hidden-surface removal*) есть то же самое по отношению к поверхностям. Удаление невидимых линий иллюстрируют рис. 3.20 и рис. 3.21. Очевидно, что эта процедура значительно облегчает восприятие объекта.

Опубликовано множество программных алгоритмов удаления невидимых линий и поверхностей. Их авторы пытаются повысить вычислительную эффективность и расширить диапазон объектов для применения своих алгоритмов. Однако лучше всего удаление реализуется посредством графического устройства, которое называется z-буфером, поэтому в настоящее время исследования на эту тему практически не ведутся. В настоящем разделе мы рассмотрим несколько типичных алгоритмов удаления скрытых линий и поверхностей программным путем, а также изучим метод использования для той же цели z-буфера.

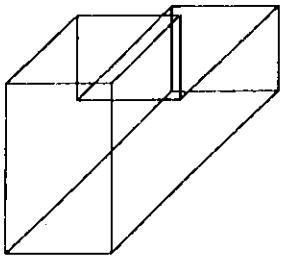


Рис. 3.20. Изображение до удаления невидимых линий

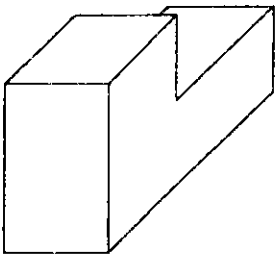


Рис. 3.21. Изображение после удаления невидимых линий

3.8.1. Алгоритм удаления невидимых граней

Алгоритм удаления невидимых граней (*back-face removal algorithm*) основан на том, что грань объекта может быть видимой только в том случае, если вектор внешней нормали к этой грани направлен в сторону наблюдателя. В противном случае грань будет невидима. Например, верхняя грань бруска, изображенного на рис. 3.22, считается видимой, если вектор внешней нормали **N** имеет положительную составляющую в направлении вектора **M**, проведенного из точки на грани к наблюдателю. Математически это записывается так:

- если $\mathbf{M} \cdot \mathbf{N} > \mathbf{0}$, поверхность видима;
- если $\mathbf{M} \cdot \mathbf{N} = \mathbf{0}$, поверхность проецируется в отрезок;
- если $\mathbf{M} \cdot \mathbf{N} < \mathbf{0}$, поверхность невидима.

Этот алгоритм легко применить к объекту, ограниченному плоскими поверхностями, поскольку вектор нормали **N** постоянен

в пределах поверхности. Однако к вогнутому[1] объекту алгоритм неприменим, поскольку грань, направленная к наблюдателю, может быть закрыта другой гранью того же объекта (рис. 3.23). Та же проблема возникает в случае нескольких выпуклых объектов, которые могут закрывать грани друг друга. Следовательно, алгоритм удаления невидимых граней применим только к одному выпуклому объекту. Более того, алгоритм неприменим к объектам, для которых вектор внешней нормали определяется неоднозначно[2] (см., например, рис. 3.24).

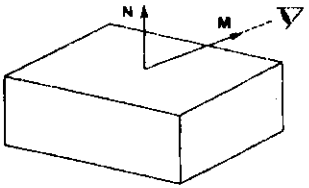


Рис. 3.22. Векторы, определяющие видимость грани

Если поверхности объекта не являются плоскими, значение **N** будет меняться в пределах одной грани в зависимости от выбранной точки. Одновременно может меняться и знак произведения **M** · **N**. Это означает, что у одной и той же грани будут как видимые, так и невидимые участки. Поэтому грань должна быть разделена на две части вдоль кривой, на которой выполняется равенство **M** · **N** = **0**. Эта кривая называется *силуэтной линией* (*silhouette line*). После разделения грани вдоль силуэтной линии знак **M** · **N** будет постоянным на каждой из частей грани. Процедура может показаться легкой, но рассчитать силуэтную линию очень сложно, а из-за этого теряется главное преимущество алгоритма удаления невидимых граней – простота реализации.

[1] Объект называется вогнутым, если, по крайней мере, две его грани смыкаются под внутренним углом, большим 180°. Если все грани смыкаются с внутренними углами, меньшими 180°, объект называется выпуклым.

[2] Такой объект вообще представляет собой достаточно сложную проблему.

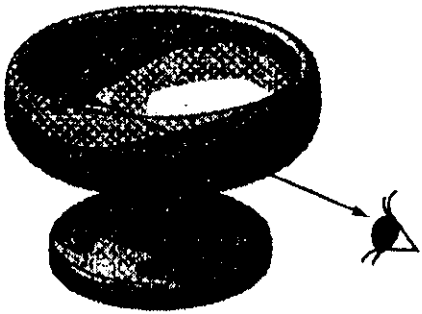


Рис. 3.23. Пример вогнутого объекта

После того как все грани классифицируются как видимые или невидимые, на экран выводятся ребра видимых граней, в результате чего получается рисунок без невидимых линий. Если же нужно получить рисунок без невидимых поверхностей, видимые поверхности заливаются выбранными цветами.

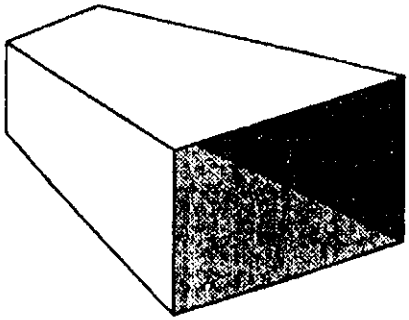


Рис. 3.24. Объект с неоднозначно определенным вектором внешней нормали

3.8.2. Алгоритм удаления невидимых линий

Алгоритм сортировки по глубине используется для удаления невидимых поверхностей. Алгоритм невидимых граней позволяет построить рисунок со скрытыми линиями, но имеет множество ограничений в общем случае. При применении

алгоритма невидимых граней к множеству объектов удалено будет лишь около 50% невидимых линий. Нам нужен алгоритм, который удалял бы все невидимые линии независимо от количества объектов, их выпуклости и наличия криволинейных поверхностей.

Один из таких алгоритмов действует следующим образом. Для каждого ребра[1] каждого объекта производится проверка, не закрыто ли оно гранями[2] каких-либо объектов. Закрытые части ребер последовательно исключаются до тех пор, пока не останется непроверенных поверхностей. Оставшиеся части всех ребер выводятся на экран.

Реализация алгоритма включает несколько этапов.

1. Поверхности, направленные к наблюдателю, выделяются из всех остальных в отдельную группу при помощи алгоритма невидимых граней. Выделенные поверхности сохраняются в массив FACE-TABLE. Грани, направленные от наблюдателя, учитывать не требуется, поскольку они сами по себе скрыты, а потому не скрывают ребра других граней. Для каждой грани сохраняется максимальное и минимальное значение Z_v. Криволинейные поверхности разделяются по силуэтным линиям (как в алгоритме невидимых граней), а видимые части этих поверхностей также сохраняются в массиве FACE-TABLE вместе с плоскими гранями.
2. Ребра граней из массива FACE-TABLE выделяются из всех прочих ребер и собираются в отдельный список. Ребра других граней, не входящих в FACE-TABLE, можно не рассматривать, поскольку они невидимы. Затем для каждого ребра из списка производится проверка, не закрывается ли это ребро гранью из FACE-TABLE.

[1] Ребром объекта называется кривая пересечения соседних поверхностей, ограничивающих внутренний объем объекта.

[2] Гранями называются поверхности, ограничивающие объем объекта. Площадь любой грани конечна, потому что все грани ограничиваются ребрами.

3. Скрытие ребра гранью можно обнаружить, сравнивая диапазоны значений Z_v ребра и грани. Возможны три случая (рис. 3.25). В случае рис. 3.25, *а* все значения Z_v ребра меньше минимального значения Z_v грани, то есть грань находится перед ребром. В случае рис. 3.25, *б* значения Z_v ребра больше максимального значения Z_v грани, то есть грань находится за ребром. В случае рис. 3.25, *в* диапазоны значений Z_v грани и ребра перекрываются, то есть часть ребра находится за гранью, а другая часть – перед ней. Если ребро находится перед проверяемой гранью, из массива FACE-TABLE выбирается следующая грань и ребро сравнивается уже с ней. Если ребро оказывается за гранью, или проходит ее насквозь, приходится выполнять дополнительное действие.
4. Ребро и грань проецируются на экран, после чего производится проверка перекрытия проекций. Если перекрытия нет, из этого следует, что ребро не закрывает проверяемую грань. Из массива FACE-TABLE выбирается следующая грань и проверяется согласно пункту 3. Если проекции перекрываются, ребро делится на две части в той точке, где она проходит сквозь проверяемую грань (рис. 3.26). Закрытая часть ребра отбрасывается, а видимые части добавляются в список. Затем пункт 3 повторяется для новых элементов списка. Исходное ребро из списка удаляется.
5. Ребра, прошедшие проверку со всеми гранями из FACE-TABLE, считаются видимыми и выводятся на экран.

