Содержание

1. История развития компьютерной графики

Отправной точкой зарождения компьютерной графики можно считать 1930 год, когда в США нашим соотечественником Владимиром Зворыкиным, работавшим в компании «Вестингхаус» (Westinghouse), была изобретена электронно-лучевая трубка (ЭЛТ), впервые позволяющая получать изображения на экране без использования механических движущихся частей. Именно ЭЛТ является прообразом телевизионных кинескопов и компьютерных мониторов.

Первой официально признанной попыткой использования дисплея для вывода изображения из ЭВМ явилось создание в 1950 в Массачусетском технологическом институте (МТИ) для системы противовоздушной обороны военно-морского флота США компьютера «Вихрь» (Whirlwind) с графическим дисплеем. Изобретателем этого дисплея был Джей Форрестер, работавший инженером в МТИ. Впервые была реализована возможность отображать на большой электронно-лучевой трубке текст и графику в реальном времени.

Таким образом, возникновение компьютерной графики можно отнести к 1950-м годам. Сам же термин «компьютерная графика» придумал в 1960 г. сотрудник компании Boeing Вильям Феттер.

Одним из отцов-основателей компьютерной графики считается Айвен Сазерленд (Ivan Sotherland), который впервые в 1963 году все в том же МТИ создал в рамках своей докторской диссертации программу компьютерной графики под названием «Блокнот» (Sketchpad). Программа использовала световое перо для рисования фигур на экране. В программе было реализовано полноценное всплывающее графическое меню, можно было рисовать достаточно простые фигуры (точки, прямые, прямоугольники, дуги окружностей), могла вращать фигуры на экране. Полученные изображения можно было сохранять и восстанавливать.

В 1964 году компания IBM представила разработанный для компании General Motors комплекс DAC-1 для проектирования автомобилей. В этом комплексе была реализована возможность создавать/обрабатывать изображения, считывать графическую информацию с бумажного носителя для дальнейшей обработки на компьютере, выводить на бумагу полученные чертежи.

В 1961 г. студент МТИ Стив Рассел создал первую компьютерную видеоигру Spacewar («Звездная война»). Игра была создана на компьютере PDP-1 корпорации

DEC. Игра была настолько популярна, что корпорация DEC решила поставлять версию Spacewar для каждого своего компьютера, выпущенного в дальнейшем.

В 1961 году Уитни, используя аналоговый компьютер, создал заглавные титры для фильма Вертиго Альфреда Хичкока.

Научный сотрудник Bell Telephone Labs (BTL) Эдвард Зайек в 1963 году создал компьютерный анимационный фильм «Simulation of a two-giro gravity control system», в котором Зайек описывает изменение положения искусственного спутника относительно земли на ее орбите. Сотрудники BTL Кен Нолтон, Фрэнк Синдон и Майкл Нолл также начали работать в области компьютерной графики. Синдон создал фильм «Force, Mass and Motion illustrating Newton's laws of motion in operation». Примерно в то же время другие ученые создавали компьютерную анимацию для иллюстрации своих исследований. Нельсон Макс (Lawrence Radiation Laboratory) создал фильмы, «Flow of a Viscous Fluid» и «Propagation of Shock Waves in a Solid Form». В компании Boeing Aircraft был создан анимационный фильм «Vibration of an Aircraft».

В середине 1960-х крупные корпорации начали проявлять интерес к компьютерной графике: TRW, Lockheed-Georgia, General Electric, Sperry Rand и многие другие.

Компания IBM быстро отреагировала на этот интерес, выпустив графический терминал IBM 2250 — первую доступную для приобретения графическую станцию.

В 1963 году Дуг Энглбарт из Стэнфордского исследовательского института изобрел манипулятор мышь. В то же время разрабатывается цифровой планшет. И мышь и планшет являются 2D устройствами и для того, чтобы задать с их помощью трехмерную точку необходимы двукратные манипуляции (например, выбрать пару X, Y, а только затем Z). Такие продукты, как Spaceball от 3Dconnexion или 3D контроллер от Logitech позволяют манипулировать трехмерными объектами. С их помощью можно эффективно перемещать, масштабировать и поворачивать 3D-модели. 3D-манипуляторы являются дорогостоящими по сей день, что является основной причиной того, что мышь по прежнему является самым популярным указательным устройством.

В 1966 году Сазерленд в МТИ изобрел первый управляемый компьютером головной дисплей (head-mounted display — HMD) — «Дамоклов меч». Этот дисплей показывал отдельное изображение для каждого глаза, что позволяло зрителю увидеть трехмерное изображение.

В конце 60-х годов Университет штата Юта пригласил профессора из Беркли

Дэйва Эванса для создания курсов по компьютерным наукам. Компьютерная графика быстро стала его главным интересом. Научная группа под его руководством и университет Юты станет главным исследовательским центром в мире компьютерной графики.

В 1967 году Эванс пригласил Сазерленда присоединиться к программе компьютерных наук в Университете штата Юта. Там Сазерленд совершенствовал свой головной дисплей.

Университет штата Юта становится центром исследований в области компьютерной графики благодаря Д. Эвансу и А. Сазерленду, которые в это время были самыми заметными фигурами в этой области. Позднее их круг стал быстро расширяться. Учеником Сазерленда стал Эд Кэтмул (1978). Здесь же работал Джон Варнок (John Warnock), автор алгоритма удаления невидимых граней на основе разбиения области (1969) и основатель Adobe System (1982), который совершил революцию в издательском деле своим языком описания страниц PostScript. Другой участник этой школы — Дж.Кларк, будущий основатель компании Silicon Graphics (1982). Все эти исследователи очень сильно продвинули алгоритмическую сторону компьютерной графики.

Будущий создатель алгоритма удаления невидимых поверхностей с использованием Z-буфера — студент по имени Эд Кэтмулл поступил в университет штата Юта в 1970 году и записался на изучение курса графики Сазерленда. Кэтмулл только что пришел из компании «Боинг» и работал над получением ученой степени в области физики. Выросший на диснеевских мультфильмах Кэтмулл любил анимацию, но понимал, что таланта к рисованию у него не было. Теперь Кэтмулл (наряду со многими другими) увидел, что компьютерная графика является естественным продолжением анимации и решил быть частью этой «революции». Первая компьютерная анимация, увиденная Кэтмуллом, была его собственной. Он создал анимацию его открывающейся и закрывающейся ладони. С тех пор одной из его целей стало создание полнометражного компьютерного фильма. Из-за Эванса и Сазерленда университет штата Юта получил славу места исследования компьютерной графики, так что Кэтмулл отправился туда, чтобы узнать больше о 3D-анимации. Позднее Кэтмулл станет основателем фирмы Pixar.

В 1971 г. Гольдштейн и Нагель впервые реализовали метод трассировки лучей с использованием логических операций для формирования трехмерных изображений.

В конце 70-х годов для космических кораблей «Шаттл» появились летные тренажеры, основанные на компьютерной графике. Такие тренажеры представляют собой полнофункциональную модель кабины космического корабля, у которой вместо окон установлены компьютерные мониторы. На этих мониторах синтезируется изображение, которое видят астронавты из взлетающего космического корабля.

В 1970-е годы произошел резкий скачок в развитии вычислительной техники благодаря изобретению микропроцессора, в результате чего началась миниатюризация компьютеров и быстрый рост их производительности. И в это же время начинает интенсивно развиваться индустрия компьютерных игр. Одновременно компьютерная графика начинает широко использоваться на телевидении и в киноиндустрии.

В 1979 году Джордж Лукас, создатель сериала «Звездные войны», организовал в своей фирме «Lucasfilm» отдел, который занимался внедрением последних достижений компьютерной графики в кинопроизводство. В 1982 году на экраны кинотеатров вышел фильм «Трон», в котором впервые использовались кадры, синтезированные на компьютере.

В середине 1970-х годов графика продолжает развиваться в сторону все большей реалистичности изображений. Э. Кэтмул в 1974 г. создает первые алгоритмы текстурирования криволинейных поверхностей. В 1975 г. появляется метод закрашивания Фонга. В 1977 г. Дж. Блин предлагает алгоритмы реалистического изображения шероховатых поверхностей (микрорельефов); Ф. Кроу разрабатывает методы устранения ступенчатого эффекта при изображении контуров (антиалиасинг). Дж. Брезенхем создает эффективные алгоритмы построения растровых образов отрезков, окружностей и эллипсов. Уровень развития вычислительной техники к этому времени уже позволил использовать «жадные» алгоритмы, требующие больших объемов памяти, и в 1978 г. Кэтмул предлагает метод Z-буфера, в котором используется область памяти для хранения информации о «глубине» каждого пикселя экранного изображения. В этом же году Кирус и Бэк развивают алгоритмы клиппирования (отсечения) линий. А в 1979 г. Кэй и Гринберг впервые реализуют изображение полупрозрачной поверхности.

В 1980 г. Т. Уиттед разрабатывает общие принципы трассировки лучей, включающие отражение, преломление, затенение и методы антиэлайзинга. В 1984 г. группой исследователей (Горэл, Торрэнс, Гринберг и др.) была предложена модель излучательности, одновременно развиваются методы прямоугольного клиппирования областей.

В 80-е годы компьютерная графика уже прочно внедряется в киноиндустрию, развиваются приложения к инженерным дисциплинам. В 1990-е годы в связи с возникновением сети Internet у компьютерной графики появляется еще одна сфера приложения.

Здесь перечислены далеко не все серьезные шаги на пути развития графики, но более подробное знакомство с ее историей требует достаточно хорошего представления о теории и алгоритмах этой дисциплины, поэтому мы ограничиваемся лишь кратким обзором. Нетрудно заметить, что приоритет в развитии данного направления в информационных технологиях достаточно прочно удерживают американские исследователи. Но и в отечественной науке тоже были свои разработки, среди которых можно назвать ряд технических реализаций дисплеев, выполненных в разные годы:

- 1968, ВЦ АН СССР, машина БЭСМ-6, вероятно, первый отечественный растровый дисплей с видеопамятью на магнитном барабане;
- 1972, Институт автоматики и электрометрии (ИАиЭ), векторный дисплей «Символ»;
- 1973, ИАиЭ, векторный дисплей «Дельта»;
- 1977, ИАиЭ, векторный дисплей ЭПГ-400;
- 1982, Киев, НИИ периферийного оборудования, векторный дисплей СМ-7316, 4096 символов, разрешение 2048×2048;
- 1979-1984, Институт прикладной физики, серия растровых цветных полутоновых дисплеев «Гамма». Последние дисплеи данной серии имели таблицу цветности, поддерживали окна, плавное масштабирование.

Долгое время считалось, что первым, кому удалось создать компьютерную анимацию с «живыми» существами, был Петер Вальдеш, сделавший несколько компьютерных мультиков в начале 70-х. годов. Недавно пальма первенства перешла советскому математику Николаю Константинову, создавшему математическую модель кошки и сделавшему на ее основе тридцатисекундный мультфильм «Кошечка» в 1968-м. Фильм был создан на основе дифференциальных уравнений, введенных в ЭВМ (это была БЭСМ-4, совершавшая 20 тыс. операций в секунду, имевшая оперативную память на ферритных сердечниках и внешнюю — на магнитных барабанах). Дифференциальные уравнения описывали движение «каркаса» животного, на печать выводились «кадры» движущегося объекта, которые затем уже собирались в фильм.

Таким образом, в процессе развития компьютерной графики можно выделить несколько этапов.

В 1960–1970-е годы она формировалась как научная дисциплина. В это время разрабатывались основные методы и алгоритмы: отсечение, растровая развертка графических примитивов, закраска узорами, реалистическое изображение пространственных сцен (удаление невидимых линий и граней, трассировка лучей, излучающие поверхности), моделирование освещенности.

В 1980-е графика развивается более как прикладная дисциплина. Разрабатываются методы ее применения в самых различных областях человеческой деятельности.

В 1990-е годы методы компьютерной графики становятся основным средством организации диалога «человек—компьютер» и остаются таковыми по настоящее время.

Области приложения компьютерной графики в настоящее время очень широки. В промышленности используется компьютерное моделирование процессов с графическим отображением происходящего на экране. Разработка новых автомобилей проходит на компьютере от стадии первичных эскизов внешнего вида корпуса автомобиля до рассмотрения поведения деталей автомобиля в различных дорожных условиях. В медицине применяются компьютерные томографы, позволяющие заглянуть внутрь тела и поставить правильный диагноз. В архитектуре широко применяются системы визуального автоматизированного проектирования (CAD - Computer Aided Design) которые позволяют разработать проект нового здания, основываясь на методах компьютерной графики. Химики изучают сложные молекулы белков, пользуясь компьютерными средствами визуального отображения данных. В телевидении и кинематографии использование компьютерной графики стало почти необходимым делом. В мире регулярно проводятся выставки, например, такие как SIGGRAPH, картин нарисованных с помощью компьютера. В математике развитие теории фракталов было бы невозможно без компьютеров с соответствующими средствами графического отображения данных. Средства мультимедиа привели к возможности совместного использования различных источников информации, объединяющих в себе статические и видео изображения, текст и звук. Новейшие операционные системы работают в графическом режиме и изначально реализуют в своих функциях методы компьютерной графики.

2. Необходимый математический аппарат

2.1. Декартова система координат

Под описанием двумерного (трехмерного) объекта будем понимать знание о положении каждой точки объекта на плоскости (в пространстве) в любой момент времени. Положение точек будем описывать с помощью декартовой системы координат (ДСК). Базовая информация о двумерной и трехмерной ДСК и координатах точки в ДСК должна быть хорошо знакома из курса геометрии. Напомним лишь некоторые определения, обозначения и соотношения.

2.1.1. Двумерные точки в ДСК

Взаимное расположение осей в двумерной ДСК может быть двух видов. Проведем ось Ox слева направо, как показано на рис. 2.3. Ось Oy при этом может проходить

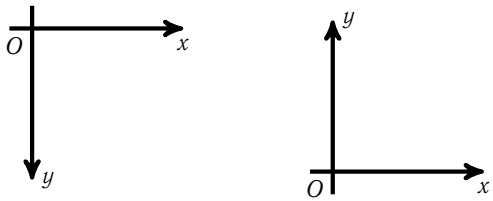


Рис. 2.1. Двумерная декартова система координат: *а) левосторонняя*, *б) правосторонняя*

вниз (рис. 2.1, *а*) или вверх (рис. 2.1, *б*). В первом случае система координат будет называться левосторонней (левой), а во втором случае — правосторонней (правой).

От расположения осей в ДСК зависит знак отмеряемых углов. В левой системе координат будем считать, что угол положительный, если он отмеряется по часовой стрелке. В противном случае — угол отрицательный. В правой системе координат положительное направление отсчета угла — против часовой стрелки.

Рассмотрим двумерную точку P в ДСК с координатами (x, y) (см. рис. 2.2).

Расстояние от точки P до начала координат (длина отрезка OP на рисунке) можно вычислить по формуле

$$|OP| = \sqrt{x^2 + y^2}.$$

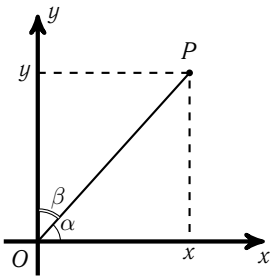


Рис. 2.2. Точка на плоскости в декартовой системе координат

Когда известна длина отрезка OP и угол ϑ между этим отрезком и положительной частью координатной оси, можно вычислить соответствующую координату точки P по формуле:

$$a = |OP| \cos \vartheta. \tag{2.1}$$

Таким образом, для системы координат, изображенной на рисунке 2.2

$$\begin{aligned} x &= |OP| \cos \alpha, \\ y &= |OP| \cos \beta = |OP| \cos \left(\frac{\pi}{2} - \alpha \right) = |OP| \sin \alpha. \end{aligned}$$

Если точка P на плоскости имеет координаты (x, y), то будем писать $P = (x, y)$.

2.1.2. Трехмерные точки в ДСК

Взаимное расположение осей в трехмерной ДСК также может быть двух видов. Проведем ось Ox слева направо, а ось Oy снизу вверх, как показано на рис. 2.3.

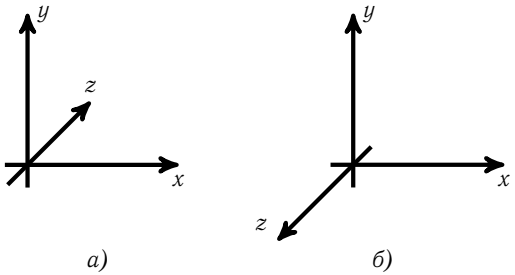


Рис. 2.3. Трехмерная декартова система координат: *а) левосторонняя*, *б) правосторонняя*

Ось Oz при этом может проходить как от наблюдателя в плоскость листа (рис. 2.3, *а*), так и в направлении от плоскости листа к наблюдателю (рис. 2.3, *б*).

В первом случае система координат будет называться левосторонней (левой), а во втором случае — правосторонней (правой).

Более точное определение правой и левой систем координат можно дать следующее. Если посмотреть из положительной полуоси Oz в направлении начала координат, то для совмещения положительной полуоси Ox с положительной полуосью Oy необходимо повернуть Ox относительно начала координат против часовой стрелки — в этом случае имеем правую систему координат; если же поворот производится по часовой стрелке — то система координат левая.

Существует также легкий способ определения вида системы координат по правой или левой руке, как показано на рис. 2.4. Для левой руки большой, указательный и средний пальцы формируют левую тройку ортогональных векторов.

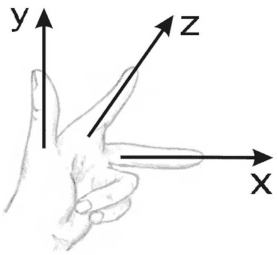


Рис. 2.4. Левая декартова система координат.

В правой системе координат углы вращения относительно осей координат отмеряются против часовой стрелки, если смотреть против направления соответствующей оси. В левой системе координат углы отмеряются по часовой стрелке.

Для точки P в трехмерном пространстве (см. рис. 2.5) расстояние до начала координат определяется по формуле

$$|OP| = \sqrt{x^2 + y^2 + z^2}.$$

При известном значении $|OP|$ значения координат точки можно определить по той-же формуле (2.1), где ϑ — угол между положительной частью соответствующей координатной оси и отрезком, соединяющим начало координат с точкой. Например, на рисунке 2.5 значение y равно $|OP| \cos \beta$.

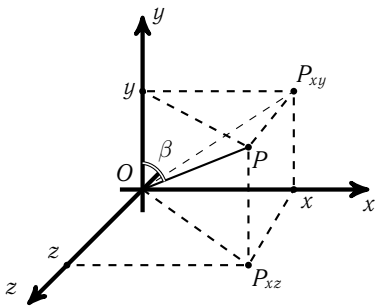


Рис. 2.5. Точка в пространстве в декартовой системе координат

Если точка P в пространстве имеет координаты (x, y, z), то будем записывать $P = (x, y, z)$.

2.2. Векторы. Операции над векторами

Различают понятие свободного и связанного вектора. Связанный вектор или направленный отрезок — упорядоченная пара точек евклидова пространства.

Свободный вектор — класс эквивалентности направленных отрезков. При этом, два направленных отрезка считаются эквивалентными если они: параллельны, равны по длине, одинаково направлены (сонаправлены).

2.2.1. Векторы в двумерном пространстве

Свободный вектор $\bar{p}$ в двумерном пространстве представляется набором из 2-х элементов — координат вектора и обозначается

$$\bar{p} = (p_1, p_2). \tag{2.2}$$

Геометрически вектор $\bar{p}$ можно представить в виде связанного вектора с точкой начала вектора в начале координат, и с точкой конца вектора в точке с координатами (p_1, p_2) (рис. 2.6).

Очевидно, что длину вектора можно вычислить по формуле

$$|\bar{p}| = \sqrt{p_1^2 + p_2^2}. \tag{2.3}$$

Единичным вектором будем называть вектор, имеющий длину равную 1.

Через $\boldsymbol{e}_1$ и $\boldsymbol{e}_2$ будем обозначать единичные векторы сонаправленные с осями координат Ox и Oy соответственно.

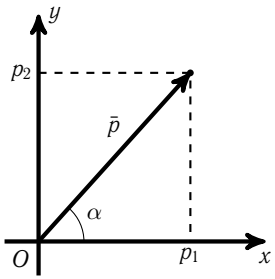


Рис. 2.6. Двумерный вектор

Зная длину вектора и угол его наклона α, следуя (2.1) можно вычислить координаты вектора:

$$p_1 = |\bar{p}| \cos \alpha,$$
$$p_2 = |\bar{p}| \sin \alpha.$$

Суммой векторов $\bar{p} = (p_1, p_2)$ и $\bar{q} = (q_1, q_2)$, называется вектор

$$\bar{r} = \bar{p} + \bar{q} = (p_1 + q_1, p_2 + q_2). \tag{2.4}$$

Результат сложения двух векторов можно представить следующим образом. Изобразим векторы $\bar{p}$ и $\bar{q}$ таким образом, чтобы точка конца вектора $\bar{p}$ совпадала с точкой начала вектора $\bar{q}$ (см. рис. 2.7). Тогда результат сложения — вектор, соединяющий на-

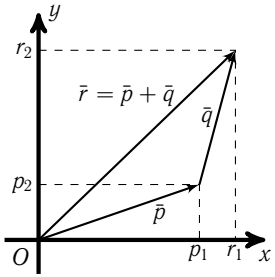


Рис. 2.7. Сумма векторов

чало вектора $\bar{p}$ с концом вектора $\bar{q}$.