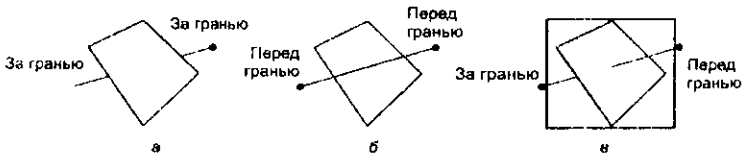


Рис. 3.25. Три возможных положения грани и ребра

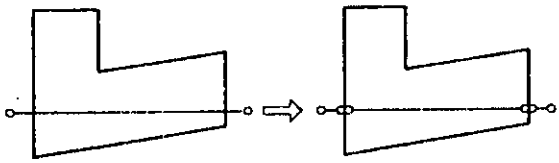


Рис. 3.26. Разбиение ребра

3.8.3. Метод z-буфера

Метод *z*-буфера основан на том же принципе, что и алгоритм сортировки по глубине: на любом участке экрана оказывается проекция элемента, расположенного ближе всех прочих к наблюдателю. Здесь под элементами понимаются точки, кривые или поверхности. Данный метод требует использования области памяти, называемой *z*-буфером. В этом буфере для каждого пиксела хранится значение координаты Z_v того элемента, проекция которого изображается данным пикселом. Как уже говорилось, значение Z_v (то есть координата z в наблюдательской системе) есть мера удаленности объекта от наблюдателя. Объем *z*-буфера определяется количеством пикселов, для каждого из которых требуется сохранить вещественное число.

Грани, векторы нормали которых направлены от наблюдателя, невидимы для него, поэтому на экран проецируются только те грани, векторы нормали которых направлены к наблюдателю. Однако в отличие от метода сортировки по глубине, в данном случае порядок проецирования значения не имеет. Причина станет очевидной, как только вы прочитаете приведенное ниже описание алгоритма.

Сначала проецируется произвольно выбранная поверхность, и в ячейки памяти *z*-буфера, соответствующие пикселам проекции поверхности, записываются значения координат Z_v точек поверхности, являющихся прообразами данных пикселов. Пикселы окрашиваются в цвет первой поверхности. Затем проецируется следующая поверхность, и все неокрашенные пикселы, относящиеся к ее проекции, окрашиваются в цвет второй

поверхности. Если пикселы уже были окрашены, соответствующие им значения Z_v сравниваются со значениями Z_v точек текущей поверхности. Если сохраненное значение Z_v какого-то пиксела оказывается больше текущего (то есть точка на предыдущей поверхности ближе к наблюдателю, чем точка на текущей поверхности), цвет пиксела не меняется. В противном случае пиксел окрашивается в цвет текущей поверхности. Изначально z-буфер инициализируется координатами Z_v соответствующими дальней плоскости (см. рис. 3.11), поэтому пикселы проекции первой поверхности автоматически окрашиваются в ее цвет. Повторяя эту процедуру для всех имеющихся плоскостей, мы окрасим все пикселы экрана в цвета ближайших поверхностей (рис. 3.27).

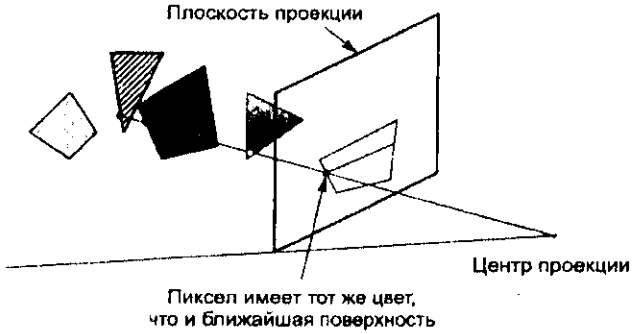


Рис. 3.27. Основы метода z-буфера

Как следует из предшествующего описания, метод z-буфера предназначен главным образом для удаления скрытых поверхностей, как и метод сортировки по глубине. Однако метод z-буфера с небольшими изменениями позволяет реализовать и построение рисунков со скрытыми линиями. Сначала все поверхности проецируются на экран, причем пикселы окрашиваются в цвет фона (на экране ничего не появляется). При этом z-буфер заполняется правильными значениями Z_v. Затем на экран проецируются ребра поверхностей. При этом значения Z_v

ребер сравниваются со значениями Z_v ближайших к наблюдателю поверхностей, уже найденными на предыдущем этапе. Окрашиваются только те пикселы, для которых новые значения Z_v больше исходных. Таким образом, участки ребер, скрытые поверхностями, на экране не появляются. Эта процедура дает правильный рисунок без скрытых линий, но некоторые граничные линии могут оказаться слишком тонкими, потому что некоторые пикселы этих линий будут относиться к поверхностям, ограниченным соответствующими ребрами. Эту проблему легко решить, пододвинув весь объект поближе в момент проецирования ребер.

3.9. Графический интерфейс пользователя

Как отмечалось ранее, важной особенностью существующих программ CAD/ CAM/CAE является взаимодействие с пользователем посредством графического ввода и графического вывода. Другими словами, программное обеспечение должно иметь возможность открывать окна (области взаимодействия) для отображения меню или значков, а также сопоставлять пунктам меню и значкам какие-либо функции. Программное обеспечение, предоставляющее такие возможности, называется *графическим интерфейсом пользователя* (*graphical user interface* – *GUI*). Программист может построить самодельный графический интерфейс на базе конкретной графической библиотеки. Этот интерфейс будет обладать недостатком, связанным с невозможностью перенести его на рабочие станции, не поддерживающие использованную графическую библиотеку. Поэтому интерфейс придется переписывать заново для каждой новой графической рабочей станции.

Чтобы избежать этой проблемы, программисты строят графический интерфейс на базе системы X window, которая в настоящее время поддерживается большинством графических рабочих станций. (Системе X window посвящен следующий раздел этой главы.) Два типичных интерфейса пользователя, основанных на X window, называются Open Look и OSF/Motif. Open Look поддерживался корпорацией Sun Microsystems, а OSF/Motif – всеми

прочими производителями, включая IBM, Hewlett-Packard, DEC и Tektronix.

Программист, пишущий приложение с использованием Open Look или OSF/ Motif, может обращаться к функциям диспетчера окон для выполнения таких операций, как открытие окон, построение меню и выполнение задач, вызываемых пунктами меню[1]. Графический экран диспетчера окон Motif показан на рис. 3.28.

Open Look и Motif хороши для разработки приложений по той причине, что они основаны на системе X window, достоинства которой будут рассмотрены в следующем разделе.

3.10. Система X window

Разработка системы X window (или просто X) началась в 1983 г. в Массачусетском Технологическом институте под кодовым названием «проект Афина». Оконный интерфейс разрабатывался на основе операционной системы под названием W, которая была создана в Стенфордском университете в начале 80-х. В 1986 г. вышла первая коммерческая версия системы X window, которая называлась X10. Позднее было объявлено о выпуске X11R5.

Система X window позволяет приложению открывать и закрывать окна на рабочих станциях, подключенных к сети. Операции ввода и вывода также могут осуществляться на любой рабочей станции. Слово «окно» имеет в этом разделе несколько иное значение, нежели в разделе 3.3. Здесь под окном понимается отдельная область экрана рабочей станции, через которую пользователь взаимодействует с вычислительными ресурсами сети. Например, мы можем открыть два окна на рабочей станции INDIG02 (рис. 3.29) и использовать одно из этих окон в качестве портов ввода и вывода для программы, выполняемой на компьютере SUN. Одновременно второе окно может использоваться для ввода и вывода программы, выполняемой на самом INDIG02.

[1] На персональных компьютерах, работающих под управлением Windows 98/NT, аналогичные возможности предоставляются набором базовых классов Microsoft (Microsoft Foundation Classes - MFC)

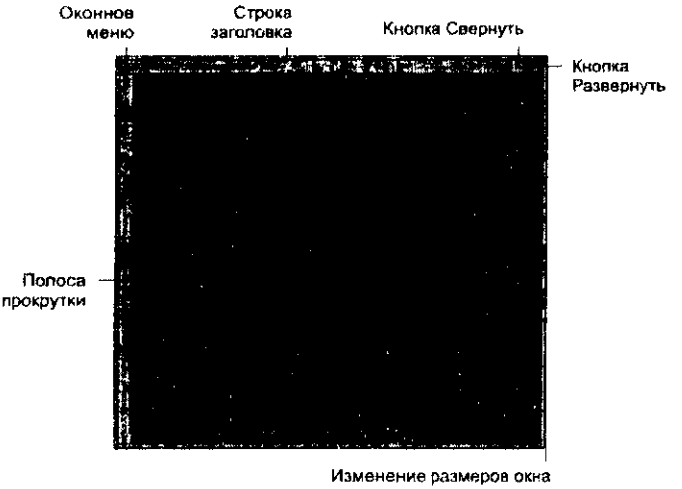


Рис. 3.28. Окно диспетчера Motif

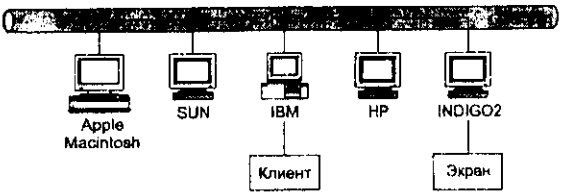


Рис. 3.29. Использование X в сетевой среде

Для выполнения задач такого рода система X window должна уметь принимать запросы от клиентов, посылать запросы на другие рабочие станции и выполнять графические операции ввода-вывода с окнами экрана. Окно может быть расположено на любом компьютере сети. На рабочей станции, где открыто окно, должен работать сервер X window, способный выполнять операции с графикой. Запросы клиента пишутся с использованием специальных функций, которые хранятся в библиотеке Xlib. Библиотека Xlib должна быть установлена и на той рабочей станции, где выполняется приложение. Пересылка запроса по сети

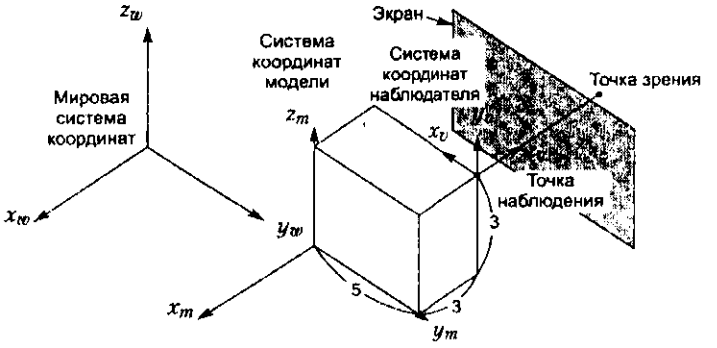
осуществляется основным кодом системы X window. Вообще говоря, в систему входят все перечисленные компоненты, включая X-сервер и Xlib.

Итак, приложение для X window обладает следующими достоинствами. Во-первых, приложение, выполняемое на одной рабочей станции, может осуществлять графический ввод-вывод через окно, открытое на другой рабочей станции. Во-вторых, на одной рабочей станции может быть открыто множество окон, через которые пользователь имеет возможность работать с вычислительными ресурсами разных машин. Наконец, приложение, написанное для системы X window, не зависит от операционной системы и типа рабочей станции. Теми же достоинствами обладает и графический интерфейс пользователя, который также может считаться приложением. То же можно сказать и об интерфейсах Open Look и Motif.

Вопросы и задачи

1. Какими недостатками обладают графические программы, написанные непосредственно с использованием команд драйвера устройства?
2. Объясните, почему графическая программа, основанная на какой-либо графической библиотеке, может выполняться лишь на ограниченном числе графических устройств.
3. Какова главная причина, по которой для задания положения на графическом устройстве используется виртуальная, а не обычная система координат устройства?
4. Почему для задания формы объекта используется модельная система координат этого объекта?
5. Объясните, каким образом задаются положение и ориентация каждого объекта сцены.
6. Кратко опишите процедуру преобразования координат точки объекта из модельной системы в экранную.
7. Объясните значение термина «окно» в компьютерной графике.

8. Объясните значение термина «окно просмотра».
9. Объясните различие между режимами выбора и опроса.
10. Какие операции могут выполняться с экранным файлом?
11. Точка зрения и точка наблюдения имеют мировые координаты (1, 1, 2) и (0, 1, 2). Вектор вертикали имеет координаты (0, 0, 1).
    1) Нарисуйте эскиз, показывающий взаимное расположение экрана, наблюдательской системы координат, точки зрения и точки наблюдения.
    2) Напишите матрицу преобразования $T_{w\text{-}v}$, осуществляющую перевод мировых координат в наблюдательские.
    3) С помощью матрицы преобразования определите наблюдательские координаты точки с мировыми координатами (5, 1, 2).
12. Какие матрицы преобразования и в каком порядке должны быть применены для поворота точки (2, 2) на плоскости xy на 30° против часовой стрелки относительно точки (3, 4)? Вычислите координаты точки после поворота, применив матрицы преобразования к точке (2, 2).
13. Взаимное положение систем координат показано на рисунке. Ответьте на приведенные ниже вопросы.



1) Точка объекта имеет координаты (-3, 0, 3) в модельной системе. Координаты этой точки в мировой системе координат (X_w, Y_w, Z_w) могут быть получены по приведенной ниже формуле, если имеется матрица преобразования T_m.

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = T_m \begin{bmatrix} -3 \\ 0 \\ 3 \\ 1 \end{bmatrix}.$$

Вычислите матрицу преобразования T_m и определите мировые координаты точки по приведенной формуле. В этой задаче модельная система координат совпадает с мировой, если мировая система координат транслируется на вектор (0, 2, -1).

2) Наблюдательская система координат на рисунке характеризуется координатами точки зрения (-10, 7, 2), точки наблюдения (-3, 7, 2) и вектора вертикали (0, 0, 1), которые заданы в мировой системе координат. Вычислите матрицу преобразования T_v, переводящую мировые координаты точки (X_w, Y_w, Z_w) в наблюдательские (X_v, Y_v, Z_v).

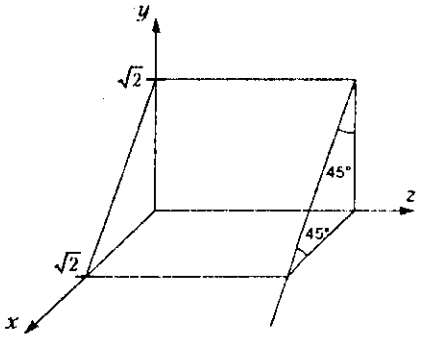
$$\begin{bmatrix} X_v \\ Y_v \\ Z_v \\ 1 \end{bmatrix} = T_v \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

3) Вычислите наблюдательские координаты точки с модельными координатами (-3, 0, 3), применив полученные ранее матрицы T_m и T_v.

14. Вычислите матрицу преобразования $T_{w\text{-}v}$ выполняющую переход от мировых координат к наблюдательским при условии, что координаты точки зрения и точки наблюдения равны (4, 5, 6) и (0, 0, 0) соответственно (в мировой системе). Ось 2 мировой системы координат

совпадает с вектором вертикали.

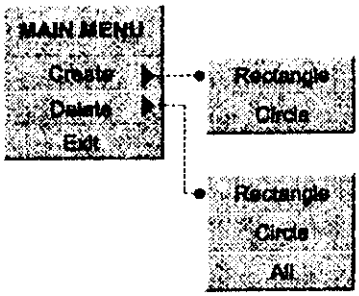
15. Представьте треугольную грань выпуклого объекта, вершины которой имеют координаты A(0, 0, 0), B(1, 1, 0) и C(0, 1, 2). Определите, видима ли эта грань из точки зрения V(0, 1, 5), используя алгоритм удаления невидимых граней. Предположите, что точка D(2, 2, 2) является одной из вершин данного объекта.
16. Точки A и B двумерного объекта перемещаются в точки C и D, что приводит к преобразованию исходной формы. Перечислите все матрицы преобразования, которые должны быть применены ко всем точкам тела, в правильном порядке. Координаты точек имеют следующие значения: A(2, 2), Б(5, 5), C(5, 2), D(7, 2 + $\sqrt{3}$).
17. Плоскость, перпендикулярная плоскости *xz*, расположена так, как показано на приведенном ниже рисунке.



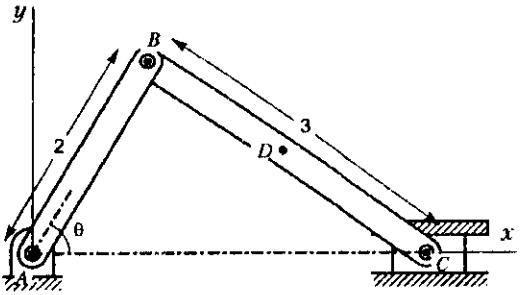
18. Зеркальное отражение P^* точки P относительно этой плоскости может быть вычислено по приведенной ниже формуле. Выразить $T_{p\text{-}p^*}$ через матрицы элементарных преобразований, к которым относятся *Trans* (*a*, *b*, *c*), *Rot* (*x*, *a*), *Rot* (*y*, *β*), *Rot* (*z*, *y*), а также матрицы отражения относительно плоскостей *xz*, *yz* и *xz*.

$$\begin{bmatrix} X^* & Y^* & Z^* & 1 \end{bmatrix} = T_{p\text{-}p^*} \cdot \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}.$$

19. Объясните отличия методов затушевывания Фонга и Гуро.
20. Используя любую доступную графическую библиотеку, напишите графическую программу, которая будет выполнять следующие действия.
    1) Отобразите систему координат и куб в ее центре. Положение точек зрения и наблюдения, направление вектора вертикали и размер куба задайте самостоятельно.
    2) Куб транслируется на +5 единиц в направлении *y* при нажатии левой кнопки мыши и на +5 единиц в направлении *z* при нажатии правой кнопки мыши. При нажатии средней кнопки куб возвращается в исходное положение (в центре системы координат).
21. Напишите двумерный графический редактор, работающий с приведенным ниже всплывающим меню.



22. Напишите графическую программу, которая будет рисовать траекторию точки D (средней точки шатуна BC) при вращении кривошипно-шатунного механизма под воздействием ведущего шатуна AB (см. приведенный ниже рисунок). По построенной траектории определите угол ведущего шатуна θ, при котором касательная к траектории становится горизонтальной.



Опишите преимущества, которые дает написание графического интерфейса приложения в системе X window.

ГЛАВА 4. СИСТЕМЫ АВТОМАТИЗИРОВАННОЙ РАЗРАБОТКИ ЧЕРТЕЖЕЙ

Как говорилось в предыдущих главах, *системы автоматизированной разработки чертежей* (*computer-aided drafting system*) – это программный продукт, позволяющий разработчику в интерактивном режиме создавать и изменять машиностроительные, архитектурные, инженерные чертежи, электрические схемы и чертежи множества других разновидностей. Эта программа, кроме того, обновляет базу данных, сохраняя готовые чертежи и их изменения. Таким образом, работа с системой автоматизированной разработки чертежей аналогична использованию текстового процессора. Единственное отличие в том, что на выходе пользователь получает чертеж, а не текстовый документ. Как в текстовом процессоре можно очень быстро подготовить новый документ на базе существующего, так и в системе автоматизированной разработки чертежей можно получить новый чертеж, изменив имеющиеся. Преимущества текстового процессора или автоматизированной системы разработки чертежей трудно оценить при подготовке абсолютно нового документа или чертежа. Но при изменении существующих документов и чертежей их преимущества становятся очевидными и неоценимыми.