Разность векторов — операция обратная сложению. Если

$$\bar{r} = \bar{p} + \bar{q},$$

то справедливо

$$\begin{aligned} \bar{p} &= \bar{r} - \bar{q} = (r_1 - q_1, r_2 - q_2), \\ \bar{q} &= \bar{r} - \bar{p} = (r_1 - p_1, r_2 - p_2). \end{aligned} \tag{2.5}$$

Результат вычитания вектора q из вектора r можно представить следующим образом. Изобразим векторы исходящими из одной точки (см. рис. 2.8). Тогда результат вы-

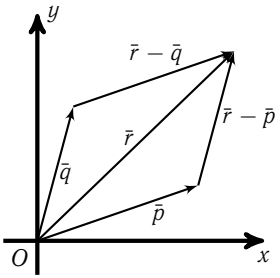


Рис. 2.8. Разность векторов

читания — вектор, начинающийся в точке окончания вектора q и заканчивающийся в точке окончания вектора r.

Результат умножения вектора $\bar{p}$ на скаляр k — вектор

$$k\bar{p} = (kp_1, kp_2). \tag{2.6}$$

Можно представить результат умножения $\bar{p}$ на скаляр k, как вектор, имеющий то же направление, что и $\bar{p}$, и длина которого в k раз больше, чем длина вектора $\bar{p}$ (см. рис. 2.9).

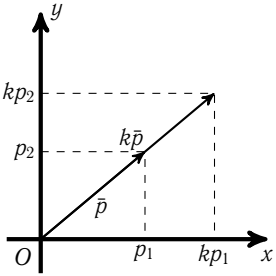


Рис. 2.9. Умножение вектора на скаляр

Произвольный вектор $\bar{p} = (p_1, p_2)$ можно представить в виде

$$\bar{p} = p_1 \boldsymbol{e}_1 + p_2 \boldsymbol{e}_2 \tag{2.7}$$

(см. рис. 2.10).

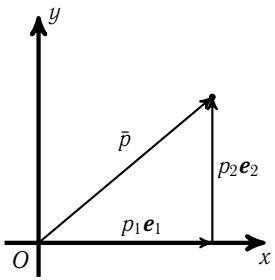


Рис. 2.10. Разложение вектора по осям координат

Скалярным произведением векторов $\bar{p}$ и $\bar{q}$ называется величина

$$\bar{p}\bar{q} = |p||q| \cos \widehat{\bar{p}\bar{q}}, \tag{2.8}$$

где через $\widehat{\bar{p}\bar{q}}$ обозначается угол, отмеренный от вектора $\bar{p}$ к вектору $\bar{q}$.

Из свойств косинуса следует, что скалярное произведение равно нулю, если векторы $\bar{p}$ и $\bar{q}$ перпендикулярны. Кроме того,

$$\boldsymbol{e}_i \boldsymbol{e}_j = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

Если вектор $\bar{q}$ — единичный, то (2.8) превращается в

$$\bar{p}\bar{q} = |p| \cos \widehat{\bar{p}\bar{q}},$$

что представляет интерпретацию (2.1) и является длиной проекции вектора $\bar{p}$ на вектор $\bar{q}$ (см. рис. 2.11). Таким образом значение скалярного произведения $\bar{p}$ на $\bar{q}$ можно представить как произведение длины вектора $\bar{q}$ на длину проекции вектора $\bar{p}$ на $\bar{q}$ или как произведение длины вектора $\bar{p}$ на длину проекции вектора $\bar{q}$ на $\bar{p}$.

Из вышесказанного следует

$$\begin{gathered} \bar{p}\bar{q} = \bar{q}\bar{p}; \\ k\bar{p}\bar{q} = (k\bar{p})\bar{q} = \bar{p}(k\bar{q}); \\ \bar{p}(\bar{q} + \bar{r}) = \bar{p}\bar{q} + \bar{p}\bar{r}. \end{gathered} \tag{2.9}$$

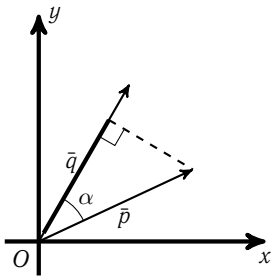


Рис. 2.11. Скалярное произведение векторов

Учитывая последние выкладки можно вывести формулу для получения значения скалярного произведения через координаты векторов:

$$\bar{p}\bar{q} = (p_1\boldsymbol{e}_1 + p_2\boldsymbol{e}_2)(q_1\boldsymbol{e}_1 + q_2\boldsymbol{e}_2) = \\ = p_1q_1\boldsymbol{e}_1\boldsymbol{e}_1 + p_1q_2\boldsymbol{e}_1\boldsymbol{e}_2 + p_2q_1\boldsymbol{e}_2\boldsymbol{e}_1 + p_2q_2\boldsymbol{e}_2\boldsymbol{e}_2 = p_1q_1 + p_2q_2 \quad (2.10)$$

Скалярное умножение вектора на себя (скалярный квадрат) в результате дает квадрат длины этого вектора:

$$\bar{p}^2 = \bar{p}\bar{p} = |p||p|\cos 0 = |p|^2,$$

откуда

$$|p| = \sqrt{\bar{p}^2} \quad (2.11)$$

Нормализацией вектора $\bar{p}$ называется операция получения единичного вектора, сонаправленного вектору $\bar{p}$. Для нормализации вектора $\bar{p}$ нужно разделить каждую координату вектора на его длину:

$$\frac{\bar{p}}{|p|} = \left(\frac{p_1}{|p|}, \frac{p_2}{|p|}\right). \quad (2.12)$$

Псевдоскалярным произведением векторов $\bar{p}$ и $\bar{q}$ называется величина

$$\bar{p} \times \bar{q} = |p||q| \sin \widehat{\bar{p}\bar{q}}. \quad (2.13)$$

Геометрически псевдоскалярное произведение можно представить как ориентированную площадь параллелограмма, построенного на перемножаемых векторах (рис. 2.12).

Из свойств синуса следует

$$\bar{p} \times \bar{q} = -(\bar{q} \times \bar{p}).$$

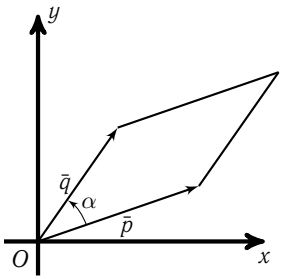


Рис. 2.12. Псевдоскалярное произведение векторов

Псевдоскалярное произведение равно нулю, если векторы $\bar{p}$ и $\bar{q}$ параллельны, что влечет

$$\begin{aligned} \boldsymbol{e}_1 \times \boldsymbol{e}_1 &= 0, \\ \boldsymbol{e}_1 \times \boldsymbol{e}_2 &= 1, \\ \boldsymbol{e}_2 \times \boldsymbol{e}_1 &= -1, \\ \boldsymbol{e}_2 \times \boldsymbol{e}_2 &= 0. \end{aligned}$$

Из вышесказанного следует

$$\begin{aligned} k(\bar{p} \times \bar{q}) &= (k\bar{p}) \times \bar{q} = \bar{p} \times (k\bar{q}); \\ \bar{p} \times (\bar{q} + \bar{r}) &= (\bar{p} \times \bar{q}) + (\bar{p} \times \bar{r}). \end{aligned}$$

Следуя свойствам псевдоскалярного произведения можно вывести формулу получения значения псевдоскалярного произведения через координаты векторов:

$$\begin{aligned} \bar{p} \times \bar{q} &= (p_1\boldsymbol{e}_1 + p_2\boldsymbol{e}_2) \times (q_1\boldsymbol{e}_1 + q_2\boldsymbol{e}_2) = \\ &= p_1q_1(\boldsymbol{e}_1 \times \boldsymbol{e}_1) + p_1q_2(\boldsymbol{e}_1 \times \boldsymbol{e}_2) + p_2q_1(\boldsymbol{e}_2 \times \boldsymbol{e}_1) + p_2q_2(\boldsymbol{e}_2 \times \boldsymbol{e}_2) = p_1q_2 - p_2q_1. \end{aligned}$$

Последнюю формулу можно представить в виде вычисления определителя:

$$\bar{p} \times \bar{q} = \begin{vmatrix} p_1 & p_2 \\ q_1 & q_2 \end{vmatrix}.$$

2.2.2. Векторы в трехмерном пространстве

Свободный вектор $\bar{p}$ в трехмерном пространстве представляется тройкой координат обозначается

$$\bar{p} = (p_1, p_2, p_3). \tag{2.14}$$

Геометрически вектор $\bar{p}$ можно представить в виде связанного вектора с точкой начала вектора в начале координат, и с точкой конца вектора в точке (p_1, p_2, p_3) (рис. 2.13).

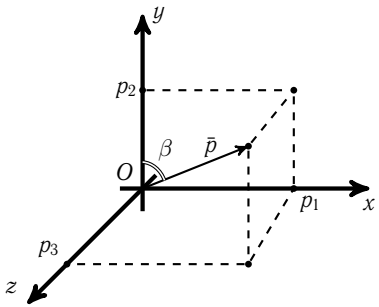


Рис. 2.13. Трехмерный вектор

Длину вектора $\bar{p}$ можно вычислить по формуле

$$|\bar{p}| = \sqrt{p_1^2 + p_2^2 + p_3^2}. \tag{2.15}$$

Вычисление i-й координаты вектора $\bar{p}$ по его длине сводится к формуле (2.1) вычисления i-й координаты точки конца вектора. Например, на рисунке 2.13 значение p_2 равно $|\bar{p}| \cos \beta$

Операции над двумерными векторами и их свойства, заданные формулами (2.4)–(2.12) легко расширяются для трехмерных векторов:

1. Сумма

$$\bar{p} + \bar{q} = (p_1 + q_1, p_2 + q_2, p_3 + q_3);$$

2. Разность

$$\bar{p} - \bar{q} = (p_1 - q_1, p_2 - q_2, p_3 - q_3);$$

3. Умножение на скаляр k

$$k\bar{p} = (kp_1, kp_2, kp_3);$$

4. Разложение вектора по проекциям на оси

$$\bar{p} = p_1\boldsymbol{e}_1 + p_2\boldsymbol{e}_2 + p_3\boldsymbol{e}_3;$$

5. Скалярное произведение

$$\bar{p}\bar{q} = |p| \cos \widehat{\bar{p}\bar{q}},$$
$$\bar{p}\bar{q} = p_1q_1 + p_2q_2 + p_3q_3,$$
$$\bar{p}\bar{q} = \bar{q}\bar{p};$$
$$k\bar{p}\bar{q} = (k\bar{p})\bar{q} = \bar{p}(k\bar{q});$$
$$\bar{p}(\bar{q} + \bar{r}) = \bar{p}\bar{q} + \bar{p}\bar{r};$$
$$|\bar{p}| = \sqrt{\bar{p}^2};$$

6. Нормализация вектора

$$\frac{\bar{p}}{|p|} = \left(\frac{p_1}{|p|}, \frac{p_2}{|p|}, \frac{p_3}{|p|}\right).$$

Операция псевдоскалярного произведения отсутствует для векторов в трехмерном пространстве. Вместо нее определяется операция векторного произведения.

Операция векторного произведения $\bar{p}$ и $\bar{q}$ обозначается $\bar{p} \times \bar{q}$. Результатом векторного произведения $\bar{p}$ и $\bar{q}$ называется новый вектор $\bar{r}$, который удовлетворяет следующим условиям:

1. длина вектора $\bar{r}$ вычисляется по формуле

$$\bar{p} \times \bar{q} = |\bar{p}||\bar{q}| \sin \widehat{\bar{p}\bar{q}};$$

2. вектор $\bar{r}$ перпендикулярен векторам $\bar{p}$ и $\bar{q}$;
3. векторы $\bar{p}$, $\bar{q}$, $\bar{r}$ образуют левую тройку векторов (рис. 2.14).

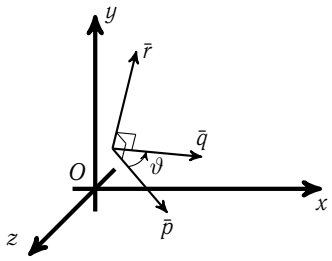


Рис. 2.14. Векторное произведение

Из определения следуют свойства векторного произведения:

$$\bar{p} \times \bar{q} = -(\bar{q} \times \bar{p}),$$
$$k(\bar{p} \times \bar{q}) = (k\bar{p}) \times \bar{q} = \bar{p} \times (k\bar{q}),$$
$$\bar{p} \times (\bar{q} + \bar{r}) = (\bar{p} \times \bar{q}) + (\bar{p} \times \bar{r}).$$

Кроме того, векторное произведение равно нулю, если перемножаемые векторы параллельны.

Для векторов $\boldsymbol{e}_1$, $\boldsymbol{e}_2$ и $\boldsymbol{e}_3$ в правой системе координат имеют место соотношения:

$$\boldsymbol{e}_1 \times \boldsymbol{e}_2 = \boldsymbol{e}_3,$$
$$\boldsymbol{e}_2 \times \boldsymbol{e}_3 = \boldsymbol{e}_1,$$
$$\boldsymbol{e}_3 \times \boldsymbol{e}_1 = \boldsymbol{e}_2.$$

В левой системе координат последние соотношения выглядят иначе:

$$\begin{aligned}\boldsymbol{e}_2 \times \boldsymbol{e}_1 &= \boldsymbol{e}_3,\\ \boldsymbol{e}_3 \times \boldsymbol{e}_2 &= \boldsymbol{e}_1,\\ \boldsymbol{e}_1 \times \boldsymbol{e}_3 &= \boldsymbol{e}_2.\end{aligned}$$

Раскладывая множители по осям координат можно выразить результат векторного произведения через координаты множителей (в правой системе координат):

$$\begin{aligned}\bar{p} \times \bar{q} = (p_1\boldsymbol{e}_1 + p_2\boldsymbol{e}_2 + p_3\boldsymbol{e}_3) \times (q_1\boldsymbol{e}_1 + q_2\boldsymbol{e}_2 + q_3\boldsymbol{e}_3) = \\ = p_1q_1(\boldsymbol{e}_1 \times \boldsymbol{e}_1) + p_1q_2(\boldsymbol{e}_1 \times \boldsymbol{e}_2) + p_1q_3(\boldsymbol{e}_1 \times \boldsymbol{e}_3) + p_2q_1(\boldsymbol{e}_2 \times \boldsymbol{e}_1) + \\ + p_2q_2(\boldsymbol{e}_2 \times \boldsymbol{e}_2) + p_2q_3(\boldsymbol{e}_2 \times \boldsymbol{e}_3) + p_3q_1(\boldsymbol{e}_3 \times \boldsymbol{e}_1) + p_3q_2(\boldsymbol{e}_3 \times \boldsymbol{e}_2) + p_3q_3(\boldsymbol{e}_3 \times \boldsymbol{e}_3) = \\ = p_1q_2\boldsymbol{e}_3 + p_1q_3(-\boldsymbol{e}_2) + p_2q_1(-\boldsymbol{e}_3) + p_2q_3\boldsymbol{e}_1 + p_3q_1\boldsymbol{e}_2 + p_3q_2(-\boldsymbol{e}_1) = \\ = (p_2q_3 - p_3q_2)\boldsymbol{e}_1) + (p_3q_1 - p_1q_3)\boldsymbol{e}_2 + (p_1q_2 - p_2q_1)\boldsymbol{e}_3 = \\ = (p_2q_3 - p_3q_2,\ p_3q_1 - p_1q_3,\ p_1q_2 - p_2q_1).\end{aligned}$$

Последнюю формулу можно представить в виде результата вычисления определителя:

$$\bar{p} \times \bar{q} = \begin{vmatrix}\boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3\\ p_1 & p_2 & p_3\\ q_1 & q_2 & q_3\end{vmatrix}.$$

2.2.3. Радиус-векторы

Радиус-вектор — связанный вектор, начало которого находится всегда в начале координат. Это свойство радиус-векторов позволяет поставить во взаимно однозначное соответствие всем точкам пространства соответствующие им радиус-векторы.

Формально это соответствие запишем в следующем виде. Пусть точка P имеет координаты (x, y, z). Тогда точке P взаимнооднозначно соответствует радиус-вектор $\bar{p} = (x, y, z)$. Таким образом, можно легко переходить от координат точек к радиус-векторам и обратно.

Отметим, что радиус-вектор иногда определяют как преобразование переноса точки из начала координат в заданную точку пространства с известными координатами. При этом умножение радиус-вектора $\bar{p}$ на число k означает перенос точки из начала координат в направлении вектора $\bar{p}$ на расстояние $k|\bar{p}|$.

Сложение радиус-векторов $\bar{p}$ и $\bar{q}$ можно рассматривать как перенос точки P по направлению вектора $\bar{q}$ на расстояние $|\bar{q}|$.

Направленный отрезок P_1P_2 можно представить в виде вектора $\bar{p}_2 - \bar{p}_1$, где $\bar{p}_1$ и $\bar{p}_2$ радиус-векторы точек P_1 и P_2 соответственно.

2.3. Уравнение прямой

Рассмотрим теперь каким образом можно использовать координаты точек и радиус-векторы для описания прямых в двумерном пространстве. Под описанием прямой понимаем знание того принадлежит ли точка с заданными координатами нашей прямой или нет. То есть нужно получить некую математическую зависимость или уравнение прямой.

Прямую можно задать, определив пару точек P_1 и P_2, через которые она должна проходить.

Проведем от точки P_1 к точке P_2 обычный вектор равный разности векторов $\bar{p}_2 - \bar{p}_1$. Этому вектору соответствует параллельный ему радиус-вектор $\bar{p}^* = \bar{p}_2 - \bar{p}_1$, как показано на рис. 2.15. Тогда радиус-вектор $\bar{p}$, определяющий некоторую точку P на

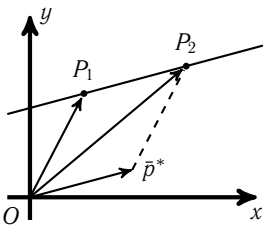


Рис. 2.15. Вывод уравнения прямой.

прямой, можно получить сложением, например, вектора $\bar{p}_1$ и вектора $\bar{p}^*$, умноженного на некоторое число t:

$$\bar{p} = \bar{p}(t) = \bar{p}_1 + \bar{p}^* t, \tag{2.16}$$

откуда получаем

$$\bar{p}(t) = \bar{p}_1 + (\bar{p}_2 - \bar{p}_1)t \tag{2.17}$$

— параметрическое уравнение прямой, заданное в векторной форме. Число t называют параметром. Когда t пробегает значения от $-\infty$ до ∞ вектор $\bar{p}(t)$ пробегает радиус-векторы всех точек заданной прямой.

Из векторного уравнения (2.17) получаем два скалярных (для координат вектора $\bar{p}(t)$):

$$\begin{cases} x(t) = x_1 + (x_2 - x_1)t \\ y(t) = y_1 + (y_2 - y_1)t. \end{cases}$$

Эту систему систему можно записать в виде

$$\begin{cases} x(t) - x_1 = (x_2 - x_1)t \\ y(t) - y_1 = (y_2 - y_1)t, \end{cases}$$

откуда

$$(x(t) - x_1)(y_2 - y_1) - (y(t) - y_1)(x_2 - x_1) = 0.$$

Здесь $x(t)$ и $y(t)$ — координаты произвольной точки заданной прямой. Последнее равенство уже не зависит от параметра t, а поэтому заменим обозначения $x(t)$ и $y(t)$ на x и y соответственно:

$$(x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1) = 0.$$

Последнее равенство представим в форме

$$Ax + By + C = 0, \tag{2.18}$$

где

$$\begin{aligned} A &= y_2 - y_1, \\ B &= x_1 - x_2, \\ C &= y_1 x_2 - x_1 y_2. \end{aligned}$$

Уравнение (2.18) с указанными значениями A, B и C представляет собой частный случай уравнения прямой на плоскости. Очевидно, что при одновременном умножении значений A, B и C на одну и ту же константу, определяемое этим уравнением множество точек не меняется.

Получить параметрическое уравнение прямой в пространстве можно тем же способом, что и на плоскости. При этом можно убедиться, что параметрическое уравнение в векторной форме не изменится. Но этому векторному уравнению будет соответствовать система уже не двух, а трех уравнений в скалярной форме:

$$\begin{cases} x(t) = x_1 + (x_2 - x_1)t \\ y(t) = y_1 + (y_2 - y_1)t \\ z(t) = z_1 + (z_2 - z_1)t. \end{cases}$$

Дополнив параметрическое уравнение прямой неравенством $0 \leqslant t \leqslant 1$ получим параметрическое уравнение отрезка P_1P_2:

$$\begin{cases} \bar{p}(t) = \bar{p}_1 + (\bar{p}_2 - \bar{p}_1)t, \\ 0 \leqslant t \leqslant 1 \end{cases} \tag{2.19}$$

или

$$\begin{cases} x(t) = x_1 + (x_2 - x_1)t \\ y(t) = y_1 + (y_2 - y_1)t \\ z(t) = z_1 + (z_2 - z_1)t \\ 0 \leqslant t \leqslant 1. \end{cases} \tag{2.20}$$

В дальнейшем, прямую, на которой лежит отрезок P_1P_2, будем называть несущей прямой этого отрезка или просто прямой отрезка P_1P_2.

2.4. Уравнение плоскости

Используем свойства скалярного произведения для получения уравнения плоскости. Рассмотрим некоторую плоскость в пространстве и некоторую точку $P_1 = (x_1, y_1, z_1)$, про которую известно, что она лежит в этой плоскости, как показано на рис. 2.16. Возьмем также некоторый вектор $\bar{n} = (n_x, n_y, n_z)$, перпендикулярный на-

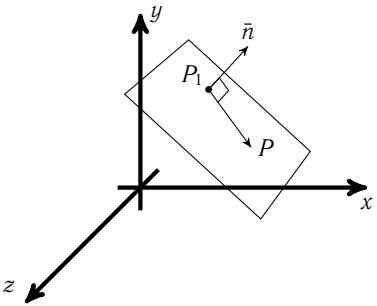


Рис. 2.16. Вывод уравнения плоскости

шей плоскости. Этот вектор назовем нормалью к плоскости. Пусть теперь требуется определить, принадлежит ли некоторая точка P плоскости или нет. Для этого заметим, что для любой точки $P = (x, y, z)$, принадлежащей плоскости, вектор разности радиус-векторов $\bar{p} - \bar{p}_1$ и вектор нормали $\bar{n}$ — перпендикулярны. А это значит, что их скалярное произведение равно нулю:

$$\bar{n}(\bar{p} - \bar{p}_1) = 0. \tag{2.21}$$

Уравнение (2.21) уже представляет собой уравнение плоскости в векторной форме.

Его можно преобразовать в скалярную форму:

$$n_x x + n_y y + n_z z - n_x x_1 - n_y y_1 - n_z z_1 = 0.$$

Последнее равенство представим в виде

$$Ax + By + Cz + D = 0, \tag{2.22}$$

где

$$\begin{aligned} A &= n_x, \\ B &= n_y, \\ C &= n_z, \\ D &= -n_x x_1 - n_y y_1 - n_z z_1. \end{aligned}$$

Форма (2.22) задает уравнение плоскости. Как и в случае с уравнением прямой в двумерном пространстве, коэффициенты уравнения плоскости задаются с точностью до общего множителя.

2.5. Полярность прямой/плоскости

Пусть задана прямая ℓ на плоскости своим уравнением:

$$Ax + By + C = 0. \tag{2.23}$$

Рассмотрим функцию

$$f(x, y) = Ax + By + C.$$

Из (2.23) следует, что $f(x, y) = 0$ в точках прямой ℓ. Следовательно неравенство $f(x, y) > 0$ удовлетворяется только во всех точках плоскости, лежащих по одну сторону от прямой ℓ, а $f(x, y) < 0$ — во всех точках плоскости, лежащих по другую сторону от ℓ. Будем говорить, что первое неравенство определяет положительную, а второе отрицательную полуплоскость относительно прямой ℓ.

Полярность прямой ℓ можно поменять, если задать новое уравнение прямой умножив обе части равенства (2.23) на отрицательную константу.

В трехмерном случае аналогичную ситуацию наблюдаем при задании уравнения плоскости. Плоскость делит трехмерное пространство на два полупространства. Полярность плоскости зависит от выбранного уравнения плоскости.

2.6. Геометрические преобразования

Упомянутые ниже преобразования играют роль элементарных преобразований, из которых складывается большинство операций, применяемых к объектам при построении изображений.

Применение любого из последующих преобразований к изображению (фрагменту изображения) означает выполнение этого преобразования для каждой точки исходного изображения (фрагмента изображения).

2.6.1. Двумерные преобразования

При описании преобразований будем предполагать, что $P = (x, y)$ — исходная точка; $P' = (x', y')$ — точка после преобразования.

Перенос (параллельный перенос)

Преобразование переноса подразумевает перемещение изображения (или его части) на новую позицию относительно начала координат. Полученное изображение после преобразования должно сохранять свои размеры и углы наклона к осям координат.

Преобразование переноса в плоском случае задается соотношениями:

$$\begin{cases} x' = x + T_x, \\ y' = y + T_y, \end{cases} \tag{2.24}$$

где T_x, T_y — величины сдвига по осям Ox и Oy соответственно (см. рис. 2.17).

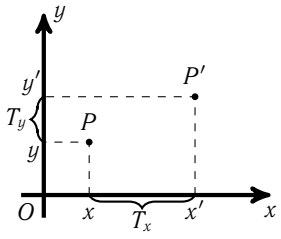


Рис. 2.17. Преобразование перенос, двумерный случай

Масштабирование относительно начала координат

При преобразовании масштабирования относительно начала координат меняется расположение всех точек изображения (за исключением точки в начале координат): расстояние от начала координат до проекции точки на ось увеличивается в соответствующее количество раз.

Преобразование масштабирования относительно начала координат имеет вид:

$$\begin{cases} x' = xS_x, \\ y' = yS_y, \end{cases} \tag{2.25}$$

S_x, S_y — коэффициенты масштабирования по осям Ox и Oy соответственно.

Преобразование масштабирования, в котором $S_x = S_y$ называется равномерным масштабированием. При равномерном масштабировании сохраняются пропорции изображения и углы наклона отрезков к осям координат; расстояния от всех точек изображения до начала координат увеличиваются в одно и то же количество раз. На рисунке 2.18 приведен пример равномерного масштабирования с коэффициентом $S_x = S_y = 2.3$.

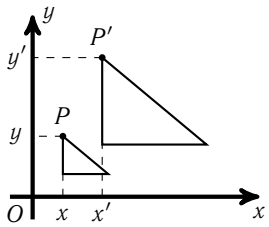


Рис. 2.18. Преобразование масштабирование, двумерный случай

Поворот

При преобразовании поворота относительно начала координат каждая точка изображения меняет свой угол наклона к осям координат на заданную величину (ϑ). При этом расстояние от каждой точки до начала координат остается неизменным.

Преобразование поворота относительно начала координат против часовой стрелки на угол ϑ (см. рис. 2.19):

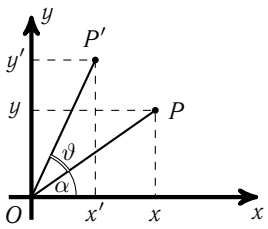


Рис. 2.19. Преобразование поворот, двумерный случай

$$\begin{cases} x' = x\cos\vartheta - y\sin\vartheta, \\ y' = x\sin\vartheta + y\cos\vartheta. \end{cases} \tag{2.26}$$

Соотношения (2.26) легко получаются из (2.1). Действительно, координаты точки P на рис. 2.19:

$$\begin{aligned} x &= |OP|\cos\alpha, \\ y &= |OP|\sin\alpha. \end{aligned} \tag{2.27}$$

После проведения поворота координаты новой точки (P') равны

$$\begin{aligned} x' &= |OP'|\cos(\alpha+\vartheta), \\ y' &= |OP'|\sin(\alpha+\vartheta). \end{aligned}$$

Принимая во внимание, что $|OP'| = |OP|$ и раскрывая формулы синуса и косинуса суммы получим

$$\begin{aligned} x' &= |OP|(\cos\alpha\cos\vartheta - \sin\alpha\sin\vartheta), \\ y' &= |OP|(\sin\alpha\cos\vartheta + \cos\alpha\sin\vartheta). \end{aligned}$$

Раскроем скобки и перегруппируем:

$$\begin{aligned} x' &= (|OP|\cos\alpha)\cos\vartheta - (|OP|\sin\alpha)\sin\vartheta, \\ y' &= (|OP|\sin\alpha)\cos\vartheta + (|OP|\cos\alpha)\sin\vartheta, \end{aligned}$$

откуда, с учетом (2.27), получается (2.26).

Зеркальное отражение

Под зеркальным отражением понимается семейство симметричных преобразований изображения относительно осей координат. Каждое такое преобразование заключается в изменении знака соответствующей координаты каждой точки. Так, зеркальное отражение относительно оси Ox заключается в том, что координата x каждой точки остается неизменной, а координата y меняет знак (см. рис. 2.20):

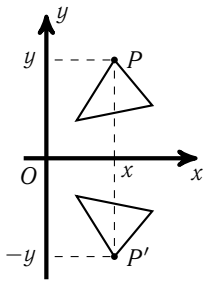


Рис. 2.20. Зеркальное отражение относительно Ox

$$\begin{cases} x' = x, \\ y' = -y. \end{cases} \tag{2.28}$$

. При зеркальном отражении относительно Oy неизменной остается координата y каждой точки, а координата x меняет знак (см. рис. 2.21):

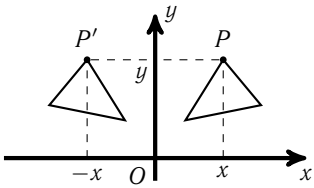


Рис. 2.21. Зеркальное отражение относительно Oy

$$\begin{cases} x' = -x, \\ y' = y. \end{cases} \tag{2.29}$$

.

Обратные преобразования

Для каждого из преобразований можно определить обратное преобразование. Так для преобразования переноса с величинами сдвига T_x и T_y обратным преобразованием будет перенос с величинами сдвига $-T_x$ и $-T_y$. Для преобразования масштабирования с коэффициентами S_x и S_y обратным преобразованием будет масштабирование с коэффициентами $1/S_x$ и $1/S_y$. Для преобразования поворота на угол ϑ обратным будет поворот на угол $-\vartheta$.

Двойственность преобразований

Любое из приведенных выше преобразований можно рассматривать как преобразование изображения (например, перенос точки из одной части изображения в другую). Но в то же время преобразование, примененное ко всем точкам изображения можно рассматривать как преобразование системы координат. Так, например, перенос точек в положительном направлении оси Ox можно рассматривать как сдвиг системы координат в отрицательном направлении оси Ox (сравни рисунки 2.17 и 2.22), а поворот

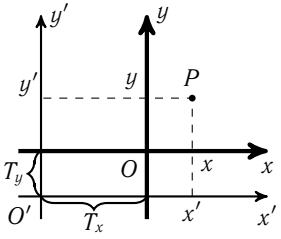


Рис. 2.22. Преобразование перенос начала координат, двумерный случай

точек против часовой стрелки относительно начала координат можно рассматривать как поворот системы координат по часовой стрелке (сравни рисунки 2.19 и 2.23).

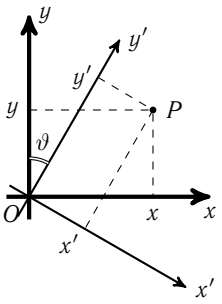


Рис. 2.23. Преобразование поворот системы координат, двумерный случай

Так как координаты любой точки совпадают с координатами ее радиус-вектора, любое из перечисленных преобразований можно отнести к векторам.

2.6.2. Трехмерные преобразования

При описании преобразований будем предполагать, что $P = (x, y, z)$ — исходная точка; $P' = (x', y', z')$ — точка после преобразования. Считаем, что преобразования выполняются в правой ДСК.

Перенос (параллельный перенос)

Преобразование переноса в трехмерном случае имеет вид:

$$\begin{cases} x' = x + T_x, \\ y' = y + T_y, \\ z' = z + T_z, \end{cases} \tag{2.30}$$

где T_x, T_y, T_z — величины сдвига по осям Ox, Oy и Oz соответственно.

Масштабирование

Преобразование масштабирования относительно начала координат имеет вид:

$$\begin{cases} x' = xS_x, \\ y' = yS_y, \\ z' = zS_z, \end{cases} \tag{2.31}$$

S_x, S_y, S_z — коэффициенты масштабирования по осям Ox, Oy и Oz соответственно.

Вращение

В трехмерном случае вместо преобразования поворота имеет место преобразование вращения вокруг координатной оси. Трем осям координат соответствуют три преобразования вращения. В результате каждого такого преобразования координата, соответствующая оси вращения, остается неизменной, и преобразование сводится к двумерному повороту относительно начала координат (см. рис. 2.24).

При совершении преобразования вращения вокруг некоторой оси отмеряем угол против часовой стрелки когда смотрим на начало координат с положительной полуоси данной оси.

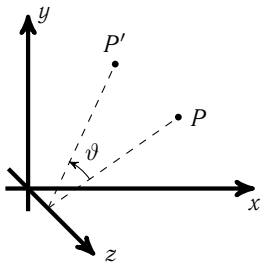


Рис. 2.24. Вращение вокруг оси Oz

Преобразование вращения относительно оси Ox начала координат против часовой стрелки на угол ϑ:

$$\begin{cases} x' = x, \\ y' = y\cos\vartheta - z\sin\vartheta, \\ z' = y\sin\vartheta + z\cos\vartheta. \end{cases} \tag{2.32}$$

Преобразование вращения относительно оси Oy начала координат против часовой стрелки на угол ϑ:

$$\begin{cases} x' = z\sin\vartheta + x\cos\vartheta, \\ y' = y, \\ z' = z\cos\vartheta - x\sin\vartheta. \end{cases} \tag{2.33}$$

Преобразование вращения относительно оси Oz начала координат против часовой стрелки на угол ϑ:

$$\begin{cases} x' = x\cos\vartheta - y\sin\vartheta, \\ y' = x\sin\vartheta + y\cos\vartheta, \\ z' = z. \end{cases} \tag{2.34}$$

Зеркальное отражение

В трехмерном случае преобразование зеркального отражения проводится относительно координатных плоскостей. Как и в двумерном случае преобразование зеркального отражения сводится к смене знака одной из координат каждой точки изображения.

Зеркальное отражение относительно плоскости xOy выражается соотношением:

$$\begin{cases} x' = x, \\ y' = y, \\ z' = -z; \end{cases} \tag{2.35}$$

относительно плоскости yOz:

$$\begin{cases} x' = -x, \\ y' = y, \\ z' = z; \end{cases} \tag{2.36}$$

и относительно плоскости xOz

$$\begin{cases} x' = x, \\ y' = -y, \\ z' = z. \end{cases} \tag{2.37}$$

2.6.3. Совмещение преобразований

Когда последовательность элементарных преобразований применяется к изображению, возможны два подхода к выполнению такой задачи. При первом походе получается последовательность изображений, где каждое последующее изображение в последовательности получено с помощью очередного элементарного преобразования из предыдущего. С другой стороны можно рассматривать задачу как получение результата из исходного изображения в результате применения единого сложного преобразования — комбинации элементарных преобразований. Такое сложное преобразование будем назвать «совмещенное преобразование».

Рассмотрим совмещение преобразований на примере преобразования поворота относительно заданной точки $A = (x_A, y_A)$ против часовой стрелки на угол ϑ. Получить такое преобразование можно совместив преобразования переноса и поворота относительно начала координат. Сначала совершим переход к новой системе координат с началом координат в точке A. То есть совершим преобразование переноса с величинами сдвига $-x_A$, $-y_A$. Теперь, когда точка A в начале координат, то необходимое преобразование — поворот относительно начала координат против часовой стрелки на угол ϑ, после чего необходимо вернуться к исходной системе координат совершив обратный перенос с величинами сдвига x_A, y_A. Таким образом должны совершить

последовательно три преобразования:

$$\begin{cases} x_1 = x - x_A, \\ y_1 = y - y_A, \end{cases}$$

$$\begin{cases} x_2 = x_1 \cos\vartheta - y_1 \sin\vartheta, \\ y_2 = x_1 \sin\vartheta + y_1 \cos\vartheta, \end{cases}$$

$$\begin{cases} x' = x_2 + x_A, \\ y' = y_2 + y_A. \end{cases}$$

Избавляясь от промежуточных величин x_1, y_1, x_2, y_2 получим формулы совмещенного преобразования:

$$\begin{cases} x' = (x - x_A)\cos\vartheta - (y - y_A)\sin\vartheta + x_A, \\ y' = (x - x_A)\sin\vartheta + (y - y_A)\cos\vartheta + y_A. \end{cases}$$

2.7. Матричные преобразования

2.7.1. Матричная форма

Когда будем представлять операции в матричной форме, координаты вектора будем записывать как вектор-столбец. Таким образом, когда у нас есть вектор $\bar{p} = (p_1, p_2)$, в матричной форме его будем представлять как

$$\bar{p} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}.$$

Тогда запись $\bar{p}^T$ будет обозначать вектор-строку с теми же элементами, что и в векторе $\bar{p}$.

Скалярное произведение двух векторов $\bar{p}$ и $\bar{q}$ в матричной форме запишется

$$\bar{p}^T \bar{q}.$$

Векторное произведение

$$\bar{p} \times \bar{q} = \begin{bmatrix} p_2 q_3 - p_3 q_2 \\ p_3 q_1 - p_1 q_3 \\ p_1 q_2 - p_2 q_1 \end{bmatrix}$$

может быть представлено в виде умножения матрицы на вектор

$$\bar{p} \times \bar{q} = [\bar{p}]_\times \bar{q},$$

где $[\bar{p}]_\times$ — кососимметричная матрица составленная из координат вектора $\bar{p}$:

$$[\bar{p}]_\times = \begin{bmatrix} 0 & -p_3 & p_2 \\ p_3 & 0 & -p_1 \\ -p_2 & p_1 & 0 \end{bmatrix}.$$

Пусть координаты точки (x, y) задаются вектором столбцом

$$\begin{bmatrix} x \\ y \end{bmatrix}.$$

В таком случае двумерные преобразования поворота, масштабирования и зеркального отражения легко представимы в матричной форме:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\vartheta & -\sin\vartheta \\ \sin\vartheta & \cos\vartheta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Для трехмерных преобразований имеют место формулы:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\vartheta & -\sin\vartheta \\ 0 & \sin\vartheta & \cos\vartheta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\vartheta & 0 & \sin\vartheta \\ 0 & 1 & 0 \\ -\sin\vartheta & 0 & \cos\vartheta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\vartheta & -\sin\vartheta & 0 \\ \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Перечисленные преобразования имеют одну и ту же форму:

$$\bar{p}' = M\bar{p}. \tag{2.38}$$

Где $\bar{p}$ и $\bar{p}'$ — радиус-векторы точек, M — матрица преобразования. Преимущество такого представления в том, что каждое преобразование задается в виде матрицы, а применение преобразования заключается в умножении соответствующей матрицы на столбец координат. При этом элементы матрицы преобразования зависят только от типа преобразования и его параметров и не зависят от координат преобразуемых точек.

Преимущество становится очевидным, когда в матричной форме представляется совмещенное преобразование. Предположим, что к точке с радиус-вектором $\bar{p}$ применяются последовательно преобразования, заданные матрицами M_1, M_2, …, M_k. Это можно представить в виде последовательных вычислений:

$$\begin{aligned} \bar{p}_1 &= M_1\bar{p}, \\ \bar{p}_2 &= M_2\bar{p}_1, \\ &\dots \\ \bar{p}' &= M_k\bar{p}_{k-1}. \end{aligned}$$

Если избавиться от вспомогательных имен $\bar{p}_2$, $\bar{p}_3$, …, $\bar{p}_{k-1}$, то получим

$$\bar{p}' = M_k \cdots M_2 M_1 \bar{p}.$$

Последняя формула подходит под шаблон (2.38), в котором

$$M = M_k \cdots M_2 M_1$$

— матрица совмещенного преобразования. То есть для получения матрицы совмещенного преобразования достаточно перемножить в обратном порядке матрицы его составляющих элементарных преобразований.

Но из этого ряда преобразований выпадает преобразование переноса. Это преобразование невозможно представить в форме (2.38). Представление преобразования переноса в форме отличной от (2.38) заметно усложняет как унификацию операций, так и получение соотношений для совмещенных преобразований.

Выход из сложившейся ситуации предлагает аппарат однородных координат, рассматриваемый в пунктах 2.7.4–2.7.5.

2.7.2. Двойственность трехмерного вращения

Взглянем еще раз на матричную форму записи преобразований трехмерного вращения.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\vartheta & -\sin\vartheta \\ 0 & \sin\vartheta & \cos\vartheta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\vartheta & 0 & \sin\vartheta \\ 0 & 1 & 0 \\ -\sin\vartheta & 0 & \cos\vartheta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\vartheta & -\sin\vartheta & 0 \\ \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Можно представить каждое из этих преобразований в форме

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_1'^T \\ \boldsymbol{e}_2'^T \\ \boldsymbol{e}_3'^T \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{2.39}$$

где $\boldsymbol{e}_i'^T$ обозначает вектор-строку — строку матрицы преобразования вращения. Например, для преобразования вращения вокруг оси Oz:

$$\boldsymbol{e}_1' = \begin{bmatrix} \cos\vartheta \\ -\sin\vartheta \\ 0 \end{bmatrix}, \quad \boldsymbol{e}_2' = \begin{bmatrix} \sin\vartheta \\ \cos\vartheta \\ 0 \end{bmatrix}, \quad \boldsymbol{e}_3' = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2.40}$$

Обратим внимание на то, что все $\boldsymbol{e}_i'$ — единичные вектора. Кроме того, эти векторы попарно перпендикулярны. Действительно, скалярное произведение пар векторов в (2.40):

$$\begin{aligned} \boldsymbol{e}_1'\boldsymbol{e}_2' &= \cos\vartheta\sin\vartheta - \sin\vartheta\cos\vartheta + 0\cdot 0 = 0, \\ \boldsymbol{e}_1'\boldsymbol{e}_3' &= 0\cdot\cos\vartheta - 0\cdot\sin\vartheta + 0\cdot 1 = 0, \\ \boldsymbol{e}_2'\boldsymbol{e}_3' &= 0\cdot\sin\vartheta + 0\cdot\cos\vartheta + 0\cdot 1 = 0. \end{aligned}$$

То есть можно воспринимать эти три вектора как систему координат.