В последующих разделах будут кратко рассмотрены наиболее типичные функции, имеющиеся в большинстве систем автоматизированной разработки чертежей. Конкретные команды вызова функции в каждой системе могут быть свои, так что при необходимости вам следует обратиться к руководству пользователя соответствующей системы.

4.1. Настройка параметров чертежа

Работу с системой автоматизированной разработки чертежей следует начинать с установки параметров, таких как единицы измерения, размеры чертежа, параметры сетки и слоев. Для быстрого и точного построения чертежей необходимо, чтобы все эти параметры имели правильные значения. Чертеж можно

построить без сетки и без слоев, но на это уйдет много времени, а изменить получившийся чертеж будет очень сложно.

4.1.1. Единицы измерения

Пользователь должен выбрать формат и точность единиц измерения расстояний и углов. Единицы измерения расстояний могут быть представлены в научном, десятичном, дробном, инженерном и архитектурном форматах. Единицы измерения углов – это градусы, градусы/минуты/секунды, грады, радианы и геодезические единицы. На рис. 4.1 показано диалоговое окно выбора единиц программы AutoCAD[1] Release 14 – одной из наиболее популярных систем автоматизированной разработки чертежей.

4.1.2. Размеры чертежа

Рисуя чертеж на бумаге, вы не можете выйти за границы листа. Точно так же и при работе с графическим устройством чертеж должен иметь определенные границы, поскольку этот чертеж когда-нибудь все равно будет напечатан на бумаге конечного размера. Значит, пользователь должен заранее установить размеры чертежа. Выполняющая эту операцию последовательность команд для AutoCAD приведена ниже.

```
Command: limits
Reset Model Space Limits
0N/0FF/<Lower left corner><0.00.0.00>:10.10
Upper right corner<12.00.9.00>:300.200
```

[1] AutoCAD – название системы автоматизированной разработки чертежей, созданной корпорацией Autodesk.

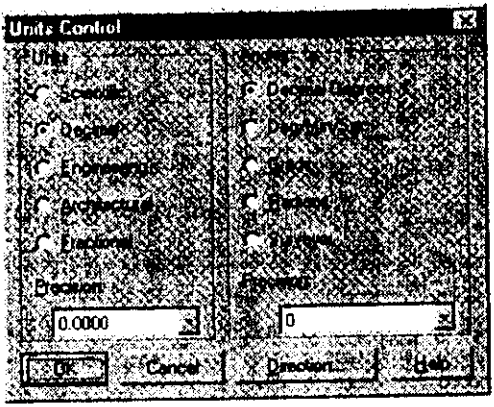


Рис. 4.1. Диалоговое окно установки единиц измерения в AutoCAD Release 14

При выборе размеров чертежа обычно учитывают следующие факторы [15]:

- фактический размер чертежа;
- пространство для нанесения размеров, примечаний, списков материалов и других необходимых данных;
- расстояние между разными видами (чертеж не должен выглядеть загроможденным);
- пространство для рамки и заголовка, если таковые предусмотрены.

Перед тем как задавать размеры чертежа, рекомендуется построить его эскиз, чтобы грубо оценить необходимое пространство. Например, если размеры вида спереди для какого-то объекта равны 6x5 единиц, размеры вида сбоку – 4x5 единиц и размеры вида сверху – 6x4 единиц, ограничения должны быть установлены таким образом, чтобы вместить весь чертеж и все относящиеся к нему данные. Предположим, вы хотите, чтобы расстояние между видами спереди и сбоку составляло 4 единицы, а расстояние между видами спереди и сверху – 3 единицы. Расстояние до границ чертежа пусть составляет 4 единицы слева, 4 справа, 2 сверху и 2 снизу (рис. 4.2). Указанные значения

выбираются таким образом, чтобы готовый чертеж выглядел гармонично.

Установив для себя размеры видов и расстояния между ними, а также расстояния до границ чертежа и от границ до краев бумаги, вы можете вычислить размеры чертежа следующим образом:

- размер по горизонтали =1+4 + 6 + 4 + 4 + 4 + 1 = 24;
- размер по вертикали = 1 + 2 + 5 + 3 + 4 + 2 + 1 = 18.

Итак, размеры чертежа составляют 24x18 единиц. Обратите внимание, что мы положили расстояние между границами чертежа и границами бумаги равным одной единице.

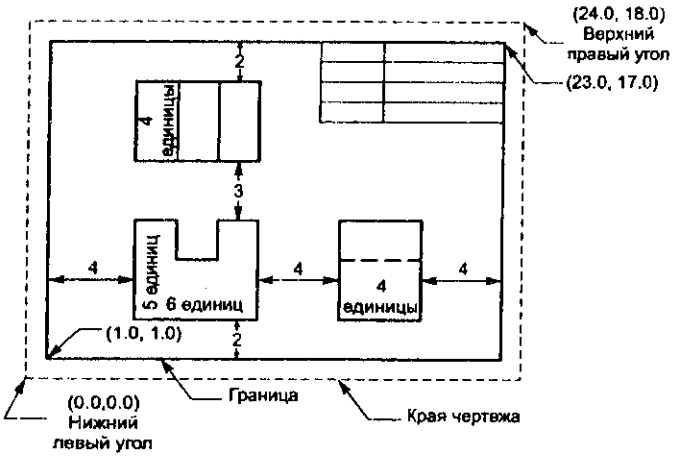


Рис. 4.2. Задание размеров чертежа

Чертеж на рис. 4.2 сделан в масштабе 1:1. Однако если вы хотите распечатать чертеж, чтобы получить твердую копию, вам придется увеличить или уменьшить его в зависимости от размеров листа. Вообще говоря, размер листа определяет границы чертежа, размер шрифта, масштаб чертежа, масштаб толщины линии и другие параметры чертежа. Стандартные размеры листов и соответствующие границы чертежей разных масштабов (в дюймах и миллиметрах) приведены в табл. 4.1.

Таблица 4.1.

Метрическая система единиц

Размер бумаги	Размер листа	Граница чертежа (M1:1)	Граница чертежа (M1:5)	Граница чертежа (M5:1)
A4	210x297	210, 297	1050, 1485	42, 59,4
A3	297x420	297, 420	1485, 2100	59,4 84
A2	420x594	420, 594	2100, 2970	84, 118,8
A1	594x841	594, 841	2970, 4205	118,8, 168,2
A0	841x1189	841, 1189	4205, 5945	168,2, 237,8

4.1.3. Слой

Разбивать чертеж на множество слоев очень удобно, особенно если чертеж достаточно сложный. Распределив поэтажный план здания и схему трубопроводов по отдельным слоям, вы значительно упростите себе задачу. Другими словами, выполнять все операции с отдельным слоем значительно проще, чем с большим чертежом, содержащим все объекты. Однако вам нужно иметь возможность переключаться между режимами просмотра, чтобы получить представление об относительном расположении элементов из разных слоев (например, труб и стен здания). Разделение на слои сохраняет возможность накладывать их друг на друга, не усложняя при этом чертеж по крайней мере в том, что касается выполняемых графических операций. Слой, с которым вы работаете в данный момент, считается активным, тогда как все остальные слои считаются неактивными. Как и любой другой фон, графические элементы неактивных слоев нечувствительны к графическим операциям, таким как выбор или удаление. Поэтому сложность чертежа остается той же самой, как если бы вы работали с одним-единственным активным слоем.

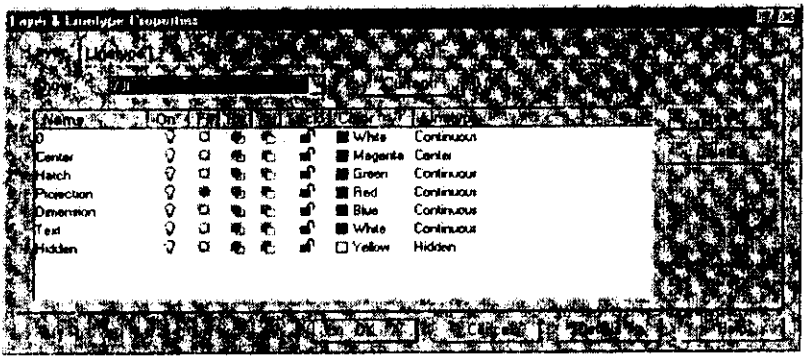


Рис. 4.3. Диалоговое окно управления слоями в AutoCAD Release 14

Функция разделения на слои может эффективно использоваться при построении чертежей отдельных слоев многослойных печатных плат. В этом случае каждый слой может строиться независимо от остальных, однако остается возможность вывести на экран связанные каким-либо образом слои для получения сведений об их относительном положении. Разделение на слои удобно и для построения чертежей отдельных деталей конструкции. Если рисовать каждую деталь в отдельном слое сборочного чертежа, чертеж любой детали легко будет получить, сделав активным нужный слой. На рис. 4.3 показано диалоговое окно управления слоями в AutoCAD Release 14.

4.1.4. Сетка и привязка

В черчении на бумаге широко используются вспомогательные линии, которые строятся заранее при помощи рейсшины. Они облегчают построение линий чертежа и делают их более точными. Линии сетки в системах автоматизированной разработки чертежей имеют то же назначение, что и линии построения в черчении. Горизонтальные и вертикальные линии сетки рисуются на равных расстояниях друг от друга в соответствии с заданным разрешением, а линии чертежа строятся поверх них. В некоторых системах автоматизированной разработки чертежей строятся только точки на перекрестьях линий сетки.