Если исходная система координат у нас правая, то и система координат $\boldsymbol{e}_1'\boldsymbol{e}_2'\boldsymbol{e}_3'$ — тоже правая. Действительно, векторное произведение пар векторов в (2.40):

$$\boldsymbol{e}_1' \times \boldsymbol{e}_2' = \begin{bmatrix} 0\cdot(-\sin\vartheta) - 0\cdot\cos\vartheta \\ 0\cdot\sin\vartheta - 0\cdot\cos\vartheta \\ \cos\vartheta\cos\vartheta - (-\sin\vartheta)\sin\vartheta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \boldsymbol{e}_3'$$

$$\boldsymbol{e}_2' \times \boldsymbol{e}_3' = \begin{bmatrix} 1\cdot\cos\vartheta - 0\cdot 0 \\ 0\cdot 0 - 1\cdot\sin\vartheta \\ 0\cdot\sin\vartheta - 0\cdot\cos\vartheta \end{bmatrix} = \begin{bmatrix} \cos\vartheta \\ -\sin\vartheta \\ 0 \end{bmatrix} = \boldsymbol{e}_1'$$

$$\boldsymbol{e}'_3 \times \boldsymbol{e}'_1 = \begin{bmatrix} 0 \cdot 0 - 1 \cdot (-\sin\vartheta) \\ 1 \cdot \cos\vartheta - 0 \cdot 0 \\ 0 \cdot (-\sin\vartheta) - 0 \cdot \cos\vartheta \end{bmatrix} = \begin{bmatrix} \sin\vartheta \\ \cos\vartheta \\ 0 \end{bmatrix} = \boldsymbol{e}'_2$$

Матричную запись преобразования вращения (2.39) можно рассматривать как результат трех скалярных произведений:

$$\begin{aligned} x' &= \boldsymbol{e}_1'^T \bar{p}, \\ y' &= \boldsymbol{e}_2'^T \bar{p}, \\ z' &= \boldsymbol{e}_3'^T \bar{p}. \end{aligned}$$

Так как каждый вектор $\boldsymbol{e}'_i$ — единичный, то результатом каждого скалярного произведения является длина проекции радиус-вектора $\bar{p}$ на вектор $\boldsymbol{e}'_i$, т. е. i-я координата радиус-вектора $\bar{p}$ в системе координат $\boldsymbol{e}'_1\boldsymbol{e}'_2\boldsymbol{e}'_3$.

Таким образом, построить матрицу поворота можно взяв в качестве ее строк компоненты единичных направляющих векторов осей повернутой системы координат.

2.7.3. Вращение относительно произвольного вектора. Формула Родригеса

Будем рассматривать вращение относительно произвольной оси, проходящей через начало координат. Будем предполагать, что ось вращения задается единичным вектором $\bar{n}$, угол вращения ϑ отсчитывается против часовой стрелки, если смотреть против направления $\bar{n}$, а вращаемая точка P задана своим радиус-вектором $\bar{p}$ (см. рис. 2.25). Выведем соотношение для получения результата поворота — координат точки P' (радиус-вектора $\bar{p}'$).

Представим вектор $\bar{p}$ в виде суммы двух двух векторов

$$\bar{p} = \bar{p}_{\parallel} + \bar{p}_{\perp},$$

где вектор $\bar{p}_{\parallel}$ параллелен вектору $\bar{n}$, а вектор $\bar{p}_{\perp}$ — перпендикулярен вектору $\bar{n}$. Так как $\bar{n}$ — единичный, вектор $\bar{p}_{\parallel}$ легко найти как результат скалярного произведения, умноженный на вектор $\bar{n}$:

$$\bar{p}_{\parallel} = (\bar{p}\bar{n})\bar{n}.$$

В свою очередь

$$\bar{p}_{\perp} = \bar{p} - \bar{p}_{\parallel}.$$

Если теперь осуществить поворот относительно оси $\bar{n}$ вектора $\bar{p}_{\perp}$ на угол ϑ, добавив к результату вектор $\bar{p}_{\parallel}$ получим координаты искомого вектора. Чтобы осуществить

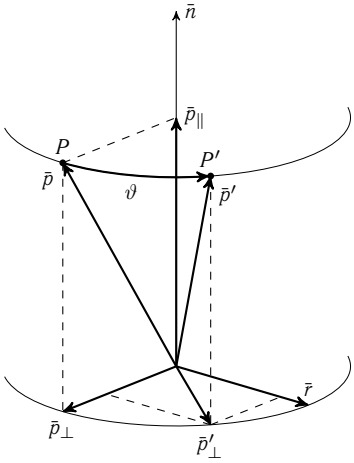


Рис. 2.25. Вращение вокруг произвольной оси

этот поворот, найдем координаты вектора $\bar{r}$, перпендикулярного векторам $\bar{p}$ и $\bar{n}$. Это можно сделать с помощью векторного произведения:

$$\bar{r} = \bar{n} \times \bar{p}.$$

Из свойств векторного произведения следует, что длины векторов $\bar{r}$ и $\bar{p}_{\perp}$ совпадают. Вследствие этого, координаты вектора $\bar{p}'_{\perp}$ (вектор $\bar{p}_{\perp}$ после поворота на угол ϑ) равны

$$\bar{p}'_{\perp} = \bar{p}_{\perp} \cos\vartheta + \bar{r} \sin\vartheta,$$

откуда получаем результат:

$$\bar{p}' = \bar{p}'_{\perp} + \bar{p}_{\parallel}.$$

Соберем все выкладки в единую формулу

$$\bar{p}' = (\bar{p} - (\bar{p}\bar{n})\bar{n}) \cos\vartheta + (\bar{n} \times \bar{p}) \sin\vartheta + (\bar{p}\bar{n})\bar{n}.$$

Раскроем первые скобки и представим векторное произведение в матричной форме

$$\bar{p}' = \bar{p} \cos\vartheta - (\bar{p}\bar{n})\bar{n} \cos\vartheta + [\bar{n}]_{\times}\bar{p} \sin\vartheta + (\bar{p}\bar{n})\bar{n}.$$

Сгруппируем второе и последнее слагаемые:

$$\bar{p}' = \bar{p} \cos\vartheta + (\bar{p}\bar{n})\bar{n}(1 - \cos\vartheta) + [\bar{n}]_{\times}\bar{p} \sin\vartheta.$$

Первое слагаемое представим в матричной форме

$$\bar{p}' = (E\cos\vartheta)\bar{p} + (\bar{p}\bar{n})\bar{n}(1-\cos\vartheta) + [\bar{n}]_\times \bar{p}\sin\vartheta,$$

где E — единичная матрица. Исходя из того, что

$$(\bar{p}\bar{n})\bar{n} = (\bar{n}\bar{p})\bar{n} = \bar{n}\bar{n}^T\bar{p}$$

получим

$$\bar{p}' = (E\cos\vartheta)\bar{p} + \bar{n}\bar{n}^T\bar{p}(1-\cos\vartheta) + [\bar{n}]_\times \bar{p}\sin\vartheta,$$

Последнее равенство можно представить в виде

$$\bar{p}' = M\bar{p},$$

где матрица M вычисляется по формуле

$$M = E\cos\vartheta + \bar{n}\bar{n}^T(1-\cos\vartheta) + [\bar{n}]_\times \sin\vartheta. \tag{2.41}$$

Можно показать, что

$$\bar{n}\bar{n}^T = [\bar{n}]_\times^2 + E. \tag{2.42}$$

Действительно,

$$[\bar{n}]_\times^2 = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix}\begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix} = \begin{bmatrix} -n_2^2-n_3^2 & n_1n_2 & n_1n_3 \\ n_1n_2 & -n_1^2-n_3^2 & n_2n_3 \\ n_1n_3 & n_2n_3 & -n_1^2-n_2^2 \end{bmatrix}.$$

С другой стороны

$$\bar{n}\bar{n}^T = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix}\begin{bmatrix} n_1 & n_2 & n_3 \end{bmatrix} = \begin{bmatrix} n_1^2 & n_1n_2 & n_1n_3 \\ n_1n_2 & n_2^2 & n_2n_3 \\ n_1n_3 & n_2n_3 & n_3^2 \end{bmatrix}.$$

Так как $\bar{n}$ — единичный вектор, имеет место соотношение

$$n_1^2 + n_2^2 + n_3^2 = 1,$$

откуда

$$\begin{aligned} n_1^2 &= 1 - n_2^2 - n_3^2, \\ n_2^2 &= 1 - n_1^2 - n_3^2, \\ n_3^2 &= 1 - n_1^2 - n_2^2. \end{aligned}$$

Таким образом

$$\bar{n}\bar{n}^T = \begin{bmatrix} 1-n_2^2-n_3^2 & n_1n_2 & n_1n_3 \\ n_1n_2 & 1-n_1^2-n_3^2 & n_2n_3 \\ n_1n_3 & n_2n_3 & 1-n_1^2-n_2^2 \end{bmatrix} = [\bar{n}]_\times^2 + E.$$

Подставим (2.42) в (2.41). Получим

$$M = E\cos\vartheta + ([\bar{n}]_\times^2 + E)(1-\cos\vartheta) + [\bar{n}]_\times \sin\vartheta.$$

Раскроем скобки во втором слагаемом

$$M = E\cos\vartheta + [\bar{n}]_\times^2 + E - [\bar{n}]_\times^2\cos\vartheta - E\cos\vartheta + [\bar{n}]_\times\sin\vartheta.$$

и приведем подобные

$$M = E + [\bar{n}]_\times \sin\vartheta + [\bar{n}]_\times^2(1-\cos\vartheta). \tag{2.43}$$

Соотношение (2.43) называется формулой Родригеса вращения вокруг произвольной оси $\bar{n}$.

Можно расписать эту формулу через координаты вектора $\bar{n}$:

$$M = \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} + \begin{bmatrix} 0&-n_3&n_2\\n_3&0&-n_1\\-n_2&n_1&0 \end{bmatrix}\sin\vartheta + \begin{bmatrix} n_1^2-1&n_1n_2&n_1n_3\\n_1n_2&n_2^2-1&n_2n_3\\n_1n_3&n_2n_3&n_3^2-1 \end{bmatrix}(1-\cos\vartheta)$$

или без последнего преобразования

$$M = \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix}\cos\vartheta + \begin{bmatrix} n_1^2&n_1n_2&n_1n_3\\n_1n_2&n_2^2&n_2n_3\\n_1n_3&n_2n_3&n_3^2 \end{bmatrix}(1-\cos\vartheta) + \begin{bmatrix} 0&-n_3&n_2\\n_3&0&-n_1\\-n_2&n_1&0 \end{bmatrix}\sin\vartheta.$$

Формула Родригеса является универсальной: легко проверить, что подставив вместо $\bar{n}$ любой из векторов $\boldsymbol{e}_i$ получим стандартную матрицу вращения относительно соответствующей оси.

2.7.4. Однородные координаты

Однородные координаты точки, прямой и т. д., — координаты, обладающие тем свойством, что определяемый ими объект не меняется, когда все координаты умножаются на одно и то же число.

Существуют различные способы определения однородных координат. Мы будем исходить из задачи унифицированного представления координат точек в пространстве.

Пусть заданы два действительных числа — a и α. Рассмотрим их отношение a/α. Зафиксируем значение a, и будем варьировать значение α. При уменьшении α, значение a/α будет увеличиваться. Заметим, что если α стремится к нулю, то a/α стремится к бесконечности. Таким образом, чтобы включить в рассмотрение понятие бесконечности, для представления значения ν используется пара чисел (a, α), таких, что $\nu = a/\alpha$. Если $\alpha \neq 0$, значение ν в точности равно a/α. В противном случае $\nu = a/0$, т.е. равно бесконечности.

Таким образом, координаты двумерной точки $v = (x, y)$ можно представить через координаты

$$(\chi, \gamma, \alpha),$$

где $\chi = \alpha x$, $\gamma = \alpha y$.

Если взять произвольную тройку (χ, γ, α), то при $\alpha \neq 0$ эти координаты описывают точку с конечными координатами $(\chi/\alpha, \gamma/\alpha)$, а при $\alpha = 0$ — точку, бесконечно удаленную в направлении (χ, γ).

Рассмотрим двумерную плоскость, некоторую точку (x, y) на ней и заданную функцию $f(x, y)$. Если заменить x и y на χ/α и γ/α, то выражение $f(x, y) = 0$ заменится на $f(\chi/\alpha, \gamma/\alpha) = 0$. Если $f(x, y)$ — многочлен, то его умножение на α^n (n — степень многочлена) уберет все знаменатели.

Например, пусть имеется прямая на плоскости, заданная своим уравнением

$$Ax + By + C = 0.$$

Замена x и y на χ/α и γ/α дает $A\chi/\alpha + B\gamma/\alpha + C = 0$. Умножая на обе части равенства на α, получаем

$$A\chi + B\gamma + C\alpha = 0. \tag{2.44}$$

Равенство (2.44) задает точки прямой в однородных координатах. Это уравнение будем называть уравнением прямой в однородных координатах, коэффициенты этого уравнения A, B и C — однородными координатами прямой (действительно, для одной и той же прямой эти значения определяются с точностью до общего ненулевого множителя).

В дальнейшем однородные координаты точки будем представлять в виде вектора-столбца

$$\begin{bmatrix} \chi \\ \gamma \\ \alpha \end{bmatrix},$$

однородные координаты двумерной прямой — в виде вектора-солбца

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix}.$$

Тогда уравнение прямой (2.44) можно представить в матричной форме:

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix}^T \begin{bmatrix} \chi \\ \gamma \\ \alpha \end{bmatrix} = 0.$$

Аналогичные рассуждения можно провести при введении однородных координат трехмерных точек. В этом случае равенство

$$A\chi + B\gamma + C\zeta + D\alpha = 0.$$

представляет собой уравнение плоскости, которое можно представить в матричной форме

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}^T \begin{bmatrix} \chi \\ \gamma \\ \zeta \\ \alpha \end{bmatrix} = 0,$$

где вектор-столбец

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}$$

— однородные координаты плоскости.

Итак, приведем более формальное определение.

Однородными координатами точки $P \in R^n$ называется вектор столбец

$$\begin{bmatrix} \chi_1 & \chi_2 & \dots & \chi_n & \chi_{n+1} \end{bmatrix}^T \in R^{n+1},$$

среди элементов которого хотя бы один элемент χ_i должен быть отличен от нуля. Если $\chi_{n+1} = 0$, то точка P является бесконечно удаленной.

Евклидовыми координатами точки P будут являться координаты

$$\left(\frac{\chi_1}{\chi_{n+1}}, \frac{\chi_2}{\chi_{n+1}}, .., \frac{\chi_n}{\chi_{n+1}} \right).$$

Преобразование из однородных координат в евклидовы однозначно; преобразование из евклидовых координат в однородные — нет.

Однородными координатами прямой в двумерном пространстве называются координаты

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix},$$

одновременно не равные нулю. Если $A = B = 0$, то прямая является бесконечно удаленной (т. к. такой прямой принадлежат только бесконечно удаленные точки).

Однородными координатами плоскости называются координаты

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix},$$

одновременно не равные нулю. Если $A = B = C = 0$, то плоскость является бесконечно удаленной.

2.7.5. Геометрические преобразования в однородных координатах

В общем случае преобразование в трехмерном пространстве имеет вид:

$$\begin{cases} x' = a_1 x + b_1 y + c_1 z + d_1, \\ y' = a_2 x + b_2 y + c_2 z + d_2, \\ z' = a_3 x + b_3 y + c_3 z + d_3. \end{cases} \tag{2.45}$$

Если перейти от обычных координат (x, y, z) к однородным $(\chi, \gamma, \zeta, \alpha)$, то преобразование (2.45) запишется в виде

$$\begin{cases} \chi' = a_1 \chi + b_1 \gamma + c_1 \zeta + d_1 \alpha, \\ \gamma' = a_2 \chi + b_2 \gamma + c_2 \zeta + d_2 \alpha, \\ \zeta' = a_3 \chi + b_3 \gamma + c_3 \zeta + d_3 \alpha, \\ \alpha' = \alpha. \end{cases}$$

Последнюю систему равенств можно записать в матричной форме (принимая во внимание тот факт, что однородные координаты точки представляем в виде вектора-столбца):

$$\begin{bmatrix} \chi' \\ \gamma' \\ \zeta' \\ \alpha' \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \chi \\ \gamma \\ \zeta \\ \alpha \end{bmatrix}. \tag{2.46}$$

В двумерном случае аналогичными рассуждениями получаем общий вид преобразования в матричной форме:

$$\begin{bmatrix} \chi' \\ \gamma' \\ \alpha' \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \chi \\ \gamma \\ \alpha \end{bmatrix}. \tag{2.47}$$

Заметим, что матрицы преобразований не зависят от множителя α перехода к однородной системе координат, а содержат только коэффициенты преобразований.

Зная матрицу преобразования можно легко получить матрицу обратного преобразования: если M — матрица преобразования, то M^{-1} — матрица обратного преобразования.

Пользуясь шаблонами (2.47) и (2.46) можем переписать формулы преобразований в однородных координатах.

Двумерные преобразования

Из (2.24) и (2.47) преобразование переноса запишется в виде:

$$\begin{bmatrix} \chi' \\ \gamma' \\ \alpha' \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \chi \\ \gamma \\ \alpha \end{bmatrix}.$$

Из (2.25) и (2.47) преобразование масштабирования запишется в виде:

$$\begin{bmatrix} \chi' \\ \gamma' \\ \alpha' \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \chi \\ \gamma \\ \alpha \end{bmatrix}.$$

Из (2.26) и (2.47) преобразование поворота относительно начала координат на угол ϑ против часовой стрелки запишется в виде:

$$\begin{bmatrix} \chi' \\ \gamma' \\ \alpha' \end{bmatrix} = \begin{bmatrix} \cos\vartheta & -\sin\vartheta & 0 \\ \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \chi \\ \gamma \\ \alpha \end{bmatrix}.$$

Из (2.28), (2.29) и (2.47) преобразования зеркального отражения относительно осей запишутся в виде:

$$\begin{bmatrix} \chi' \\ \gamma' \\ \alpha' \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \chi \\ \gamma \\ \alpha \end{bmatrix}$$

и

$$\begin{bmatrix} \chi' \\ \gamma' \\ \alpha' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \chi \\ \gamma \\ \alpha \end{bmatrix}.$$

Таким образом все элементарные двумерные преобразования представлены в одной и той же форме

$$\bar{p}' = M\bar{p}.$$

Следовательно можно представить в такой же форме и любое совмещенное преобразование. Рассмотрим, например, еще раз преобразование поворота относительно произвольной точки, описанное в п. 2.6.3.

Последовательность преобразований можно представить в виде последовательного домножения полученного результат на матрицу очередного преобразования. Тогда получим

$$\begin{bmatrix}\chi'\\ \gamma'\\ \alpha'\end{bmatrix} = \begin{bmatrix}1 & 0 & x_A\\ 0 & 1 & y_A\\ 0 & 0 & 1\end{bmatrix}\left(\begin{bmatrix}\cos\vartheta & -\sin\vartheta & 0\\ \sin\vartheta & \cos\vartheta & 0\\ 0 & 0 & 1\end{bmatrix}\left(\begin{bmatrix}1 & 0 & -x_A\\ 0 & 1 & -y_A\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\chi\\ \gamma\\ \alpha\end{bmatrix}\right)\right).$$

Раскроем скобки и перегруппируем, получим

$$\begin{bmatrix}\chi'\\ \gamma'\\ \alpha'\end{bmatrix} = \left(\begin{bmatrix}1 & 0 & x_A\\ 0 & 1 & y_A\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\vartheta & -\sin\vartheta & 0\\ \sin\vartheta & \cos\vartheta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -x_A\\ 0 & 1 & -y_A\\ 0 & 0 & 1\end{bmatrix}\right)\begin{bmatrix}\chi\\ \gamma\\ \alpha\end{bmatrix}.$$

Результат выражения в круглых скобках — матрица совмещенного преобразования. Перемножив матрицы получим

$$\begin{bmatrix}\chi'\\ \gamma'\\ \alpha'\end{bmatrix} = \begin{bmatrix}\cos\vartheta & -\sin\vartheta & x_A(1-\cos\vartheta)+y_A\sin\vartheta\\ \sin\vartheta & \cos\vartheta & y_A(1-\cos\vartheta)-x_A\sin\vartheta\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\chi\\ \gamma\\ \alpha\end{bmatrix}.$$

где

$$\begin{bmatrix}\cos\vartheta & -\sin\vartheta & x_A(1-\cos\vartheta)+y_A\sin\vartheta\\ \sin\vartheta & \cos\vartheta & y_A(1-\cos\vartheta)-x_A\sin\vartheta\\ 0 & 0 & 1\end{bmatrix}$$

— матрица совмещенного преобразования.

Трехмерные преобразования

Из (2.30) и (2.46) преобразование переноса запишется в виде:

$$\begin{bmatrix}\chi'\\ \gamma'\\ \zeta'\\ \alpha'\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & T_x\\ 0 & 1 & 0 & T_y\\ 0 & 0 & 1 & T_z\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\chi\\ \gamma\\ \zeta\\ \alpha\end{bmatrix}.$$

Из (2.31) и (2.46) преобразование масштабирования запишется в виде:

$$\begin{bmatrix}\chi'\\ \gamma'\\ \zeta'\\ \alpha'\end{bmatrix} = \begin{bmatrix}S_x & 0 & 0 & 0\\ 0 & S_y & 0 & 0\\ 0 & 0 & S_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\chi\\ \gamma\\ \zeta\\ \alpha\end{bmatrix}.$$

Из (2.35)–(2.37) и (2.46) преобразования зеркального отражения относительно координатных плоскостей запишутся в виде:

$$\begin{bmatrix}\chi'\\ \gamma'\\ \zeta'\\ \alpha'\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & -1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\chi\\ \gamma\\ \zeta\\ \alpha\end{bmatrix},$$

$$\begin{bmatrix}\chi'\\ \gamma'\\ \zeta'\\ \alpha'\end{bmatrix} = \begin{bmatrix}-1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\chi\\ \gamma\\ \zeta\\ \alpha\end{bmatrix}$$

и

$$\begin{bmatrix}\chi'\\ \gamma'\\ \zeta'\\ \alpha'\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & -1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\chi\\ \gamma\\ \zeta\\ \alpha\end{bmatrix}.$$

Из (2.43) и (2.46) преобразование вращения относительно оси $\bar{n}$ на угол ϑ против часовой стрелки запишется в виде:

$$\begin{bmatrix}\chi'\\ \gamma'\\ \zeta'\\ \alpha'\end{bmatrix} = \begin{bmatrix}R_{11} & R_{12} & R_{13} & 0\\ R_{21} & R_{22} & R_{23} & 0\\ R_{31} & R_{32} & R_{33} & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\chi\\ \gamma\\ \zeta\\ \alpha\end{bmatrix},$$

где R_{ij} — элементы матрицы вращения из (2.43).

Преобразование прямых/плоскостей

Пусть заданы однородные координаты прямой в двумерном пространстве

$$\begin{bmatrix}A\\ B\\ C\end{bmatrix}.$$

Для точек, принадлежащих прямой, имеет место

$$\begin{bmatrix}A\\ B\\ C\end{bmatrix}^T \begin{bmatrix}\chi\\ \gamma\\ \alpha\end{bmatrix} = 0.$$

В результате проведения преобразования точки должны остаться точками прямой. Пусть преобразование для точек задается матрицей M.

$$\begin{bmatrix}\chi'\\ \gamma'\\ \alpha'\end{bmatrix} = M \begin{bmatrix}\chi\\ \gamma\\ \alpha\end{bmatrix}. \tag{2.48}$$

Предположим, что координаты прямой после преобразования —

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix}.$$

Тогда должно выполняться

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix}^T \begin{bmatrix} \chi' \\ \gamma' \\ \alpha' \end{bmatrix} = 0.$$

В последнее равенство подставим (2.48):

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix}^T M \begin{bmatrix} \chi \\ \gamma \\ \alpha \end{bmatrix} = 0,$$

откуда вытекает

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix}^T = \begin{bmatrix} A \\ B \\ C \end{bmatrix}^T M^{-1},$$

и следовательно

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \left(M^{-1}\right)^T \begin{bmatrix} A \\ B \\ C \end{bmatrix}.$$

По аналогии, получаем соотношение для трехмерного преобразования плоскости. Если преобразование для точек задается матрицей M, то это же преобразование для координат плоскости задается матрицей $\left(M^{-1}\right)^T$.

3. Основные понятия компьютерной графики

3.1. Векторные и растровые изображения

Векторное изображение — это изображение, состоящее из графических примитивов, таких как точки, линии, сплайны и многоугольники и др.

Для того, чтобы некоторое графическое устройство строило векторные изображения необходимо, чтобы оно умело строить графические примитивы (например провести непрерывный отрезок от точки A до точки B). Существует узкий класс устройств, ориентированных исключительно на отображение векторных данных. К ним относятся мониторы с векторной развёрткой, графопостроители, а также некоторые типы лазерных проекторов.

Большинство современных компьютерных устройств отображают информацию в растровом формате. Растровое устройство можно рассматривать как матрицу дискретных ячеек (точек, пикселов), каждая из которых может быть закрашена (подсвечена). Таким образом, оно является точечно-рисующим устройством. Невозможно, за исключением специальных случаев, непосредственно нарисовать отрезок прямой из одной адресуемой точки или пиксела в матрице в другую адресуемую точку или пиксел. Отрезок можно лишь аппроксимировать последовательностями точек (пикселов), близко лежащих к реальной траектории отрезка. Эту идею иллюстрирует рис. 3.2.

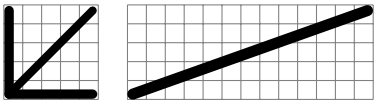


Рис. 3.1. Изображение отрезка на векторном устройстве.

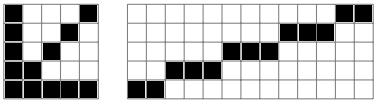


Рис. 3.2. Аппроксимация отрезка на растровом устройстве.

Преобразование растрового изображения в векторный формат называется векторизацией, а преобразование векторного формата в растровый — растеризацией. При

выводе векторного изображения на растровом устройстве растеризация обычно проводится преобразователями, программными или аппаратными, встроенными в видеокарту.

3.2. Сцена. Системы координат

Совокупность естественных или абстрактных объектов, предназначенных для построения изображения с помощью компьютера, будем называть сценой. В зависимости от размерности пространства, в котором рассматриваются объекты, сцена бывает трехмерной или двумерной.

Построенное изображение сцены будем называть образом сцены. Образ двумерной сцены называют двумерным образом, трехмерной сцены — трехмерным образом.

Часть сцены, для которой получен образ называют прообразом.

Для описания объектов сцены с ней связывают систему координат.

Декартова система координат, в которой заданы объекты сцены называют мировой системой координат. Для двумерной сцены такую систему координат еще называют картинной (или системой координат картинной плоскости), а для трехмерной сцены — объектной. Объектная система координат может быть как левой, так и правой.

Как двумерный, так и, в большинстве случаев, трехмерный образ представляет собой двумерное прямоугольное изображение. Для описания расположения объектов образа используют двумерную ДСК, связанную с портом вывода: экраном компьютера, окном на экране, листом печати и т. п. Такую ДСК называют системой координат экрана (СКЭ). Обычно, при выводе в окно на экране компьютера, СКЭ представляет собой ДСК, начало координат которой обычно располагается в верхнем левом углу, ось Oy направлена вертикально вниз, а ось Ox направлена вправо. Обычно, значения координат в СКЭ целочисленные.

Для двумерного случая прямоугольнику образа соответствует некоторая прямоугольная область прообраза — кадр. Построение двумерного образа заключается в совмещении кадра с прямоугольником образа. Эту операцию будем называть операцией кадрирования.

Пусть в картинной системе координат выделен кадр со следующими параметрами (см. рис. 3.3): V_x, V_y — размеры кадра по горизонтали и вертикали соответственно; (V_{cx}, V_{cy}) — координаты нижнего левого угла кадра. Не теряя общности предположим, что стороны кадра параллельны осям координат — в противном случае мож-

но перейти к новой системе координат с помощью одного преобразования поворота.

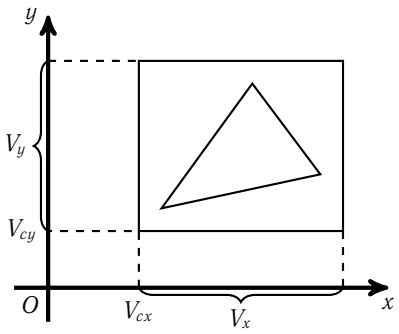
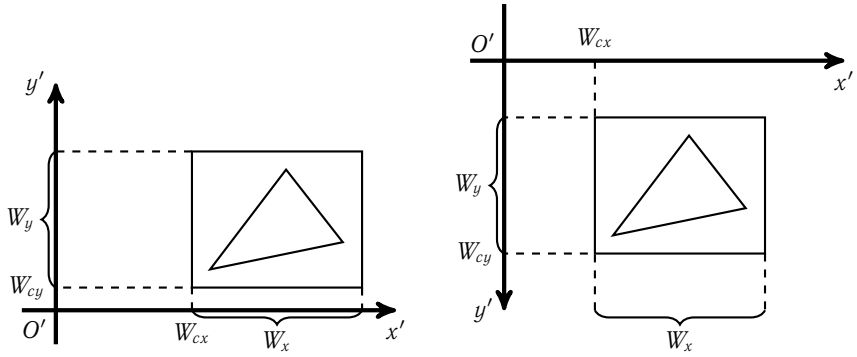


Рис. 3.3. Кадр в картинной плоскости

Пусть в СКЭ задано прямоугольное окно, с которым необходимо совместить изображение попавшее в кадр. Пусть параметры этого окна следующие (см. рис. 3.4, *а*): W_x, W_y — размеры окна по горизонтали и вертикали соответственно; (W_{cx}, W_{cy}) — координаты нижнего левого угла окна.

Если и в картинной системе координат и в СКЭ ось Oy направлена вверх, то такое совмещение можно выразить преобразованием:

$$\begin{cases} x' = \dfrac{x - V_{cx}}{V_x} W_x + W_{cx}, \\ y' = \dfrac{y - V_{cy}}{V_y} W_y + W_{cy}, \end{cases}$$



(*а*) (*б*)

Рис. 3.4. Кадр в экранной плоскости: (*а*) в правой системе координат, (*б*) в левой системе координат

или в матричной форме

$$\begin{bmatrix}\chi'\\ \gamma'\\ \alpha'\end{bmatrix} = \begin{bmatrix}W_x/V_x & 0 & W_{cx} - V_{cx}W_x/V_x\\ 0 & W_y/V_y & W_{cy} - V_{cy}W_y/V_y\\ 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\chi\\ \gamma\\ \alpha\end{bmatrix}. \tag{3.1}$$

Если и в картинной системе координат ось Oy направлена вверх, а в СКЭ — вниз (см. рис. 3.4, *б*), то такое совмещение можно выразить преобразованием:

$$\begin{cases} x' = \dfrac{x - V_{cx}}{V_x} W_x + W_{cx} \\ y' = W_{cy} - \dfrac{y - V_{cy}}{V_y} W_y \end{cases}$$

или в матричной форме

$$\begin{bmatrix}\chi'\\ \gamma'\\ \alpha'\end{bmatrix} = \begin{bmatrix}W_x/V_x & 0 & W_{cx} - V_{cx}W_x/V_x\\ 0 & -W_y/V_y & W_{cy} + V_{cy}W_y/V_y\\ 0 & 0 & 1\end{bmatrix} \begin{bmatrix}\chi\\ \gamma\\ \alpha\end{bmatrix}. \tag{3.2}$$

Соотношения (3.1) и (3.2) задают преобразование кадрирования.

Построение двумерного образа может заключаться в том, что для каждой точки кадра в картинной плоскости выполняется операция кадрирования и, тем самым, вычисляется положение этой точки в окне в СКЭ. Или наоборот (в частности, при растеризации изображения), для каждой точки растра в окне в СКЭ выполняется обратное преобразование для операции кадрирования и, тем самым, вычисляется положение той точки в картинной плоскости, которая должна быть отображена в исходной точке системы координат экрана.

Построение трехмерного образа происходит не так элементарно.

Трехмерный образ есть изображение трехмерной сцены с некоторой точки зрения. Точку пространства, из которой наблюдается трехмерная сцена будем называть точкой наблюдения. Направление, в котором происходит наблюдение будем задавать с помощью вектора — вектора наблюдения. Субъекта наблюдения будем называть наблюдателем. Будем предполагать, что точка наблюдения находится в одном из глаз наблюдателя.

Пусть наблюдатель смотрит на трехмерную сцену через прямоугольное окно, причем вектор наблюдения перпендикулярен плоскости этого окна, а взгляд наблюдателя устремлен точно в его центр (рис. 3.5). Такое окно будем называть окном наблюдения. Считаем, что между наблюдателем и окном наблюдения не содержится никаких объектов. Пусть трехмерная сцена, содержащая объекты, видимые для наблюдателя,

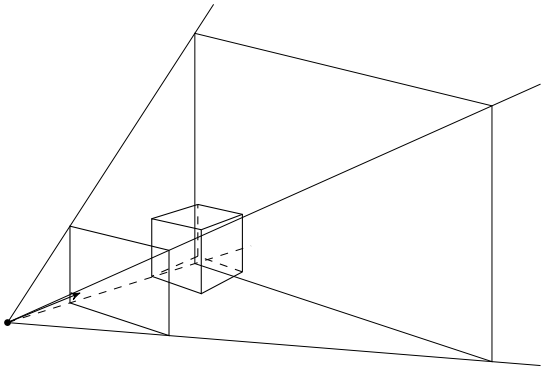


Рис. 3.5. Трехмерная сцена

ограничена сзади плоскостью, параллельной окну наблюдения, т. е. никакие объекты, лежащие за этой плоскостью не являются видимыми для наблюдателя.

Таким образом прообраз трехмерного изображения ограничен усеченной пирамидой, основания которой образованы окном наблюдения и плоскостью, ограничивающей дальность обзора, а боковые грани опираются на стороны окна наблюдения (рис. 3.6). Внутрь пирамиды видимости попадают те части объектов трехмерной сцены, которые

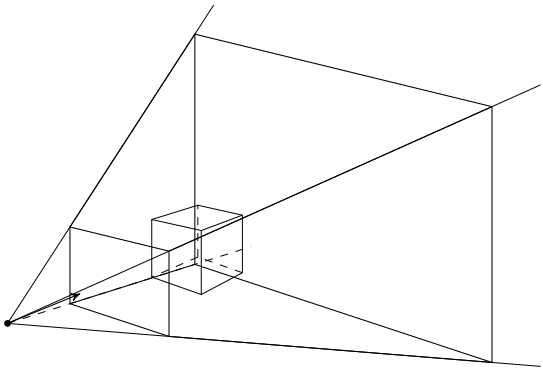


Рис. 3.6. Пирамида видимости

потенциально может видеть наблюдатель через окно наблюдения.

Можно представить, что вместо трехмерной сцены наблюдатель видит на плоскости окна наблюдения её двумерное изображение (как если бы то, что мы видим за окном, было, на самом деле, нарисовано на окне). Это изображение и будет представ-

лять кадр в трехмерном случае.

С наблюдателем связывают специальную систему координат — систему координат наблюдателя (СКН) (см. рис. 3.7). СКН — правая ДСК, в которой начало координат

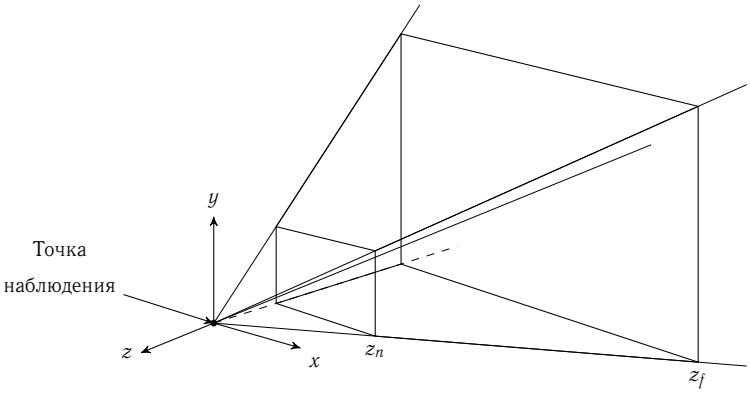


Рис. 3.7. Система координат наблюдателя

находится в точке наблюдения, ось Oz параллельна вектору наблюдения, но направлена в противоположную сторону — на наблюдателя, ось Ox направлена вправо от направления наблюдения, а ось Oy — вверх. Тогда ось Oz проходит перпендикулярно плоскости окна наблюдения через его центр. Предполагаем, что стороны окна наблюдения параллельны осям Ox и Oy.

Обозначим V — матрицу преобразования мировой системы координат в систему координат наблюдателя. Элементы матрицы V можно вычислить, зная координаты точки наблюдения и направление наблюдения (преобразование сведется к нескольким вращениям и одному переносу).

Переход от мировой системы координат к системе координат наблюдателя

Пусть задана трехмерная сцена в мировой системе координат (см. рисунок 3.8). Будем предполагать, что мировая система координат — правая. Кроме того, пусть заданы

- точка наблюдения $S = (x_s, y_s, z_s)$,
- точка $P = (x_p, y_p, z_p)$ на которую направлен вектор наблюдения,

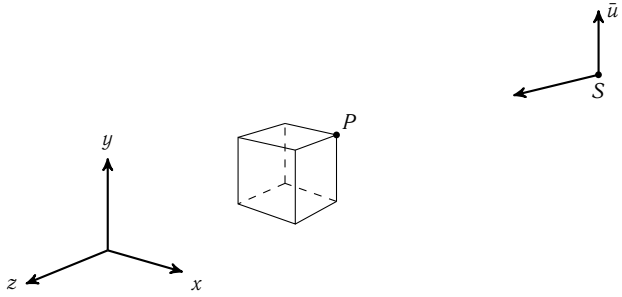


Рис. 3.8. Трехмерная сцена в мировой системе координат

- вектор $\bar{u} = (u_x, u_y, u_z)$ указывающий направление вверх (возможно, примерное направление вверх).

Система координат наблюдателя — правая декартова система координат, начало которой лежит в точке наблюдения, а ось Oz параллельна вектору наблюдения. Таким образом, для перехода от мировой системы координат к системе координат наблюдателя необходимо:

1. перенести начало координат в точку наблюдения;
2. организовать вращение осей координат таким образом, чтобы ось Oz была направлена в сторону противоположную вектору наблюдения, а ось Oy была направлена вверх.

Первое преобразование простое: достаточно к однородным координатам каждой точки применить преобразование переноса с коэффициентами $T_x = -x_s$, $T_y = -y_s$, $T_z = -z_s$, т. е. необходимо выполнить преобразование заданное матрицей

$$R_1 = \begin{bmatrix} 1 & 0 & 0 & -x_s \\ 0 & 1 & 0 & -y_s \\ 0 & 0 & 1 & -z_s \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Для второго преобразования сначала найдем направляющие векторы $\boldsymbol{e}'_1$, $\boldsymbol{e}'_2$, $\boldsymbol{e}'_3$ для осей системы координат наблюдателя в мировой системе координат.

Направляющий вектор для оси Oz должен быть направлен от точки P к точке S. Следовательно его направление совпадает с направлением вектора $\bar{v}_{PS} = \bar{s} - \bar{p}$, где $\bar{s}$ и $\bar{p}$ — радиус-векторы точек S и P, соответственно. Таким образом

$$\boldsymbol{e}'_3 = \frac{\bar{v}_{PS}}{|\bar{v}_{PS}|} = \frac{\bar{s} - \bar{p}}{|\bar{s} - \bar{p}|}.$$

Так как вектор $\bar{u}$ указывает примерное направление оси Oy правой системы координат, то если векторно умножить его на полученный вектор $\boldsymbol{e}'_3$, получим вектор, направление которого совпадает с направлением оси Ox

$$\boldsymbol{e}'_1 = \frac{\bar{u} \times \boldsymbol{e}'_3}{|\bar{u}|}.$$

В правой части равенства присутствует деление на длину вектора $\bar{u}$ для нормализации.

Наконец, чтобы получить направляющий вектор для оси Oy достаточно векторно умножить $\boldsymbol{e}'_3$ на $\boldsymbol{e}'_1$.

$$\boldsymbol{e}'_2 = \boldsymbol{e}'_3 \times \boldsymbol{e}'_1.$$

Теперь можно получить матрицу вращения системы координат, совместив в ней векторы $\boldsymbol{e}'_1$, $\boldsymbol{e}'_2$, $\boldsymbol{e}'_3$ в качестве строк:

$$R_2 = \begin{bmatrix} & \boldsymbol{e}'_1 & & 0 \\ & \boldsymbol{e}'_2 & & 0 \\ & \boldsymbol{e}'_3 & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Таким образом, для перехода из мировой системы координат в систему координат наблюдателя необходимо для каждой точки $(\chi, \gamma, \zeta, \alpha)$ выполнить преобразование V, где

$$V = R_2 \cdot R_1$$

Матрицу V полученную таким способом назовем $LookAt(S, P, \bar{u})$.

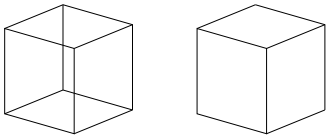
3.3. Классификация изображений

Различают проволочные, контурные и полутоновые трехмерные изображения.

При построении проволочного изображения предполагается, что объекты — многогранники и они представляются набором ребер (см. рис. 3.9, *а*). В образе отражаются все ребра всех объектов, попавших внутрь пирамиды видимости.

При построении контурного изображения предполагается, что объекты — многогранники и они представляются набором своих граней (см. рис. 3.9, *б*). В образе отражаются видимые грани видимых объектов.

Построение полутонового изображения предполагает, что точки объектов могут иметь некоторые свойства (цвет, прозрачность и т. п.). Кроме того в трехмерной сцене могут присутствовать источники света. При построении образа в этом случае атрибуты каждой точки могут меняться в зависимости от расстояния от неё до наблюдателя и до источников света, от взаимного расположения объектов на сцене.



(*а*) (*б*)

Рис. 3.9. Проволочное (*а*) и контурное (*б*) изображения

3.4. Построение ортогональной трехмерной проекции

Простейший способ построения трехмерного образа — получение ортогональной (прямоугольной) проекции объектов трехмерной сцены на окно наблюдения. При условии параллельности окна наблюдения плоскости xOy построение такой проекции заключается в отбрасывании третьей координаты (координаты z) для каждой точки. Этого приема достаточно для построения проволочного изображения. Построение об-

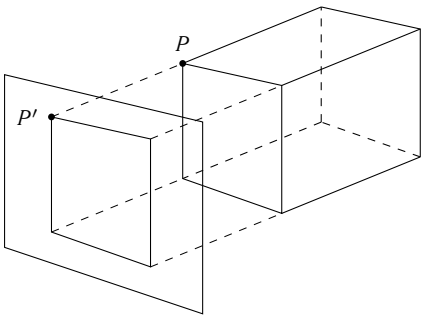


Рис. 3.10. Построение проволочной прямоугольной проекции

раза по такой проекции сводится к операции кадрирования.