Чтобы провести прямую линию поверх линии сетки, нужно задать положение двух ее концов. Их координаты можно ввести с клавиатуры или указать мышью, установив курсор в нужное положение и нажав кнопку. Вспомните (см. главу 3), что курсор отслеживает движение мыши, когда она находится в режиме локатора. Положение точки, указанное вторым методом, может быть не совсем точным из-за дрожания человеческой руки или неточности механизма мыши. Чтобы справиться с этой проблемой, можно включить привязку курсора к ближайшему пересечению линий сетки. При нажатии кнопки мыши компьютер будет воспринимать точные координаты этого пересечения. Точность задания координат будет определяться разрешением сетки, которое пользователь может настраивать по своему желанию. Эта функция называется *привязкой* (*snapping*). Команда включения сетки в программе AutoCAD выглядит следующим образом.

```
Command: grid
Grid spacing(X) or ON/OFF/Snap/Aspect<0>:0.75 /* расстояние между линиями сетки устанавливается равным 0.75 экранной единицы */
```

4.2. Базовые функции черчения

4.2.1. Прямая линия

В системах автоматизированной разработки чертежей существует множество способов построения отрезков. Наиболее популярным из них является построение по двум конечным точкам. Положение точек может быть задано различными способами. В предыдущем разделе мы предложили два метода: ввод координат с клавиатуры и нажатие кнопки мыши в режиме локатора. Помимо этого вы можете указать конечную точку отрезка, выбрав одну из уже имеющихся на экране точек.

Отрезок можно построить и без явного указания обоих концов. Один из способов — попросить систему провести касательную линию к имеющейся кривой из указанной точки. В этом случае явно указывается только одна точка, а вторую точку система определяет самостоятельно. В качестве атрибутов линии

могут быть указаны ее тип и толщина. Типы линий, поддерживаемые большинством систем автоматизированной разработки чертежей, показаны на рис. 4.4. Построение отрезка в AutoCAD Release 14 осуществляется следующим образом:

Command: line
From point: 1. 1
To point: 5. 2
To point: return

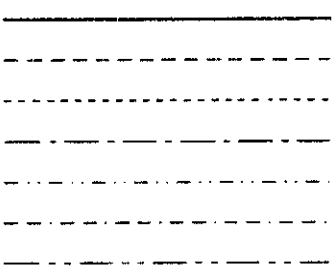


Рис. 4.4. Различные типы линий

4.2.2. Окружность и дуга окружности

Простейший метод задания окружности – указание ее центра и длины радиуса. Другой способ – задание трех точек на самой окружности. Большинство систем автоматизированной разработки чертежей позволяют создавать окружности и другими методами. Например, система может построить окружность, касательную к двум прямым или к другой окружности и прямой. В любом случае вам нужно выбрать соответствующие объекты. Дуга окружности – это частный случай окружности, она определяется заданием точек начала и конца (помимо параметров, задаваемых для обычной окружности). В AutoCAD Release 14 окружность строится следующим образом:

Command: circle
3P/2P/TTR/<Center point>: 5. 5
Diameter7<Radius><current>: 3

Дуга окружности строится так, как показано ниже. В нашем примере дуга проходит через три заданные точки.

Command: arc
Center/<Start point>: 7. 4
Center7End/<Second point>: 6. 5
End point: 6. 3

4.2.3. Сплайн

Сплайны используются для построения произвольных кривых подобно тому, как в черчении от руки это делается с помощью лекала. Пользователь указывает точки на кривой, а система строит интерполяционную кривую, проходящую через эти точки. Получившаяся кривая обычно представляется уравнением третьего порядка. Иногда кривые могут строиться по задающим точкам, которые определяют кривую, но не обязаны лежать на ней.

4.2.4. Удаление

Функция удаления действует как стиральная резинка в черчении на бумаге. Когда вы выбираете графические элементы, такие как точки, отрезки и кривые, они исчезают с экрана. Режим выбора был описан нами в главе 3.

4.2.5. Скругление и снятие фасок

Скругление и *закругление* (*filleting, rounding*) состоят в построении дуги окружности между двумя пересекающимися отрезками (рис. 4.5, *а*) таким образом, что построенная дуга оказывается касательной к обоим отрезкам (рис. 4.5, *б*). Скругление используется для вогнутых углов, а закругление – для выпуклых. *Снятие фасок* (*chamfering*) – примерно то же, что и скругление, но вместо дуги строится отрезок прямой (рис. 4.5, *в*). Скругление и снятие фасок осуществляются в следующем порядке:

1. Указывается радиус скругления или размер фаски.
2. Выбираются два пересекающихся отрезка. Скругление или фаска будут построены около точки пересечения.
3. Ненужные части исходных отрезков удаляются после

построения скругления или фаски. В некоторых системах удаление производится автоматически, а в других это приходится делать вручную.

Рис. 4.5. Скругление и снятие фасок

В AutoCAD скругление можно выполнить следующим образом:

Command: fillet
(TRIM mode) Current fillet radius = 10.00
Polyline/Radius/Trim/<Select first object>: r
Enter fillet radius<current>: 3
(TRIM mode) Current fillet radius - 3.00
Command: fillet
Polyline/Radius/Trim/<Select first object>: /* Выбирается первый отрезок. */
Select second line: /* Выбирается второй отрезок. */

4.2.6. Штриховка

Штриховкой называется заполнение замкнутого многоугольника каким-либо шаблоном. Штриховка часто используется для обозначения сечений в машиностроительных чертежах и выделения разных материалов в архитектурных чертежах. Некоторые наиболее типичные шаблоны, предоставляемые большинством систем автоматизированной разработки чертежей, показаны на рис. 4.6.

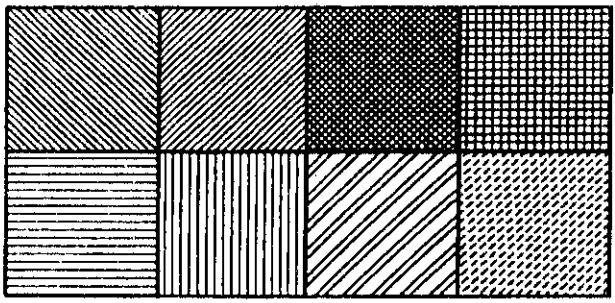


Рис. 4.6. Образцы штриховки

Штриховка начинается с указания замкнутого многоугольника. Эта операция может осуществляться по-разному. В некоторых системах вам придется указать все отрезки, составляющие многоугольник. В других системах достаточно указать один из них, а все остальные система найдет автоматически. Если внутри многоугольника имеются участки, которые штриховать нежелательно, их границы также должны быть указаны. Штриховка – одна из функций систем автоматизированной разработки чертежей, повышающих производительность чертежника. Чертеж с заштрихованными элементами сечений показан на рис. 4.7.

4.3. Функции аннотирования

4.3.1. Простановка размеров

Возможность простановки размеров считается одной из наиболее привлекательных особенностей систем автоматизированной разработки чертежей. Вручную проставить размеры с той же быстротой просто невозможно. В системах автоматизированной разработки чертежей простановка размеров осуществляется следующим образом. Чтобы указать горизонтальный или вертикальный размер, достаточно всего лишь выбрать два графических элемента (обычно точки) и желаемое положение размерной линии. В этом случае расстояние между точками автоматически определяется по чертежу. Стрелки,

размерные линии, выносные линии и значение размера наносятся системой самостоятельно. Система автоматически измеряет расстояние по вертикали, если графические элементы расположены на вертикальной линии, или расстояние по горизонтали, если они находятся на горизонтальной линии. Если объекты расположены как-то иначе, система просит уточнить, какой именно размер вы хотите проставить: вертикальный, горизонтальный или реальный.

Размеры радиусов и диаметров проставляются путем выбора окружности или дуги и последующего указания положения размерной линии. Угловые размеры проставляются аналогичным образом: нужно выбрать два отрезка и указать положение размерного текста. Какой именно угол будет измерен (внешний или внутренний), зависит от порядка выбора отрезков. В каждой системе используется свое собственное соглашение о порядке выбора, поэтому вам будет лучше обратиться к руководству пользователя. Чертеж с проставленными размерами показан на рис. 4.7.

У вас может возникнуть вопрос: зачем проставлять размеры в интерактивном режиме, если чертеж уже содержит все сведения о размерах и положении объектов? Теоретически можно было бы полностью автоматизировать простановку размеров на чертежах, но на практике разработчики сталкиваются со следующими проблемами. Существует много способов проставить размеры на одном и том же чертеже. Вообще говоря, проектировщик учитывает методы производства, контроля и сборки детали, изображенной на чертеже, и на основании главным образом своего опыта выбирает оптимальную схему образмеривания. Воспроизвести опыт проектировщика на компьютере очень сложно. Более того, найти хотя бы одну полную и безызбыточную схему расстановки размеров не так-то просто. В настоящий момент ведутся исследования возможности решения этой задачи.