При построении контурного или полутонового изображения, бывает необходимо провести вспомогательное преобразование трехмерной сцены с целью получения промежуточного трехмерного прообраза, расположенного в ограниченном пространстве. Размерности такого пространства могут варьироваться для различных графических

систем. Здесь, в качестве примера, будем использовать ограничения

$$-1 \leqslant x \leqslant 1,$$
$$-1 \leqslant y \leqslant 1,$$
$$-1 \leqslant z \leqslant 1.$$

Предположим, что исходный прообраз задан в системе координат наблюдателя. Пусть размеры окна наблюдения, как и раньше, V_x по горизонтали и V_y по вертикали и окно наблюдения расположено на расстоянии $near$ от наблюдателя. Пусть видимая часть сцены ограничена расстоянием far по оси Oz от наблюдателя. Тогда вспомогательное преобразование заключается в сдвиге исходного трехмерного прообраза в положительном направлении оси Oz на $(far + near)/2$ (чтобы совместить центр сцены с началом координат) с последующим масштабированием в $2/V_x$ раз по оси Ox, $2/V_y$ раз по оси Oy и $2/(far - near)$ раз по оси Oz:

$$\begin{bmatrix} \frac{2}{V_x} & 0 & 0 & 0 \\ 0 & \frac{2}{V_y} & 0 & 0 \\ 0 & 0 & \frac{2}{far - near} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{far + near}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

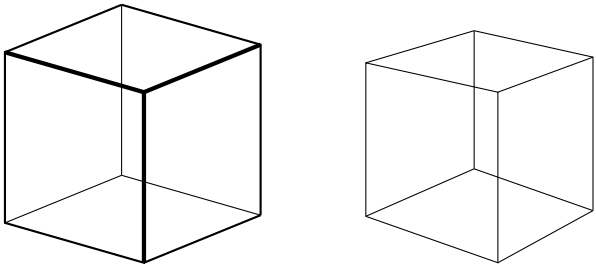
Назовем матрицу совмещенного преобразования $Ortho(V_x, V_y, near, far)$:

$$Ortho(V_x, V_y, near, far) = \begin{bmatrix} \frac{2}{V_x} & 0 & 0 & 0 \\ 0 & \frac{2}{V_y} & 0 & 0 \\ 0 & 0 & \frac{2}{far - near} & \frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

3.5. Факторы повышения наглядности глубины

Глубиной точки трехмерной сцены назовем расстояние от наблюдателя до этой точки.

При построении трехмерного образа зачастую необходимо учитывать восприятие глубины объектов в изображении. Факторы, влияющие на повышение восприятия глубины объектов образа называются факторами повышения наглядности глубины изображения. Существует несколько факторов повышения наглядности глубины изображения. Один из таких факторов — регулировка яркости (см. рис. 3.11, *а*). В этом



(*а*) (*б*)

Рис. 3.11. Регулировка яркости (*а*) и перспектива (*б*)

случае яркость точек образа понижается по мере увеличения глубины соответствующих точек прообраза.

Другой, наиболее естественный и наиболее распространенный фактор повышения наглядности глубины — перспектива (см. рис. 3.11, *б*). Перспектива — намеренное искажение объектов в образе, при котором изменяются размеры объектов по мере изменения расстояния от них до наблюдателя (например, размер шпал у колеи железной дороги кажется уменьшающимся по мере удаления от наблюдателя к горизонту).

Изображение, построенное с перспективой, называется перспективным изображением.

3.6. Построение проволочного перспективного изображения

Пусть задана трехмерная сцена в системе координат наблюдателя. Пусть $near$ — расстояние от наблюдателя до окна наблюдения, V_x, V_y — размеры окна наблюдения по вертикали и горизонтали соответственно.

Рассмотрим трехмерную сцену в проекции на плоскость yOz (см. рис. 3.13). Наблюдатель видит точку $P = (x, y, z)$ через точку $P' = (x', y', z')$ окна наблюдения.

Из подобия треугольников можно вывести

$$\begin{cases} y' = \dfrac{near}{-z} y, \\ z' = -near. \end{cases}$$

Если дополнительно рассмотреть трехмерную сцену в проекции на плоскость xOz, то

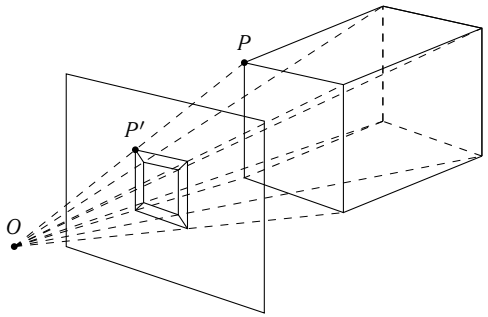


Рис. 3.12. Построение проволочной перспективной проекции

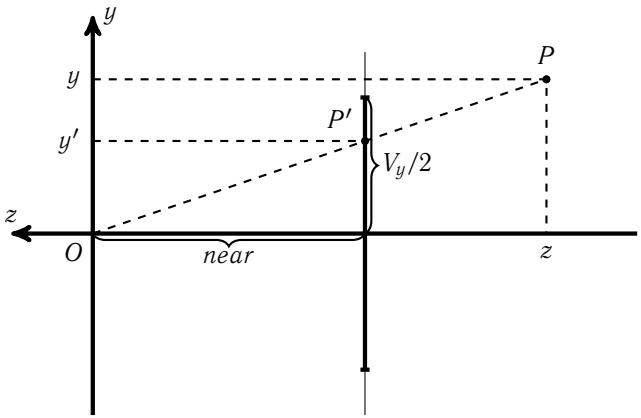


Рис. 3.13. Трехмерная сцена в проекции на плоскость yOz

можно получить

$$\begin{cases} x' = -\dfrac{near}{z}x, \\ y' = -\dfrac{near}{z}y, \\ z' = -near. \end{cases} \tag{3.4}$$

Последнее преобразование можно записать в матричной форме:

$$\begin{bmatrix} \chi' \\ \gamma' \\ \zeta' \\ \alpha' \end{bmatrix} = \begin{bmatrix} near & 0 & 0 & 0 \\ 0 & near & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \chi \\ \gamma \\ \zeta \\ \alpha \end{bmatrix}. \tag{3.5}$$

В преобразовании (3.5) игнорируется третья координата результата — здесь она приравнивается к нулю. Кроме того заметим, что исходная третья координата присваи-

вается четвертой координате, вследствие чего после перехода из однородной системы координат в обычную произойдет деление на значение z.

Приведенное преобразование называется простым перспективным преобразованием, а матрица в последнем соотношении называется матрицей перспективной проекции.

Так как все точки трехмерной сцены после перспективного преобразования располагаются в плоскости окна наблюдения, т. е. сцена из трехмерной превратилась в двумерную, то дальнейшее построение образа сводится к операции кадрирования с параметрами $V_{cx} = -V_x/2$, $V_{cy} = -V_y/2$.

Для проведения операций удаления невидимых линий и поверхностей введем вспомогательный промежуточный прообраз, подобный прообразу, полученному с помощью преобразования (3.3). Так же как и в случае построения заготовки для прямоугольной проекции получим заготовку в области

$$\begin{aligned} -1 \leqslant x \leqslant 1, \\ -1 \leqslant y \leqslant 1, \\ -1 \leqslant z \leqslant 1. \end{aligned}$$

К каждой точке исходного прообраза применим перспективное преобразование, но кроме этого сохраним её глубину.

Для координат x и y применяем перспективное преобразование

$$\begin{cases} x' = -\dfrac{near}{z} x, \\ y' = -\dfrac{near}{z} y, \end{cases} \tag{3.6}$$

после этого проведем масштабирование, подобное преобразованию (3.3).

$$\begin{cases} x'' = x' \dfrac{2}{V_x}, \\ y'' = y' \dfrac{2}{V_y}. \end{cases} \tag{3.7}$$

Что касается координаты z — нам необходимо преобразование из z в z'', для которого выполняются условия:

- если $z_1 = z_2$, то $z_1'' = z_2''$;
- если $z = -near$, то $z'' = 1$;
- если $z = -far$, то $z'' = -1$;

- если z_1 и z_2 такие, что

$$-near \geqslant z_1 > z_2 \geqslant -far,$$

то для z_1'' и z_2'' должно выполняться

$$1 \geqslant z_1'' > z_2'' \geqslant -1.$$

Для получения общего преобразования представим его в матричной форме:

$$\begin{bmatrix} \chi'' \\ \gamma'' \\ \zeta'' \\ \alpha'' \end{bmatrix} = \begin{bmatrix} near \cdot \dfrac{2}{V_x} & 0 & 0 & 0 \\ 0 & near \cdot \dfrac{2}{V_y} & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Осталось выяснить, чему должны равняться A и B.

В скалярной форме преобразование запишется как система

$$\begin{cases} \chi'' = near \cdot \dfrac{2}{V_x} \cdot x \\ \gamma'' = near \cdot \dfrac{2}{V_y} \cdot y \\ \zeta'' = Az + B \\ \alpha'' = -z \end{cases}$$

После перехода из однородной системы координат получим

$$\begin{cases} x'' = -\dfrac{near}{z} \cdot \dfrac{2}{V_x} \cdot x \\ y'' = -\dfrac{near}{z} \cdot \dfrac{2}{V_y} \cdot y \\ z'' = -A - \dfrac{B}{z} \end{cases}$$

Для получения A и B воспользуемся вторым и третьим из вышеперечисленных свойств преобразования

$$\begin{cases} z = -near & \rightarrow \quad z'' = 1 \\ z = -far & \rightarrow \quad z'' = -1 \end{cases}$$

Отсюда

$$\begin{cases} -A - \dfrac{B}{-near} = 1 \\ -A - \dfrac{B}{-far} = -1 \end{cases}$$

Выразим из последней системы равенств A и B, получим

$$\begin{cases} A = \dfrac{far + near}{far - near} \\ B = \dfrac{2 \cdot far \cdot near}{far - near} \end{cases}$$

Таким образом матрица преобразования сформирована полностью. Обозначим эту матрицу $Frustrum(V_x, V_y, near, far)$:

$$Frustrum(V_x, V_y, near, far) = \begin{bmatrix} near \cdot \frac{2}{V_x} & 0 & 0 & 0 \\ 0 & near \cdot \frac{2}{V_y} & 0 & 0 \\ 0 & 0 & \frac{far + near}{far - near} & \frac{2 \cdot far \cdot near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Исходные данные для задачи перспективного преобразования могут задаваться иначе: вместо размеров окна наблюдения задается значение соотношения сторон окна *aspect* ($aspect = V_x/V_y$) и угол вертикального обзора *fovy* (см. рис. 3.14).

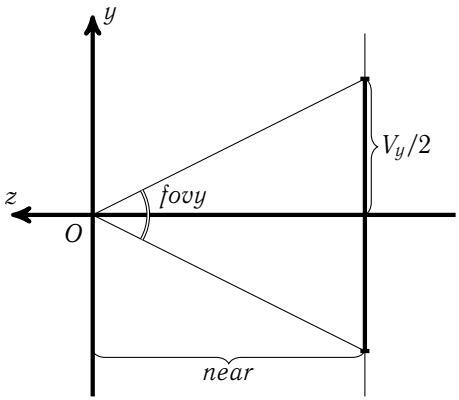


Рис. 3.14. Угол вертикального обзора

Из треугольника на рисунке 3.14

$$\frac{V_y}{2} = near \cdot \operatorname{tg} \frac{fovy}{2},$$

откуда

$$V_y = 2 \cdot near \cdot \operatorname{tg} \frac{fovy}{2}.$$

Следовательно

$$V_x = aspect \cdot V_y = 2 \cdot aspect \cdot near \cdot \operatorname{tg} \frac{fovy}{2}.$$

Подставим эти значения в матрицу $Frustrum(V_x, V_y, near, far)$ и полученную матрицу

обозначим через $Perspective(fovy, aspect, near, far)$:

$$Perspective(fovy, aspect, near, far) =$$
$$= \begin{bmatrix} \frac{1}{aspect} \operatorname{ctg} \frac{fovy}{2} & 0 & 0 & 0 \\ 0 & \operatorname{ctg} \frac{fovy}{2} & 0 & 0 \\ 0 & 0 & \frac{far + near}{far - near} & \frac{2 \cdot far \cdot near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

3.7. Экранная система координат

Полученную в результате преобразований *Ortho*, *Frustrum* или *Perspective* трехмерную сцену обычно используют как заготовку для алгоритмов отсечения невидимых линий и граней, запускаемых для получения окончательного образа. Будем называть правую систему координат связанную с такой трехмерной сценой — экранной системой координат.

Переход от экранной системы координат к системе координат экрана заключается в отбрасывании третьей координаты для всех точек и последующей операции кадрирования с параметрами $V_{cx} = -1$, $V_{cy} = -1$, $V_x = 2$, $V_y = 2$.

4. Растровые алгоритмы

4.1. Алгоритмы заливки области

В большинстве приложений используется одно из существенных достоинств растровых устройств — возможность заполнения областей экрана.

Существует две разновидности заполнения:

- для заполнения внутренней части многоугольника, заданного координатами его вершин.
- для заливки области, которая либо очерчена границей с кодом пиксела, отличающимся от кодов любых пикселов внутри области, либо закрашена пикселами с заданным кодом;

Для приложений, связанных в основном с интерактивной работой, используются алгоритмы заполнения области с затравкой. При этом тем или иным образом задается заливаемая (перекрашиваемая) область, атрибут (например, цвет), которым будет выполняться заливка и точка в области, начиная с которой начнется заливка.

В дальнейшем, когда будем говорить «цвет пиксела», будем подразумевать, что на месте цвета может быть любой атрибут пиксела.

По способу задания области делятся на два типа:

- гранично-определенные, задаваемые своей (замкнутой) границей такой, что цвет пикселов границы отличен от цвета внутренней, перекрашиваемой части области. На цвет пикселов внутренней части области налагаются два условия: они должны быть отличны от цвета пикселов границы и цвета заливки. Если внутри гранично-определенной области имеется еще одна граница, нарисованная пикселами с тем же цветом, что и внешняя граница, то соответствующая часть области не должна перекрашиваться;
- внутренне-определенные, нарисованные одним определенным цветом пиксела. При заливке этот цвет заменяется на новый цвет.

В этом состоит основное отличие заливки области с затравкой от заполнения многоугольника. В последнем случае мы сразу имеем всю информацию о предельных размерах части экрана, занятой многоугольником. Поэтому определение принадлежности пиксела многоугольнику базируется на быстро работающих алгоритмах, использующих когерентность строк и ребер. В алгоритмах же заливки области с затравкой нам

вначале надо прочитать пиксел, затем определить принадлежит ли он области и если принадлежит, то перекрасить.

Заливаемая область или ее граница — некоторое связное множество пикселов. По способам доступа к соседним пикселам области делятся на 4-х и 8-ми связные. В 4-х связных областях доступ к соседним пикселам осуществляется по четырем направлениям — горизонтально влево и вправо и в вертикально вверх и вниз. В 8-ми связных областях к этим направлениям добавляются еще 4 диагональных. Используя связность мы можем, двигаясь от точки затравки по направлениям связности, достичь и закрасить все пикселы области.

Важно отметить, что для 4-х связной области достаточно 8-ми связной границы (рис. 4.1, а), а для 8-ми связной области граница должна быть 4-х связна (см.

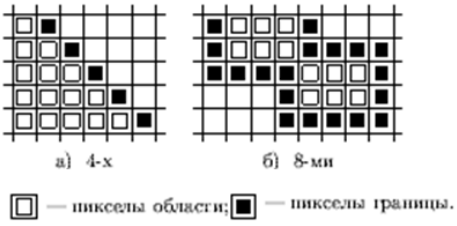


Рис. 4.1. 4-х и 8-ми связные области.

рис. 4.1, б). Поэтому заполнение 4-х связной области 8-ми связным алгоритмом может привести к «просачиванию» через границу и заливке пикселов в примыкающей области.

4.1.1. Простой алгоритм заливки

Рассмотрим простой алгоритм заливки гранично-определенной 4-х связной области (алгоритм 1).

Заливка выполняется следующим образом:

- определяется является ли пиксел граничным или уже закрашенным,
- если нет, то пиксел перекрашивается, затем проверяются и если надо перекрашиваются 4 соседних пиксела.

На рис. 4.2 показан выбранный порядок перебора соседних пикселов, а на рис. 4.3

Алгоритм 1: Простой алгоритм заливки

Вход: P — закрашиваемая область, C_{gr} — цвет границы области, C_{zal} — цвет заливки области.

начало алгоритма
- Пусть $S = \varnothing$ — пустой стек, в котором будем сохранять точки;
- Определить координаты (x_i, y_i) внутренней точки области;
- Закрасить точку (x_i, y_i) цветом C_{zal};
- Положить точку (x_i, y_i) в стек S;
- **цикл пока** $S \neq \varnothing$ **выполнять**
  - Взять (забрать) точку (x_j, y_j) из стека S;
  - **цикл для** $(x, y) \in \{(x_j + 1, y_j), (x_j, y_j + 1), (x_j - 1, y_j), (x_j, y_j - 1)\}$ **выполнять**
    - Определить C_{xy} — цвет точки (x, y);
    - **если** $C_{xy} \neq C_{gr}$ *и* $C_{xy} \neq C_{zal}$ **то**
      - Закрасить точку (x, y);
      - Положить точку (x, y) в стек S;

конец алгоритма

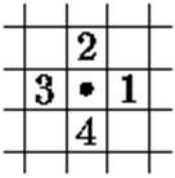


Рис. 4.2. Порядок перебора соседних точек.

соответствующий ему порядок закраски простой гранично-определенной области.

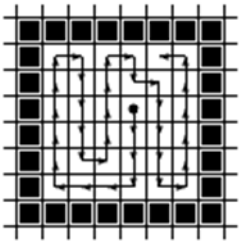


Рис. 4.3. Порядок закраски области.

Рассмотренный алгоритм легко модифицировать для работы с 8-ми связными гранично-определенными областями или же для работы с внутренне-определенными областями.

Очевидный недостаток алгоритмов непосредственно использующих связность за-

крашиваемой области — большие затраты памяти на стек. Проверка цвета некоторых пикселов может проводиться многократно. Это приводит к потере быстродействия. Несколько более экономен далее рассматриваемый построчный алгоритм заливки.

4.1.2. Построчный алгоритм заливки с затравкой

Использует пространственную когерентность:

- пикселы в строке меняются только на границах;
- при перемещении к следующей строке размер заливаемой строки скорее всего или неизменен или меняется на 1 пиксел.

Таким образом, на каждый закрашиваемый фрагмент строки в стеке хранятся координаты только одного начального пиксела, что приводит к существенному уменьшению размера стека.

Последовательность работы алгоритма для гранично определенной области следующая:

1. Координата затравки помещается в стек, затем до исчерпания стека выполняются пункты 2-4.
2. Координата очередной затравки извлекается из стека и выполняется максимально возможное закрашивание вправо и влево по строке с затравкой, т.е. пока не попадется граничный пиксел. Пусть это x_l и x_r, соответственно.
3. Анализируется строка ниже закрашиваемой в пределах от x_l до x_r и в ней находятся крайние правые пикселы всех незакрашенных фрагментов. Их координаты заносятся в стек.
4. То же самое проделывается для строки выше закрашиваемой.

Более детально метод излагается в алгоримтме 2

Алгоритм 2: Построчный алгоритм заливки области

Вход: P — закрашиваемая область, C_{gr} — цвет границы области, C_{zal} — цвет заливки области.

начало алгоритма
- Пусть $S = \varnothing$ — пустой стек, в котором будем сохранять точки;
- Определить координаты (x_i, y_i) внутренней точки области;
- Положить точку (x_i, y_i) в стек S;
- **цикл пока** $S \neq \varnothing$ **выполнять**
  - Взять (забрать) точку (x_j, y_j) из стека S;
  - Определить C_{xy} — цвет точки (x_j, y_j). **если** $C_{xy} \neq C_{zal}$ **то**
    - $x_{\min} = x_j$; $x_{\max} = x_j$;
    - **повторять вычисления**
      - $x_{\min} = x_{\min} - 1$;
      - Определить $C_{\min}$ — цвет точки $(x_{\min}, y_j)$;
    - **пока** $C_{\min} \neq C_{gr}$;
    - **повторять вычисления**
      - $x_{\max} = x_{\max} + 1$;
      - Определить $C_{\max}$ — цвет точки $(x_{\max}, y_j)$;
    - **пока** $C_{max} \neq C_{gr}$;
    - Начертить линию от $(x_{\min} + 1, y_j)$ до $(x_{\max} - 1, y_j)$ цветом C_{zal};
    - **цикл для каждого** x *от* $x_{\min} + 1$ *до* $x_{\max} - 1$ **выполнить**
      - Определить C_{up} — цвет точки $(x, y_j + 1)$;
      - **если** $C_{up} \neq C_{gr}$ *и* $C_{up} \neq C_{zal}$ **то**
        - Положить точку $(x, y_j + 1)$ в стек S;
      - Определить C_{dn} — цвет точки $(x, y_j - 1)$;
      - **если** $C_{dn} \neq C_{gr}$ *и* $C_{dn} \neq C_{zal}$ **то**
        - Положить точку $(x, y_j - 1)$ в стек S;

конец алгоритма

4.1.3. Заполнение многоугольника

В данном разделе рассмотрим алгоритм заполнения многоугольника.