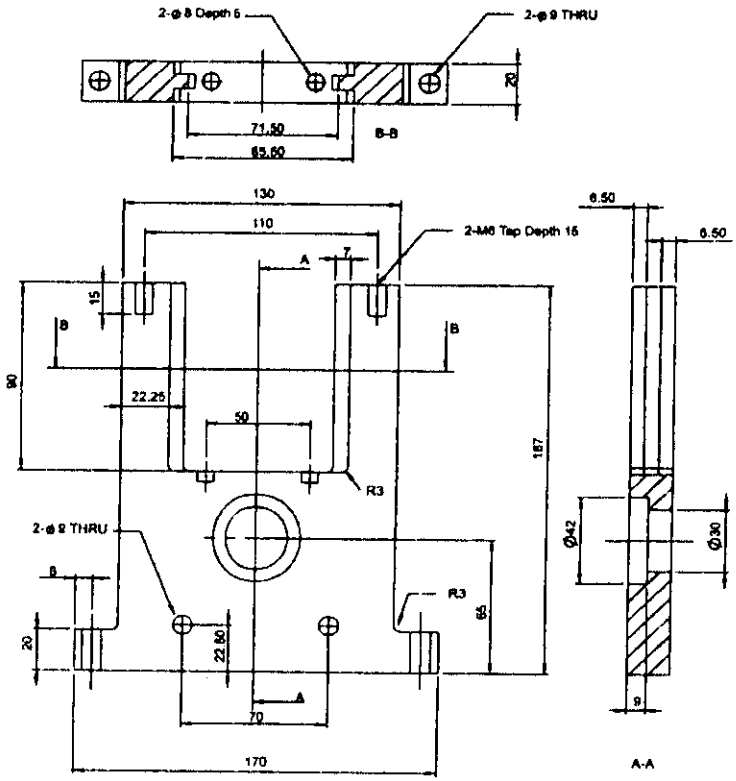


Рис. 4.7. Чертеж с разрезами, штриховкой и примечанием

4.3.2. Примечания

Чтобы добавить к чертежу примечание, то есть текстовую строку, нужно задать расположение и ориентацию этой строки, а также размер и шрифт символов. Последние три параметра обычно имеют некоторые значения по умолчанию, которые используются в том случае, если пользователь не указывает никаких конкретных значений. Пример примечания приведен на рис. 4.7. Добавление примечания в системе AutoCAD Release 14 осуществляется при помощи команды MTEXT.

Command: text
Justify/Style/<start point>: 2. 1
Height<0.20>: 0.25 Rotation angle<0>: Text: MTEXT

4.4. Вспомогательные функции

4.4.1. Копирование

Функция копирования работает точно так же, как операции «Вырезать» и «Вставить» в текстовом процессоре. Набор графических элементов может быть выделен, сохранен в буфере и вставлен в любое место того же или любого другого чертежа. Выбор графических элементов производится путем обведения их прямоугольником нужного размера. Прямоугольник рисуется на экране точно так же, как и при задании окна просмотра. Графические элементы, пересекаемые границами прямоугольника, могут по желанию пользователя быть включены в копируемый набор или исключены из него. Курсор устанавливается в той точке, куда должны быть вставлены выбранные объекты. Функция копирования удобна в том случае, если на чертеже есть повторяющиеся элементы, как, например, на архитектурных чертежах многоквартирных домов. Копирование полезно и при разработке чертежей деталей, потому что вы можете скопировать часть чертежа устройства в целом, после чего уточнить получившийся чертеж.

Частным случаем копирования является зеркальное отражение, которое позволяет строить формы, обладающие осевой симметрией. Эта функция полезна при построении объектов с одной или несколькими осями симметрии. Многие системы автоматизированной разработки чертежей предоставляют дополнительные функции, располагающие повторяющиеся объекты упорядоченно. Например, некоторые системы могут нарисовать несколько головок болтов, расположенных по окружности с определенным шагом, после того как вы нарисуете только одну из них и зададите нужные параметры. Функция копирования вызывается в AutoCAD Release 14 следующим образом.

Command: copy

Select objects: /* Выбирается набор графических элементов. */

<Base point or displacement/Multiple: /* Указывается первая точка. */

Second point of displacement: /* Указывается вторая точка вектора смещения. */

4.4.2. Окно

Иногда при работе со сложным чертежом может потребоваться увеличить его часть. Часто бывает затруднительно выбрать нужный графический элемент, если он сливается с соседними. Эту проблему можно решить, увеличив ту часть чертежа, в которой находится интересующий вас элемент. Использование меньшего окна с отображением в окно просмотра того же размера дает эффект увеличения без изменения числовых параметров графических элементов. Окно определяется заданием двух конечных точек его диагонали аналогично тому, как определяется копируемая область (см. предыдущий раздел). В AutoCAD Release 14 окно изменяется приведенной ниже командой.

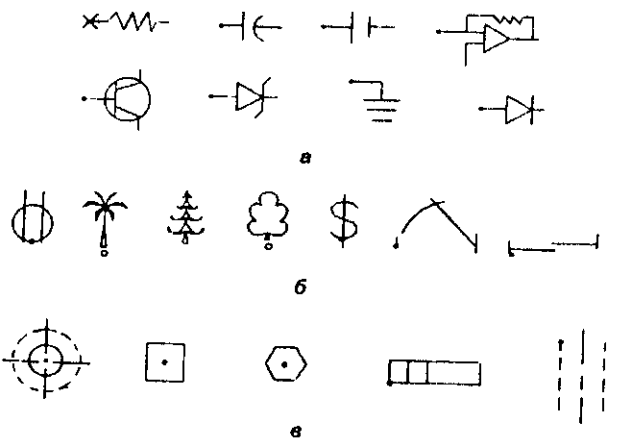


Рис. 4.8. Типичные символы, используемые на чертежах: *а* – электрических систем; *б* – архитектурных; *в* – машиностроительных

Command: zoom
All/Center/Dynamic/Extents/Left/Previous/Vmax/Window/<Scale (X/Xp)>: w
First corner: /* Указывается первая точка диагонали. */
Second corner: /* Указывается вторая точка диагонали. */

4.4.3. Символы

Часто используемые фигуры могут сохраняться в виде символов, а затем вызываться из памяти в любой момент для добавления в нужное место чертежа. Например, значительно упрощается создание машиностроительных чертежей, если формы стандартных компонентов, таких как болты и гайки, а также обозначения шероховатости и допусков поверхности сохраняются в виде символов, которые могут быть в любой момент вызваны и построены. Функции для работы с символами действуют подобно функции копирования и реализуются приблизительно тем же образом. В AutoCAD Release 14 нарисовать символ можно так:

Command: block
Block name (or ?): /* Имя сохраненного символа */

Типичные символы, часто используемые в электрических схемах, а также в архитектурных и машиностроительных чертежах, показаны на рис. 4.8.

4.4.4. Макропрограммирование

Программирование макросов или макропрограммирование заключается в объединении наборов графических команд под одним именем. Если графические команды объединяются в программу, которая называется *макропрограммой* (*macro program*), к ним могут добавляться некоторые условные и арифметические операторы из обычных компьютерных языков. Входные параметры графических команд могут быть определены как переменные, что позволяет задавать макропрограмме разные значения и получать разные чертежи. Макропрограмма такого рода называется *параметрической программой* (*parametric program*), поскольку чертеж, который она строит, зависит от значений, присвоенных

соответствующим параметрам. Хорошим примером параметрической программы может быть автоматическая программа построения чертежей винтов. Пользователь вводит характеристики нагрузки, программа рассчитывает размеры винтов по этим характеристикам, а затем строит их чертежи с учетом вычисленных размеров. В такую параметрическую программу входят арифметические операторы, позволяющие вычислить размеры винтов, а также графические команды, строящие чертеж винта. Функция макропрограммирования очень важна, поскольку она позволяет приспосабливать коммерческие системы автоматизированной разработки чертежей под конкретные приложения. Многообразие параметрических программ, разработанных компанией, фактически может быть мерой эффективности использования этой компанией имеющейся у нее системы автоматизированной разработки чертежей.

4.4.5. Измерения

Функция измерения позволяет выполнять вычисления по готовому или строящемуся чертежу. Система позволяет определить площадь любой области, угол между двумя отрезками, минимальное расстояние между графическими элементами и другие параметры. Эта функция может быть очень полезна, если построение чертежа и проектирование осуществляются в системе автоматизированной разработки чертежей одновременно. Например, проектировщик может проверить, соответствует ли получившаяся конструкция требованиям к площади теплопередачи или к минимальному пространству для обслуживания. В AutoCAD Release 14 измерение осуществляется следующим образом.

```
Command: dist
First point: /* Выбирается первая точка. */
Second point: /* Выбирается вторая точка. */
Distance = <Рассчитанное расстояние>
Angle in XY plane - <Угол в плоскости XY>
Angle from XY plane - <Угол к плоскости XY>
Delta X = <Разность по X>, Delta Y = <Разность по Y>, Delta
Z = <Разность по Z>
```

4.4.6. Дополнительные функции

Помимо функций, описанных в предыдущих разделах, имеются вспомогательные функции, позволяющие обновлять чертежи в базе данных, получать чертежи из базы данных и составлять списки материалов.

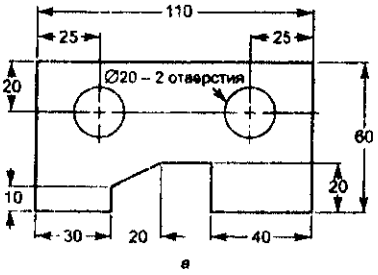
4.5. Совместимость файлов чертежей

Ранее мы показали, что настоящим преимуществом системы автоматизированной разработки чертежей является возможность сохранения файла чертежа в базе данных, откуда его смогут получить сотрудники разных отделов. Этим достоинством легко воспользоваться на практике, если все сотрудники работают в одной системе автоматизированной разработки чертежей и не испытывают проблем с чтением файлов, сделанных другими сотрудниками. Однако преимущество легко утратить, если разные отделы в одной и той же компании работают с разными системами, не способными читать файлы друг друга. Проблема становится еще более серьезной, если речь идет о системах разных производителей. В этом случае единственным реальным методом взаимодействия будет построение чертежей на бумаге и механическое их воспроизведение подобно тому, как это делалось в прошлом.