Простейший способ заполнения многоугольника, заданного координатами вершин, заключается в определении принадлежит ли текущий пиксел внутренней части многоугольника. Если принадлежит, то пиксел заносится. Определить принадлежность пиксела многоугольнику можно, например, подсчетом суммарного угла с вершиной на пикселе при обходе контура многоугольника. Если пиксел внутри, то угол будет равен 360 , если вне - 0.

Вычисление принадлежности должно производиться для всех пикселов экрана и так как большинство пикселов скорее всего вне многоугольников, то данный способ слишком расточителен. Объем лишних вычислений в некоторых случаях можно сократить использованием прямоугольной оболочки — минимального прямоугольника, объемлющего интересующий объект, но все равно вычислений будет много.

4.1.4. Построчное заполнение многоугольника

Реально используются алгоритмы построчного заполнения, основанные на том, что соседние пикселы в строке скорее всего одинаковы и меняются только там где строка пересекается с ребром многоугольника. Это свойство называется когерентностью растровых строк (строки сканирования y_i, y_{i+1}, y_{i+2} на рис. 4.4). При этом достаточно определить x-координаты пересечений строк сканирования с ребрами. Пары отсортированных точек пересечения задают интервалы заливки.

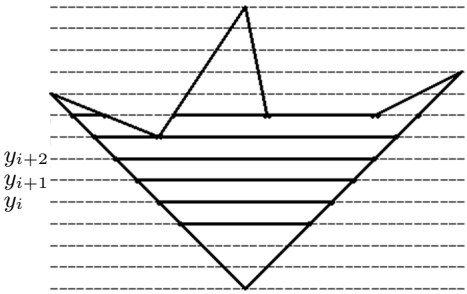


Рис. 4.4. Построчное заполнение многоугольника.

Кроме того, если какие-либо ребра пересекались i-й строкой, то они скорее всего будут пересекаться также и строкой $i+1$. (строки сканирования y_i и y_{i+1} на рис. 4.4). Это называется когерентностью ребер. При переходе к новой строке легко вычислить новую x-координату точки пересечения ребра, используя x-координату старой точки пересечения и тангенс угла наклона ребра:

$$x_{i+1} = x_i + \frac{1}{k},$$

где $k = \dfrac{\Delta y}{\Delta x}$ — тангенс угла наклона ребра, так как $y_{i+1} = y_i + 1$.

Смена же количества интервалов заливки происходит только тогда, когда в строке сканирования появляется вершина.

Учет когерентности строк и ребер позволяет построить для заполнения многоугольников различные высокоэффективные алгоритмы построчного сканирования. Для каждой строки сканирования рассматриваются только те ребра, которые пересекают строку. Они задаются списком активных ребер (*AEL*). При переходе к следующей

строке для пересекаемых ребер перевычисляются x-координаты пересечений. При появлении в строке сканирования вершин производится перестройка AEL. Ребра, которые перестали пересекаться, удаляются из AEL, а все новые ребра, пересекаемые строкой заносятся в него.

Общая схема алгоритма, динамически формирующего список активных ребер и заполняющего многоугольник снизу-вверх, следующая:

1. Подготовить служебные целочисленные массивы y-координат вершин и номеров вершин.
2. Совместно отсортировать y-координаты по возрастанию и массив номеров вершин для того, чтобы можно было определить исходный номер вершины.
3. Определить пределы заполнения по оси Oy — y_{min} и y_{max}. Стартуя с текущим значением $y_t = y_{min}$, исполнять пункты 4-9 до завершения раскраски.
4. Определить число вершин, расположенных на строке y_t — текущей строке сканирования.
5. Если вершины есть, то для каждой из вершин дополнить список активных ребер, используя информацию о соседних вершинах.

   Для каждого ребра в список активных ребер заносятся:

   - начальное значение x-координаты,
   - максимальное значение y-координаты ребра,
   - тангенс угла наклона ребра.

   Если обнаруживаются горизонтальные ребра, то они просто закрашиваются и информация о них в список активных ребер не заносится.

   Если после этого обнаруживается, что список активных ребер пуст, то заполнение закончено.
6. По списку активных ребер определяется y_{next} — y-координата ближайшей вершины. (Вплоть до y_{next} можно не заботиться о модификации AEL а только менять x-координаты пересечений строки сканирования с активными ребрами).
7. В цикле от y_t до y_{next}:

   а) выбрать из списка активных ребер и отсортировать x-координаты пересечений активных ребер со строкой сканирования;

б) определить интервалы и выполнить закраску;

в) перевычислить координаты пересечений для следующей строки сканирования.

8. Проверить не достигли ли максимальной y-координаты. Если достигли, то заливка закончена, иначе выполнить пункт 9.
9. Очистить список активных ребер от ребер, закончившихся на строке y_{next} и перейти к пункту 4.

Более детально метод изложен в алгоритме 3.

Алгоритм 3: Алгоритм построчной заливки многоугольника

Вход: P — список вершин многоугольника (без самопересечений), C_{zal} — цвет закраски.
начало алгоритма
- Сформировать список S ребер многоугольника, где для каждого ребра $[(x_1, y_1), (x_2, y_2)]$ выполняется $y_1 \leqslant y_2$;
- Упорядочить список S по возрастанию значения y_1;
- Найти y_{min} и y_{max} — минимальное и максимальное значение координаты y точек вершин многоугольника;
- $AEL = \varnothing$; $y_t = y_{min}$; $y_{S\,next} = y_{min}$;

цикл пока $y_t \leqslant y_{max}$ **выполнять**
- **если** $y_t = y_{S\,next}$ **то**
  - Добавить в AEL все тройки $\left(x_1, y_2, \dfrac{x_2 - x_1}{y_2 - y_1}\right)$, составленные для каждого отрезка из S, у которого $y_1 = y_t$ и $y_1 \neq y_2$;
  - Начертить все ребра в S, у которых $y_1 = y_t$ и $y_1 = y_2$;
  - Удалить все ребра в S, у которых $y_1 = y_t$;
  - Для отрезков в S найти $y_{S\,next}$ — минимальное значение y_1 у отрезков в S;
  - Отсортировать AEL по возрастанию первого элемента и по возрастанию третьего элемента;
  - Найти $y_{AEL\,next}$ — минимальное значение второго элемента, отличное от y_t;
- **цикл для** i *от* 1 *до* $|AEL|$ *с шагом* 2 **выполнять**
  - Начертить отрезок $[(x_i, y_t), (x_{i+1}, y_t)]$ цветом C_{zal}, где $(x_i, y_i, \Delta_i x)$ обозначает i-й элемент списка AEL;
- В каждой тройке $(x_j, y_j, \Delta_j x)$ в AEL заменить x_j на $x_j + \Delta_j x$;
- $y_t = y_t + 1$;
- **если** $y_t \geqslant y_{AEL\,next}$ **то**
  - Удалить из AEL все тройки, второй элемент которых меньше или равен y_t;
  - Найти $y_{AEL\,next}$ — минимальное значение второго элемента, отличное от y_t;

конец алгоритма

5. Алгоритмы отсечения

5.1. Двумерные алгоритмы отсечения невидимых линий

Рассмотрим проблему, незримо присутствующую в большинстве задач компьютерной графики. Эта проблема отсечения изображения по некоторой границе, например, по границе экрана, или, в общем случае, некоторого окна. Рассмотрим задачу отсечения применительно к отрезкам прямых. Некоторые из них полностью лежат внутри области экрана, другие целиком вне ее, а некоторые пересекают границу экрана. Правильное отображение отрезков означает нахождение точек пересечения их с границей экрана и рисование только тех их частей, которые попадают на экран.

Область, относительно границ которой проводится отсечение, будем называть областью видимости (ОВ). Будем считать, что областью видимости может являться некоторый выпуклый многоугольник (см. рис. 5.1). Продолжим каждую из границ

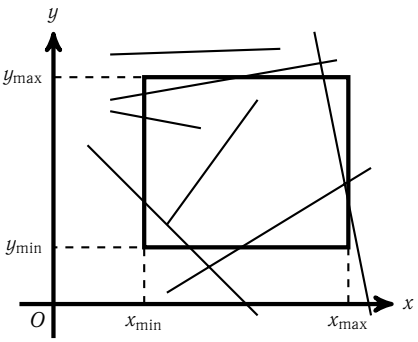


Рис. 5.1. Границы прямоугольной области видимости.

области видимости до бесконечных прямых. Каждая из таких прямых делит плоскость на 2 части. Назовем «стороной внутренности» ту полуплоскость, в которой находится область видимости, как это показано на рис. 5.2. Другую полуплоскость назовем «стороной внешности».

Таким образом, область видимости может быть определена как область, которая находится на стороне внутренности всех линий границ.

Рассматривая положение отрезка относительно линии i-ой границы ОВ будем представлять его в виде связанного вектора. В таком случае можно охарактеризовать

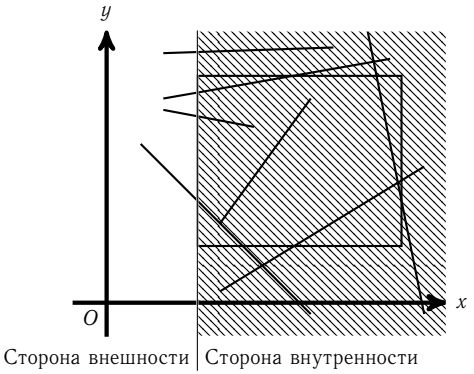


Рис. 5.2. Стороны внешности и внутренности.

направление этого отрезка. Продолжим отрезок до прямой линии. Будем говорить, что отрезок направлен в сторону внутренности относительно линии i-ой границы OB, если двигаясь по линии отрезка в направлении, заданном отрезком как связанным вектором, пересекая i-ю границу OB попадаем со стороны внешности на сторону внутренности.

Подобное определение можно сказать о направлении отрезка в сторону внешности относительно линии i-ой границы OB.

5.1.1. Алгоритм Коэна — Сазерленда

Алгоритм предполагает, что область видимости — прямоугольник, стороны которого параллельны осям координат.

Этот алгоритм позволяет быстро выявить отрезки, которые могут быть или приняты или отброшены целиком. Вычисление пересечений требуется когда отрезок не попадает ни в один из этих классов. Этот алгоритм особенно эффективен в двух крайних случаях:

- большинство примитивов содержится целиком в большом окне,
- большинство примитивов лежит целиком вне относительно маленького окна.

Идея алгоритма состоит в следующем: Прямые линии, ограничивающие область видимости делят всё двумерное пространство на 9 областей (см. рис. 5.3).

Каждой из областей присваивается 4-х-разрядный двоичный код. Пример такой кодировки приведен на рис. 5.4. Здесь:

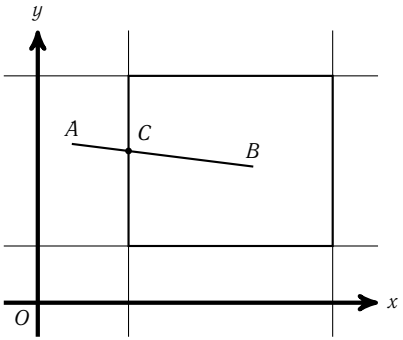


Рис. 5.3. Девять областей при прямоугольной области видимости

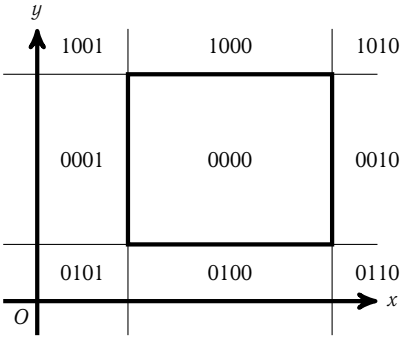


Рис. 5.4. Кодировка областей.

- единица в первом разряде — область слева от левой границы области видимости;
- единица во втором разряде — область справа от правой границы области видимости;
- единица в третьем разряде — область под нижней границей области видимости;
- единица в четвертом разряде — область над верхней границей области видимости;

Две конечные точки отрезка получают 4-х-разрядные двоичные коды, соответствующие областям, в которые они попали. В нижеприведенном алгоритме будем обозначать $C(A)$ — код области, в которую попала точка A.

Определение того является ли отрезок видимым выполняется следующим образом:

- если коды обоих концов отрезка равны 0, то отрезок целиком внутри окна, отсечение не нужно, отрезок принимается как тривиально видимый (отрезок AB на рис. 5.5);

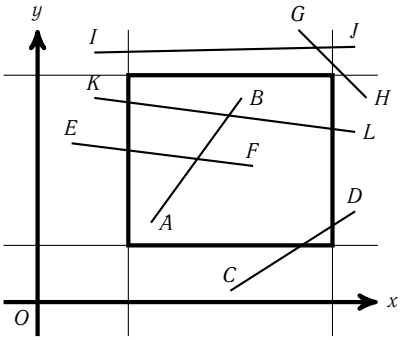


Рис. 5.5. Примеры расположения отрезков.

- если логическое поразрядное «И» кодов обоих концов отрезка не равно нулю, то отрезок целиком вне окна, отсечение не нужно, отрезок отбрасывается как тривиально невидимый (отрезок IJ на рис. 5.5);
- если логическое поразрядное «И» кодов обоих концов отрезка равно нулю, то отрезок подозрительный, он может быть частично видимым (отрезки CD, EF, KL) или целиком невидимым (отрезок GH); для него нужно определить координаты пересечений со сторонами окна и для каждой полученной части определить тривиальную видимость или невидимость. При этом для отрезков EF и GH потребуется вычисление одного пересечения, для остальных (CD и KL) - двух.

При расчете пересечения используется горизонтальность либо вертикальность сторон окна, что позволяет определить координату x или y точки пересечения без вычислений.

Несколько детальней метод Коэна—Сазерленда представлен в алгоритме 4.

Алгоритм 4: Алгоритм Коэна—Сазерленда

Вход: S — список всех отрезков сцены.
Выход: S_1 — список всех видимых частей отрезков сцены.
начало алгоритма
  $S_1 = \varnothing$;
  цикл пока $S \neq \varnothing$ **выполнять**
    Пусть AB очередной отрезок из S;
    Определить C_1 и C_2 — коды областей, в которые попали точки A и B соответственно;
    если $C_1 = C_2 = 0$ **то** *// отрезок полностью видим*
      Поместить AB в список S_1;
    иначе если $C_1 \& C_2 = 0$ **то** *// отрезок может быть частично видим*
      если $C_1 = 0$ **то** поменять местами значения A и B;
      // Теперь A точно за пределами области видимости.
      Определить одну из границ области видимости, за которой лежит точка A;
      Найти точку C — точку пересечения AB с этой границей;
      Поместить отрезок CB в список S;
    // Иначе, если $C_1 \& C_2 \neq 0$, то отрезок полностью невидим.
  Выдать S_1;
конец алгоритма

5.1.2. Алгоритм Лианга — Барски

В 1982 г. Лианг и Барски предложили алгоритмы отсечения прямоугольным окном с использованием параметрического представления для двух, трех и четырехмерного отсечения. По утверждению авторов, данный алгоритм в целом превосходит алгоритм Коэна — Сазерленда. Однако, это утверждение справедливо только для случая когда оба конца видимого отрезка вне окна и окно небольшое (до 50×50 при разрешении 1000×1000).

Пусть внутренняя часть области видимости может быть выражена с помощью следующих неравенств (рис. 5.1).

$$\begin{cases} x_{\min} \leqslant x \leqslant x_{\max}, \\ y_{\min} \leqslant y \leqslant y_{\max}. \end{cases} \tag{5.1}$$

Отсекаемый отрезок прямой может быть представлен в параметрическом представлении (как в п. 2.3).

$$\begin{cases} x(t) = x_A + (x_B - x_A)t, \\ y(t) = y_A + (y_B - y_A)t, \\ 0 \leqslant t \leqslant 1. \end{cases} \tag{5.2}$$

Подставляя параметрическое представление, заданное уравнениями (5.2), в неравенства (5.1), получим следующие соотношения для части отрезка, находящейся в окне отсечения:

$$\begin{cases} x_{\min} \leqslant x_A + (x_B - x_A)t \leqslant x_{\max}, \\ y_{\min} \leqslant y_A + (y_B - y_A)t \leqslant y_{\max}, \\ 0 \leqslant t \leqslant 1. \end{cases} \tag{5.3}$$

Рассматривая неравенства (5.3), видим, что они имеют одинаковую форму вида:

$$\begin{cases} P_i t \leqslant Q_i, \\ i = 1, 2, 3, 4, \\ 0 \leqslant t \leqslant 1. \end{cases}$$

Здесь использованы следующие обозначения:

$$\begin{aligned} P_1 &= x_A - x_B; & Q_1 &= x_A - x_{\min}; \\ P_2 &= x_B - x_A; & Q_2 &= x_{\max} - x_A; \\ P_3 &= y_A - y_B; & Q_3 &= y_A - y_{\min}; \\ P_4 &= y_B - y_A; & Q_4 &= y_{\max} - y_A. \end{aligned} \tag{5.4}$$

Вспоминая определения стороны внутренности и стороны внешности, замечаем, что каждое из неравенств (5.4) соответствует одной из граничных линий (левой, правой, нижней и верхней, соответственно) и описывает ее видимую сторону. Удлиним AB до бесконечной прямой. Тогда каждое неравенство задает диапазон значений параметра t, для которых эта удлиненная линия находится на видимой стороне соответствующей линии границы. Более того, конкретное значение параметра t для точки пересечения есть $t = Q_i/P_i$.

Знак Q_i показывает на какой стороне соответствующей линии границы находится точка A. А именно, если $Q_i \geqslant 0$, тогда A находится на стороне внутренности линии границы, включая и ее. Если же $Q_i < 0$, тогда A находится на внешней (невидимой) стороне.

Рассмотрим P_i в соотношениях 5.4. Ясно, что любое P_i может быть меньше 0, больше 0 и равно 0. Если $P_i < 0$, тогда отрезок направлен в сторону внутренности линии i-ой границы, а значит, при пересечении с линией i-ой границы начало отрезка остается на стороне внешности (см. рис. 5.6, *а*). Если $P_i > 0$, тогда отрезок направлен

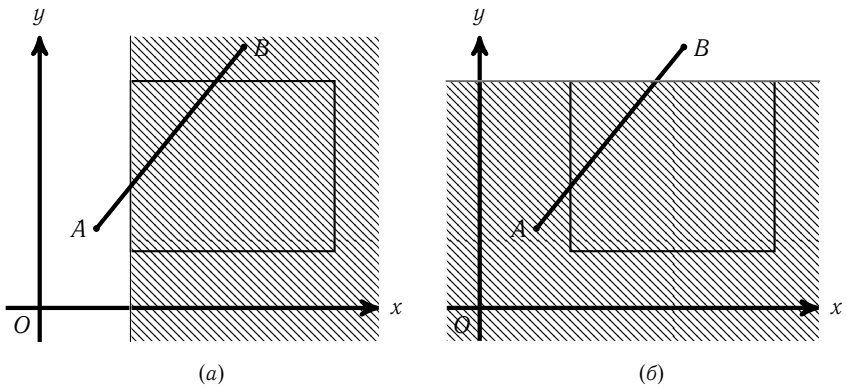


Рис. 5.6. Отрезок направлен: (*а*) в сторону внутренности относительно левой (первой) границы; (*б*) в сторону внешности относительно верхней (четвертой) границы

в сторону внешности линии i-ой границы, а значит, при пересечении с линией i-ой границы конец отрезка остается на стороне внешности (см. рис. 5.6, *б*). Если $P_i = 0$ — отрезок или вырожден в точку, или параллелен i-ой границе, но, в любом случае, он либо полностью находится на стороне внутренности или полностью на стороне внешности относительно линии i-ой границы.

Для всех i таких, что $P_i < 0$, условия видимости имеет вид: $t \geqslant Q_i/P_i$. Из условия принадлежности точек удлиненной линии отрезку AB имеем $t \geqslant 0$. Таким образом, нужно искать:

$$t \geqslant \max(\{Q_i/P_i \mid P_i < 0,\ i = 1,2,3,4\} \cup \{0\}). \tag{5.5}$$

Аналогично, для всех i таких что $P_i > 0$, условия видимости — $t \leqslant Q_i/P_i$ и $t \leqslant 1$. А следовательно

$$t \leqslant \min(\{Q_i/P_i \mid P_i > 0,\ i = 1,2,3,4\} \cup \{1\}). \tag{5.6}$$

Наконец, для всех i, таких что $P_i = 0$ следует проверить знак Q_i. Если $Q_i < 0$, то это случай тривиального отбрасывания, задача отсечения решена и дальнейшие вычисления не нужны. Если же $Q_i \geqslant 0$, то информации, даваемой неравенством, недостаточно и это неравенство игнорируется.