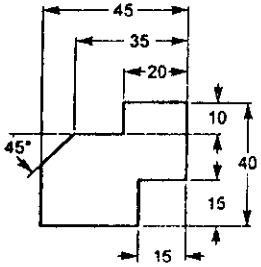
Чтобы избежать этой проблемы, можно потребовать от всех производителей систем автоматизированной разработки чертежей сохранения файлов в стандартном формате. Наиболее популярным стандартным форматом в настоящий момент является Initial Graphics Exchange Specification (IGES), принятый Американским Национальным институтом стандартов (American National Standards Institute – ANSI) под номером Y14.26M. Практически все коммерческие системы автоматизированной разработки чертежей поддерживают формат IGES. Следовательно, файлы, созданные в одной системе, могут быть перенесены в другую систему. Однако для некоторых символов проблема корректного переноса еще не решена. Кроме IGES, существует еще формат DXF – формат чертежей AutoCAD, который становится стандартом де-факто благодаря популярности AutoCAD.

Вопросы и задачи

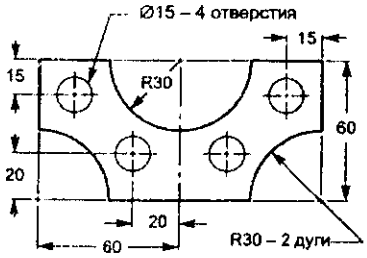
1. Скопируйте приведенные ниже чертежи при помощи любой доступной вам системы автоматизированной разработки чертежей. Размеры указаны в миллиметрах. Вам их проставлять не нужно.



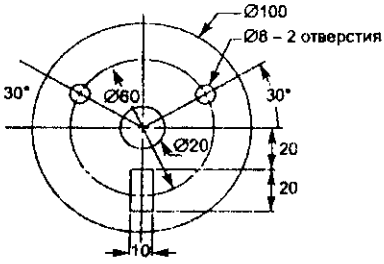
а



б

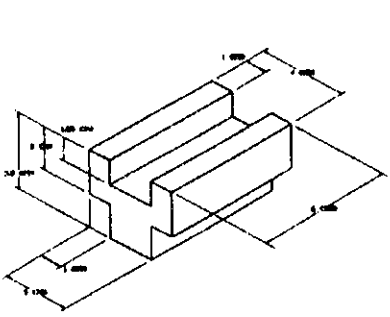


в

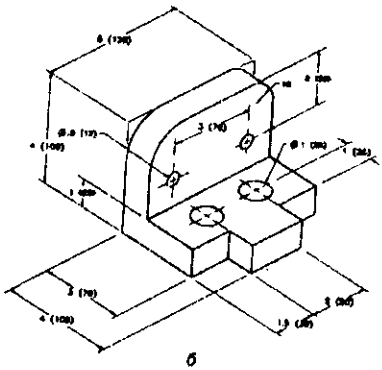


г

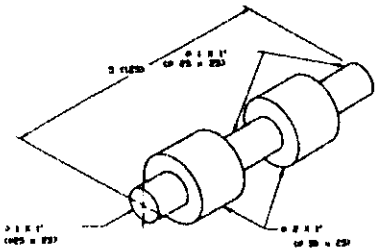
2. Постройте вид сверху, спереди и справа для следующих объектов. Размеры указаны в дюймах, а в скобках даны те же размеры в миллиметрах. Вам их проставлять не нужно. (*Источник: J. Luckow, The Technical Drawing Workbook,* Addison-Wesley Publishing Company, 1994.)



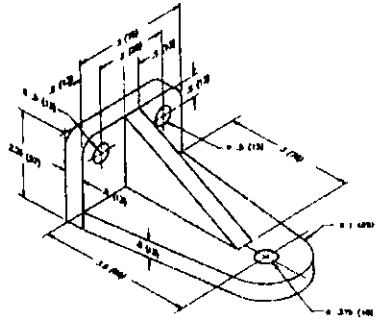
а



б

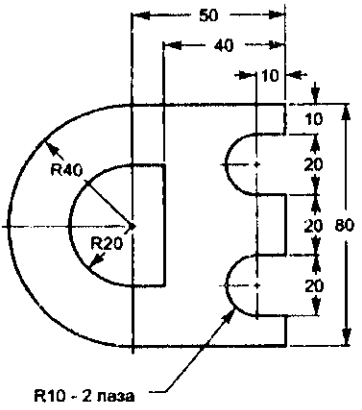


в

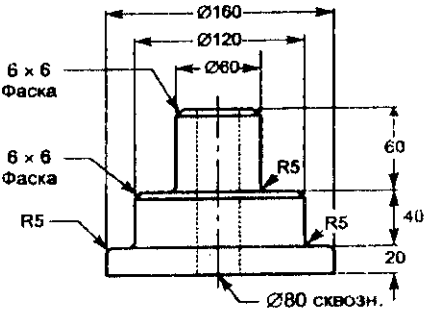


г

3. Скопируйте приведенные ниже чертежи вместе с размерами. Размеры указаны в миллиметрах.



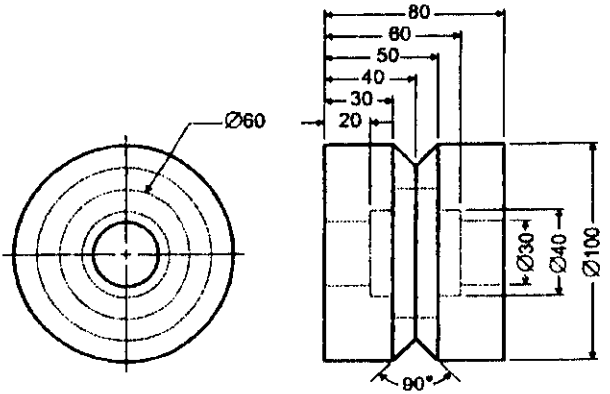
а



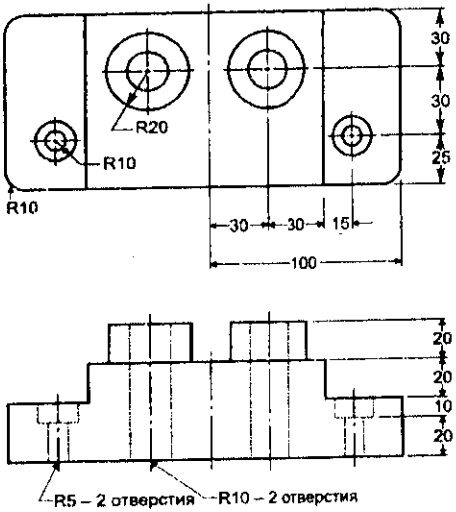
б

4. Создайте приведенные на рисунках чертежи, проставьте размеры. Размеры указаны в миллиметрах.

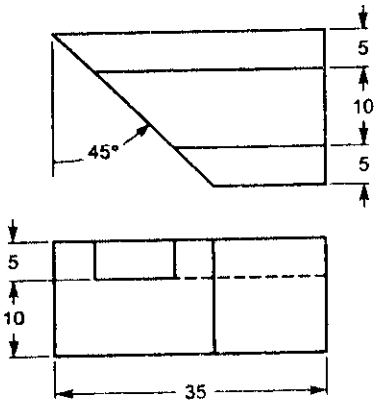
1) Замените вид справа разрезом по вертикальной средней линии вида спереди.



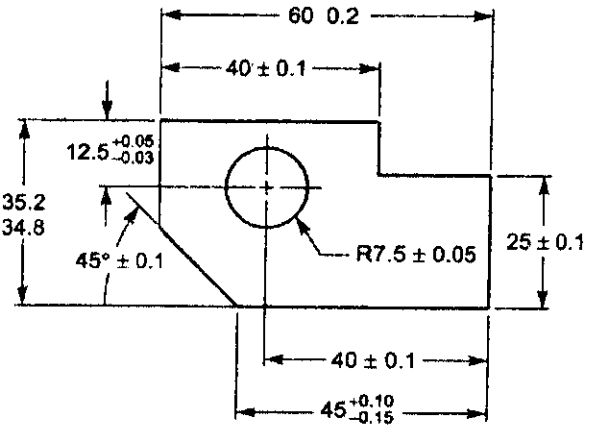
2) Представьте вид спереди в виде частичного сечения и покажите соответствующую секущую плоскость на виде сверху. Добавьте все необходимые примечания.



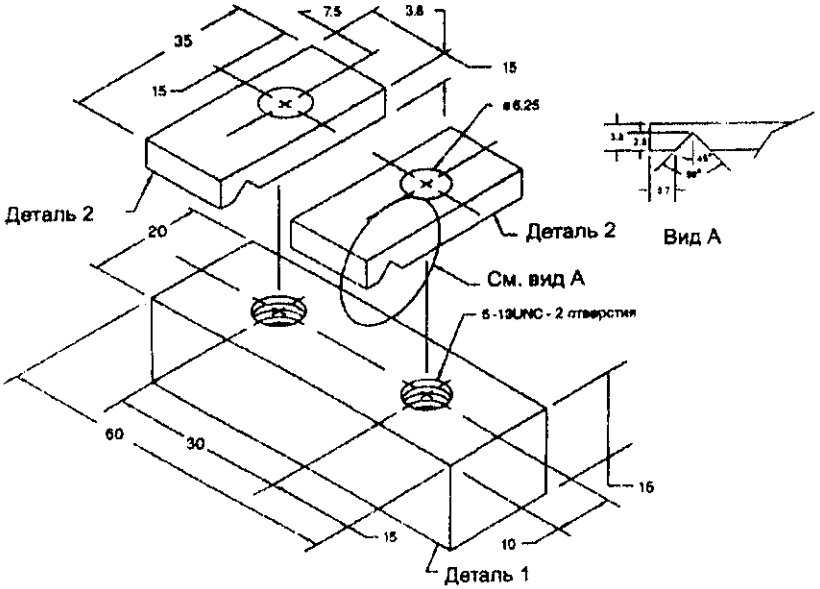
3) Создайте дополнительный вид скошенной поверхности



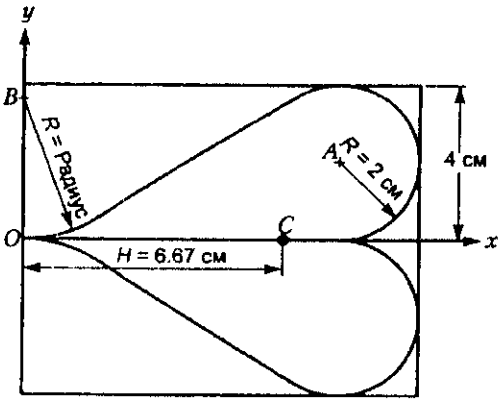
4. Скопируйте приведенный ниже чертеж вместе с размерами и допусками. Размеры указаны в миллиметрах.



5. Подготовьте сборочный чертеж и список деталей в соответствии с приведенными сведениями. Создайте чертежи деталей 1 и 2. Соедините деталь 2 с деталью 1, используя два длинных болта с шестигранными головками M5.00-13 UNC25. На каждый болт наденьте гайку 5,00-13 UNC. Под головками болтов, между деталями 1 и 2, а также между деталью 1 и гайками разместите шайбы 6,2512,51,25. На каждом болте будет три шайбы и гайка. (*Источник: S. Lockhart, A Tutorial Guide to AutoCAD, Release 12,* Addison-Wesley Publishing Company, 1994.)



6. Спроектируйте маятник для часов, который будет удовлетворять приведенным требованиям.



1) Маятник должен быть вписан в прямоугольник размером 108 см.

2) Маятник должен быть симметричен относительно оси *x*.

3) Верхняя часть маятника ограничена кривой, состоящей из дуги окружности с центром в точке *B*, отрезка и второй дуги радиусом 2 см. Отрезок между двумя дугами должен быть касательным к ним.

4) Стержень маятника присоединяется к нему в точке *C*, которая является центром тяжести. Расстояние между точками *C* и *O* должно составлять 6,67 см.

5) Площадь маятника должна лежать в диапазоне 36-44 см^2, чтобы он удовлетворял ограничению на вес.

Определите положение точки *B* и радиус дуги, центром которой является эта точка, испытывая различные варианты чертежей до тех пор, пока не будут удовлетворены все требования. Проверить множество чертежей с различными параметрами достаточно просто: для этого нужно воспользоваться функцией макропрограммирования и написать параметрическую программу.

ГЛОССАРИЙ

Автоматизированное проектирование CAD – эта технология, состоящая в использовании компьютерных систем для облегчения создания, изменения, анализа и оптимизации проектов.

Автоматизированное конструирование (computer-aided engineering – CAE) – это технология, состоящая в использовании компьютерных систем для анализа геометрии *CAD*, моделирования и изучения поведения продукта для усовершенствования и оптимизации его конструкции.

Автоматизированное производство CAM – это технология, состоящая в использовании компьютерных систем для планирования, управления и контроля операций производства через прямой или косвенный интерфейс с производственными ресурсами предприятия.

Алгоритм удаления невидимых граней (back-face removal algorithm) основан на том, что грань объекта может быть видимой только в том случае, если вектор внешней нормали к этой грани направлен в сторону наблюдателя.

Видовой экран (view port) – это область экрана, где будет отображаться проецируемое изображение; в эту область проецируется просматриваемый объем, определяемый «обычным» окном.

Виртуальная система координат устройства (virtual device coordinate system) – фиксирует точку отсчета, направление и масштаб осей для всех рабочих станций.

Графическим интерфейсом пользователя (graphical user interface – GUI) – это программное обеспечение, имеющее возможность открывать окна (области взаимодействия) для отображения меню или значков, а также сопоставлять пунктам меню и значкам какие-либо функции.

Графическое программирование (graphics programming) – на входе и выходе «сочинений» все чаще находится графическая информация.

Дисплейный файл (display list) – это группа команд графической библиотеки, сохраненная для последующего выполнения.

Ломаная (polyline) – это набор соединенных друг с другом отрезков.

Макропрограммирование – объединение наборов графических команд под одним именем.

Окно (window) – область экрана монитора рабочей станции, посредством которого пользователь взаимодействует с вычислительными ресурсами, подключенными к той же сети; в компьютерной графике этот термин имеет иное значение; *окно* – это область пространства, проецируемая на монитор.

Отображение (mapping) – перевод координат из одной системы в другую.

Параллельная проекция (parallel projection) – линии от всех точек объекта проводятся в направлении наблюдателя параллельно направлению наблюдения, а точки пересечения этих линий с экраном формируют проекцию.

Перспективная проекция (perspective projection) – все точки рассматриваемого объекта соединяются с центром проекции, который обычно лежит на линии, соединяющей точку зрения и цель.

Примитивы (primitives) – это элементы графики, которые могут отображаться графической библиотекой.

Программирование на компьютере (computer programming) – написание «сочинения» на языке компьютерных команд в соответствии с предопределенными правилами грамматики.

Система автоматизированной разработки чертежей (computer-aided drafting system) – это программный продукт, позволяющий разработчику в интерактивном режиме создавать и изменять машиностроительные, архитектурные, инженерные чертежи, электрические схемы и чертежи множества других разновидностей.

Система координат устройства (device coordinate system) – определяет положение точки на экране; эта система состоит из горизонтальной оси *u* и вертикальной оси *v*.

Точка зрения (viewpoint) – это глаз наблюдателя.

Точка наблюдения (view site) – это точка объекта, определяющая направление «луча зрения»; вектор, проведенный от точки зрения к цели, задает направление наблюдения.

Удаление невидимых линий (hidden-line removal) – это блокирование отображения отрезков, скрытых от наблюдателя.

Удаление невидимых поверхностей (hidden-surface removal) – это блокирование отображения поверхностей, скрытых от наблюдателя.

ЛИТЕРАТУРА

1. **Глушаков С.В., Лобяк А.В.** AutoCAD 2008. Самоучитель / изд. 2-е, доп и перераб. – М.: АСТ: АСТ МОСКВА: Хранитель, 2008. – 448 с.
2. **Двигатели внутреннего сгорания.** В 3 кн. Кн. 3. Компьютерный практикум: Учеб./ В. Н. Луканин, М.Г. Шатров, А. Ю. Труш и др.; Под ред. В.Н. Луканина. – М.: Высшая школа. 1995.
3. **Корячко В.П., Курейчик В.М., Норенков И.П.** Теоретические основы САПР. - Минск.: Вышэйшая школа. 1987.
4. **Красильникова Г., Самсонов В., Тарелкин С.** Автоматизация инженерно-графических работ. – СПб.: Питер, 2000. – 256 с.
5. **Кунву Ли.** Основы САПР (CAD/CAM/CAE). –СПб.: Питер, 2004. –560 с.
6. **Левицкий В.С.** Машиностроительное черчение и автоматизация выполнения чертежей. – М.: Высш. шк., 1998. – 433 с.
7. **Максимей И.В.** Имитационное моделирование на ЭВМ. М.; 1988.
8. **Петров А.В., Черненький В.М.** Проблемы и принципы создания САПР. – М.: Высшая школа. 1990.
9. **Пятаев А.В.** Автокад. – Т.: ТГАИ. 2008. – 74 с.
10. **Ткачев Д.А.** AutoCAD 2007. – СПб.: Питер, Киев, BHV, 2007. – 464 с.
11. **Тулаев Б.Р.** Основы автоматизированного проектирования: Учебное пособие. – Т.: ТашГТУ. 2004.
12. **Хрящев В., Шипова Г.** Моделирование и создание чертежей в системе AutoCAD. – СПб.: BHV, 2006. – 224 с.
13. **Bendsoe, M.P., Diaz, A., and Kikuchi, N.** «Topology and Generalized Layout Optimization of Elastic Structures», In Bendsoe and Scares (eds.), Topology Design of Structures, Kluwer Academic Publishers, Dordrecht, Holland, 1992.
14. **Kohn, R.V. and Strang, G.** «Optimal Design and Relaxation of

Variational Problems», Communic. Pure and Appl. Math., Vol. 39, (Part 1), pp. 113-137; (Part 2), pp. 139-182; (Part 3), pp. 333-350, 1986.

15. **Tickoo, S.** AutoCAD: A Problem-Solving Approach, Release13 DOS, Delmar, Albany, NY, 1995.
16. **Tulaev B.R. Zakirova N.S.** Basic of computer and design. The textbook. –T., 2005. – 131p.
17. **Zeid, I.** CAD/CAM Theory and Practice, McGraw-Hill, New York, 1991.

ОГЛАВЛЕНИЕ

Редактор Ахметжанова Г.М.

Подписано к печати 20.02.2009 г. Формат 60x84 1/16.
Объем 6 п.л. Тираж 50 экз. Заказ № 67.

Отпечатано в типографии ТГТУ. г.Ташкент,
ул.Талабалар 54. тел: 246-63-84.