Правая часть неравенств (5.5) и (5.6) — значения параметра t, соответствующие началу и концу видимого сегмента, соответственно. Обозначим эти значения как $t_{\min}$

и $t_{\max}$:

$$\begin{cases} t_{\min} = \max(\{Q_i/P_i \mid P_i < 0,\ i = 1,2,3,4\} \cup \{0\}), \\ t_{\max} = \min(\{Q_i/P_i \mid P_i > 0,\ i = 1,2,3,4\} \cup \{1\}). \end{cases} \tag{5.7}$$

Если сегмент отрезка AB видим, то ему соответствует интервал параметра

$$t_{\min} \leqslant t \leqslant t_{\max}.$$

Следовательно, необходимое условие видимости сегмента:

$$t_{\min} \leqslant t_{\max}$$

Но это недостаточное условие, так как оно игнорирует случай тривиального отбрасывания при $P_i = 0$, если $Q_i < 0$. Тем не менее это достаточное условие для отбрасывания, т. е. если $t_{\min} > t_{\max}$, то отрезок должен быть отброшен. Алгоритм проверяет, если $P_i = 0$ с $Q_i < 0$, или $t_{\min} > t_{\max}$ и в этом случае отрезок немедленно отбрасывается без дальнейших вычислений.

В алгоритме $t_{\min}$ и $t_{\max}$ инициализируются в 0 и 1, соответственно. Затем последовательно рассматривается каждое отношение Qi/Pi.

Если $Pi < 0$, то отношение вначале сравнивается с $t_{\max}$ и, если оно больше $t_{\max}$, то это случай отбрасывания. В противном случае оно сравнивается с $t_{\min}$ и, если оно больше, то $t_{\min}$ должно быть заменено на новое значение.

Если $P_i > 0$, то отношение вначале сравнивается с $t_{\min}$ и, если оно меньше $t_{\min}$, то это случай отбрасывания. В противном случае оно сравнивается с $t_{\max}$ и, если оно меньше, то $t_{\max}$ должно быть заменено на новое значение.

Наконец, если $P_i = 0$ и $Q_i < 0$, то это случай отбрасывания.

На последнем этапе алгоритма, если отрезок еще не отброшен, то $t_{\min}$ и $t_{\max}$ используются для вычисления соответствующих точек.

Общую схему метода Лианга—Барски можно увидеть в алгоритме 5.

Алгоритм 5: Алгоритм Лианга—Барски отсечения отрезков

Вход: S — список всех отрезков сцены.
Выход: S_1 — список всех видимых частей отрезков сцены.
начало алгоритма
  $S_1 = \varnothing$;
  цикл пока $S \neq \varnothing$ **выполнять**
    Пусть AB очередной отрезок из S;
    $t_{min} = 0$; $t_{\max} = 1$; $visible = True$;
    цикл для i *от* 1 *до* 4 **выполнять**
      Вычисляем P_i и Q_i;
      если $P_i = 0$ **то**
        если $Q_i < 0$ **то** *// отрезок полностью невидим*
          $visible = False$; **закончить цикл**;
      иначе
        если $P_i > 0$ **то** $t_{\max} = \min\left\{t_{\max}, \dfrac{Q_i}{P_i}\right\}$;
        иначе $t_{\min} = \max\left\{t_{\min}, \dfrac{Q_i}{P_i}\right\}$;
        если $t_{\min} > t_{\max}$ **то** $visible = False$; **закончить цикл**;
    если $visible$ **то**
      $A' = A + (B - A)t_{\min}$; $B' = A + (B - A)t_{\max}$;
      Поместить отрезок $A'B'$ в S_1;
  Выдать S_1;
конец алгоритма

5.1.3. Алгоритм Кируса — Бека

Рассмотренные выше алгоритмы проводили отсечение по прямоугольному окну, стороны которого параллельны осям координат. Это, конечно, наиболее частый случай отсечения. Однако, во многих случаях требуется отсечение по произвольному многоугольнику, например, в алгоритмах удаления невидимых частей сцены. В этом случае наиболее удобно использование параметрического представления линий, не зависящего от выбора системы координат.

Из предыдущего пункта ясно, что для выполнения отсечения в параметрическом представлении необходимо иметь способ определения ориентации удлиненной линии, содержащей отсекаемый отрезок, относительно линии границы — с внешней стороны на внутреннюю или с внутренней на внешнюю, а также иметь способ определения расположения точки, принадлежащей отрезку, относительно окна — вне, на границе, внутри.

Для этих целей в алгоритме Кируса—Бека, реализующем отсечение произвольным выпуклым многоугольником, используется вектор внутренней нормали к ребру окна.

Пусть область видимости — n-угольник, с вершинами, перечисленными по часовой

стрелке, $F_1, F_2, \ldots, F_n$. i-м ребром этого многоугольника будем называть ребро

$$\begin{cases} F_iF_{i+1}, & \text{если } 1 \leqslant i < n; \\ F_nF_1, & \text{если } i = n. \end{cases}$$

Будем обозначать N_i — нормаль к i-му ребру области видимости, направленную в сторону внутренности относительно i-го ребра (см. рис. 5.7).

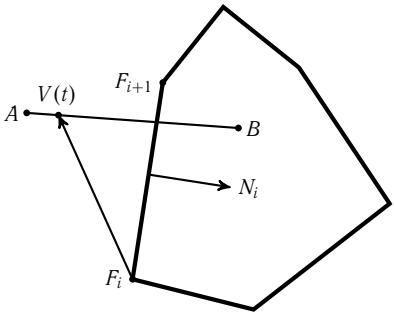


Рис. 5.7. Рассматриваемые векторы в алгоритме Кируса—Бека.

Рассмотрим основные идеи алгоритма Кируса — Бека.

Ориентация отрезка AB относительно i-й стороны области видимости определяется знаком скалярного произведения $P_i = (B - A)N_i$.

- При $P_i < 0$ отсекаемый отрезок направлен с внутренней на внешнюю стороны i-й граничной линии окна (так как, в таком случае, угол между отрезком и вектором нормали N_i больше $\pi/2$);
- при $P_i = 0$ либо точки A и B совпадают, либо отсекаемый отрезок параллелен i-й граничной линии окна;
- при $P_i > 0$ отсекаемый отрезок направлен с внешней на внутреннюю сторону i-й граничной линии окна (так как, в этом случае, угол между отрезком и вектором нормали N_i меньше $\pi/2$).

Для определения расположения точки относительно окна вспомним параметрическое представление отсекаемого отрезка:

$$\begin{cases} V(t) = A + (B - A)t, \\ 0 \leqslant t \leqslant 1. \end{cases} \tag{5.8}$$

Рассмотрим скалярное произведение внутренней нормали N_i к i-й границе на вектор $V(t) - F_i$ — вектор, начинающийся в начальной точке ребра окна и заканчивающийся в некоторой точке $V(t)$ удлиненной линии (см. рис. 5.7).

$$Q_i(t) = (V(t) - F_i)N_i. \tag{5.9}$$

Имеем:

- при $Q_i(t) < 0$ точка $V(t)$ лежит с внешней стороны i-ой границы;
- при $Q_i(t) = 0$ точка $V(t)$ лежит на несущей прямой i-ой границы;
- при $Q_i(t) > 0$ точка $V(t)$ лежит с внутренней стороны i-ой границы.

Таким образом, если необходимо найти точку пересечения линии отрезка с несущей прямой границы области видимости, то нужно найти такое t, для которого выполняется

$$Q_i(t) = 0$$

Распишем $Q_i(t)$ через (5.9) и подставим вместо $V(t)$ правую часть равенства из (5.8), получим

$$(A + (B - A)t - F_i)N_i = 0.$$

Отсюда получим

$$(B - A)N_i t - (F_i - A)N_i = 0. \tag{5.10}$$

Вспомним, что $A = V(0)$, откуда следует

$$P_i t + Q_i(0) = 0.$$

Разрешая равенство относительно t получим

$$t = -\frac{Q_i(0)}{P_i}. \tag{5.11}$$

Равенство (5.11) дает значение параметра t в параметрическом уравнении удлиненной прямой с границей области видимости.

Алгоритм построен следующим образом: Искомые значения параметров $t_{\min}$ и $t_{\max}$ точек концов видимой части отрезка инициализируются значениями 0 и 1, соответствующими началу и концу отсекаемого отрезка.

Затем в цикле для каждой i-й стороны окна отсечения вычисляются значения скалярных произведений, входящих в (5.11).

Если очередное P_i равно 0, то отсекаемый отрезок либо вырожден в точку, либо параллелен i-й стороне окна. В этом случае достаточно проанализировать знак Q_i. Если $Q_i < 0$, то отрезок вне окна и отсечение закончено иначе рассматривается следующая сторона окна.

Если же P_i не равно 0, то по (5.11) можно вычислить значение параметра t для точки пересечения отсекаемого отрезка с i-й границей. Так как отрезок AB соответствует диапазону $0 \leqslant t \leqslant 1$, то все решения, выходящие за данный диапазон следует отбросить. Выбор оставшихся решений определяется знаком P_i.

Если $P_i < 0$, т. е. удлиненная линия направлена с внутренней на внешнюю стороны граничной линии, то проводится поиск значения параметра для конечной точки видимой части отрезка. В этом случае определяется минимальное значение из всех получаемых решений. Оно даст значение параметра $t_{\max}$ для конечной точки отсеченного отрезка. Если текущее полученное значение $t_{\max}$ окажется меньше, чем $t_{\min}$, то отрезок отбрасывается, так как нарушено условие $t_{\min} \leqslant t_{\max}$.

Если же $P_i > 0$, т. е. удлиненная линия направлена с внешней на внутреннюю стороны граничной линии, то проводится поиск значения параметра для начальной точки видимой части отрезка. В этом случае определяется максимальное значение из всех получаемых решений. Оно даст значение параметра $t_{\min}$ для начальной точки отсеченного отрезка. Если текущее полученное значение $t_{\min}$ окажется больше, чем $t_{\max}$, то отрезок отбрасывается, так как нарушено условие $t_{\min} \leqslant t_{\max}$.

На заключительном этапе алгоритма значения $t_{\min}$ и $t_{\max}$ используются для вычисления координат точек пересечения отрезка с границами OB.

Общая схема метода изложена в алгоритме 6.

Алгоритм 6: Алгоритм Кируса—Бека отсечения отрезков

Вход: S — список всех отрезков сцены.
Выход: S_1 — список всех видимых частей отрезков сцены.
начало алгоритма
  $S_1 = \varnothing$;
  цикл пока $S \neq \varnothing$ **выполнять**
    Пусть AB очередной отрезок из S;
    $t_{min} = 0$; $t_{\max} = 1$; $visible = True$;
    цикл для i *от* 1 *до* n **выполнять** *// n — количество ребер OB*
      Вычисляем P_i и $Q_i(0)$;
      если $P_i = 0$ **то**
        если $Q_i(0) < 0$ **то** *// отрезок полностью невидим*
          $visible = False$; **закончить цикл**;
      иначе
        если $P_i > 0$ **то** $t_{\min} = \max\left\{t_{\min}, -\dfrac{Q_i(0)}{P_i}\right\}$;
        иначе $t_{\max} = \min\left\{t_{\max}, -\dfrac{Q_i(0)}{P_i}\right\}$;
        если $t_{\min} > t_{\max}$ **то** $visible = False$; **закончить цикл**;
    если $visible$ **то**
      $A' = A + (B - A)t_{\min}$; $B' = A + (B - A)t_{\max}$;
      Поместить отрезок $A'B'$ в S_1;
  Выдать S_1;
конец алгоритма

5.1.4. Алгоритм Скала

В 1993 году Вацлав Скала предложил еще один алгоритм отсечения отрезков относительно области видимости, заданной выпуклым многоугольником. Основная идея его алгоритма состоит в том, что так как область видимости — выпуклый многоугольник, то любая частично-видимая прямая пересекает границу этого многоугольника точно в двух точках (исключая вырожденные случаи, когда прямая касается границы области видимости). Чтобы определить пересекает ли прямая, на которой лежит отрезок AB, ребро F_iF_{i+1} области видимости Скала предлагает сравнивать знаки псевдоскалярных произведений $\xi = (B - A) \times (F_i - A)$ и $\eta = (B - A) \times (F_{i+1} - A)$, т. е. произведение вектора отрезка с векторами начинающимися в начальной точке отрезка и заканчивающимися в конечных точках ребра области видимости (см. рис. 5.8). Если знаки ξ и η одинаковые, то ребро F_iF_{i+1} лежит по одну сторону от прямой, на которой лежит отрезок AB. Если знаки разные, то прямая на которой лежит AB, пересекает F_iF_{i+1}.

В ходе алгоритма производится подсчет количества ребер области видимости, для

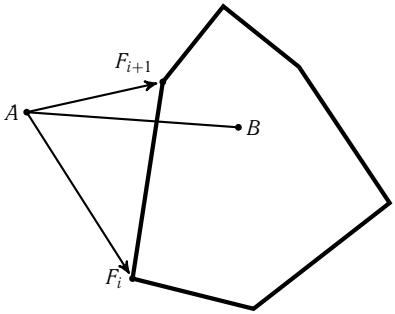


Рис. 5.8. Векторы для произведения в алгоритме Скала.

которых найдется точка пересечения с прямой отрезка AB.

Если найдутся два ребра области видимости, которые пересекаются с прямой отрезка AB, то предлагается проверить принадлежат ли точки пересечения прямой отрезку. Если так, то в соответствующие точки пересечения следует перенести концы отрезка. Например, на рисунке 5.9 прямая отрезка AB пересекает пересекает ребра области видимости в точках C_1 И C_2. Но только лишь точка C_1 принадлежит отрез-

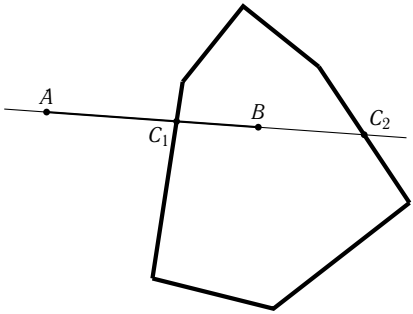


Рис. 5.9. Отсечение отрезка в алгоритме Скала.

ку AB. Только один конец отрезка будет изменен: точка A будет перенесена в точку пересечения C_1.

Когда показатели ξ или η обращается в ноль, то прямая отрезка AB проходит через точку F_i или F_{i+1} соответственно, и в этом случае необходимо проводить дополнительные проверки. Если ξ и η обратились в ноль одновременно, то прямая отрезка AB является несущей прямой данного ребра области видимости. В таком случае не стоит увеличивать счетчик точек пересечения (так как он будет увеличен при рас-

смотрении соседних ребер). Если же в ноль обратился только один из показателей, то нужно зачесть вершину, через которую проходит прямая, за точку пересечения. При этом стоит учесть, что при проверке второго ребра, инцидентного данной вершине, данная точка уже не будет рассматриваться как точка пересечения.

Как и в алгоритме Кируса—Бека, точки пересечения отрезка с ребрами многоугольника, ограничивающего область видимости, будем вычислять посредством параметрического уравнения прямой отрезка. Воспользуемся тем свойством, что если $V(t)$ в параметрическом уравнении (5.8) является точкой пересечения, то в ноль обращается следующее псевдоскалярное произведение

$$(V(t) - F_i) \times F_iF_{i+1} = 0.$$

Распишем в последнем равенстве $V(t)$

$$(A + (B - A)t - F_i) \times F_iF_{i+1} = 0.$$

Перепишем левую часть равенства через сумму двух произведений и вынесем t за пределы произведения

$$(A - F_i) \times F_iF_{i+1} + t((B - A) \times F_iF_{i+1}) = 0.$$

Отсюда, значение t для точки пересечения несущей прямой отрезка AB и ребра F_iF_{i+1}:

$$t = \frac{(F_i - A) \times F_iF_{i+1}}{(B - A) \times F_iF_{i+1}} = \frac{AF_i \times F_iF_{i+1}}{AB \times F_iF_{i+1}}.$$

В общем виде метод Скала представлен в алгоритме 7.

Алгоритм 7: Алгоритм Скала

Вход: список S отрезков сцены.
Выход: S_1 — список видимых частей отрезков из S.
начало алгоритма
  Пусть F_i — все вершины области видимости, $1 \leqslant i \leqslant n$;
  цикл для каждого $2 \leqslant i \leqslant n$ **выполнить** $\overline{w}_i = F_i - F_{i-1}$;
  $\overline{w}_1 = F_1 - F_n$;
  $S_1 = \varnothing$;
  цикл пока S *не пуст* **выполнять**
    Взять из S отрезок AB;
    Присвоить $i = 1$; $k = 0$; $specialCase = True$; $\bar{v}_0 = F_n - A$; $\xi = AB \times \bar{v}_0$;
    цикл пока $i \leqslant n$ *и* $k \neq 2$ **выполнять**
      $\bar{v}_1 = F_i - A$; $\eta = AB \times \bar{v}_1$; $pointFound = False$;
      если $\xi\eta = 0$ **то**
        если $\xi \neq 0$ **то** $pointFound = True$;
        иначе если $\eta \neq 0$ *и* $specialCase$ **то** $pointFound = True$;
      если $\xi\eta < 0$ *или* $pointFound$ **то** $k = k + 1$; $t_k = (\bar{v}_1 \times \overline{w}_i)/(AB \times \overline{w}_i)$;
      если $\xi = 0$ *и* $\eta = 0$ **то** $specialCase = True$;
      иначе $specialCase = False$;
      $\xi = \eta$; $i = i + 1$;
    если $k = 2$ **то**
      если $t_1 > t_2$ **то** поменять местами значения t_1 и t_2;
      если $0 \leqslant t_1 \leqslant 1$ **то** $A' = A + (B - A)t_1$ **иначе** $A' = A$;
      если $0 \leqslant t_2 \leqslant 1$ **то** $B' = A + (B - A)t_2$ **иначе** $B' = B$;
      Поместить $A'B'$ в S_1;
  Выдать S_1;
конец алгоритма

5.1.5. FC-алгоритм

В 1987 г. Собков, Поспишил и Янг предложили алгоритм, названный ими FC-алгоритмом (Fast Clipping), в котором в отличие от алгоритма Коэна—Сазерленда кодируются не конечные точки отрезка, а отрезки целиком. Схема кодирования повторяет используемую в алгоритме Коэна—Сазерленда. На рисунке 5.10 представлен вариант кодирования областей (повторяющий вариант на рисунке 5.4), где каждый код области представлен одной шестнадцатеричной цифрой ($0x$ — префикс, показывающий, что последующее число записано в шестнадцатеричной форме). Код отрезка — комбинация кодов начала и конца отрезка. Так, на рисунке 5.5, код отрезка KL — $0x12$, отрезка GH — $0x82$, а отрезка IJ — $0x9A$.

Если вычислены коды концов отрезка C_1 и C_2, то код отрезка вычисляется по формуле:

$$C_1 * 16 + C_2$$

Так как каждый код может принимать одно из девяти значений, то всего имеется

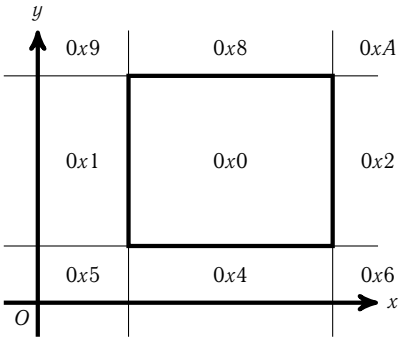


Рис. 5.10. Кодировка областей.

72 возможных варианта расположения отрезка (81 минус 9 случаев, при которых код начала отрезка равен коду конца отрезка).

Каждый код отрезка требует своего набора вычислений для определения отсечения отрезка за минимальное время. Всего имеется одиннадцать основных случаев отсечения, а остальные симметричны к ним. Рассмотрим эти случаи.

Будем использовать следующие обозначения.

- (x_1, y_1), (x_2, y_2) — координаты точки начала и конца отрезка соответственно;
- Clip1Left, Clip1Bottom обозначают алгоритмы переноса начальной точки отрезка в точку пересечения отрезка с левой или нижней границей соответственно;
- Clip2Right, Clip2Top обозначают алгоритмы переноса конечной точки отрезка в точку пересечения отрезка с правой или верхней границей соответственно.

Код отрезка $0x00$ (рис. 5.11, отрезок MN) Случай простого принятия отрезка;

Код отрезка $0x11$ (рис. 5.11, отрезок BL) Случай простого отбрасывания;

Код отрезка $0x19$ (рис. 5.11, отрезок AD) Случай простого отбрасывания;

Код отрезка $0x18$ (рис. 5.11, отрезки AE и BG) Отрезки точно пересекают левую границу области видимости, поэтому вначале надо выполнить Clip1Left, после чего проверить координату y_1. Для отрезка AE это дает $y_1 > y_{\max}$, так что отрезок должен быть отброшен без дальнейших вычислений. Для отрезка BG будет выполняться $y_1 < y_{\max}$, поэтому он входит в окно с левой стороны и,

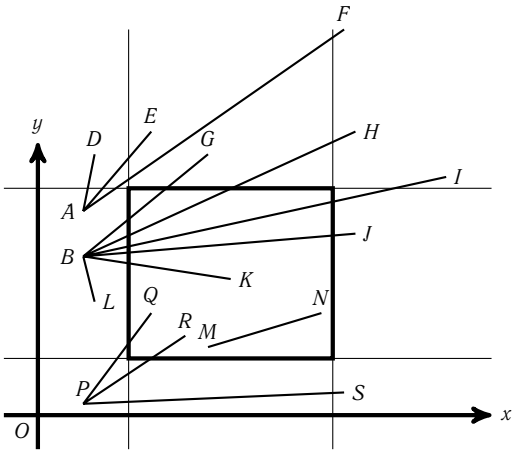


Рис. 5.11. Варианты расположения отрезков

следовательно, выходит через верхнюю границу. Следовательно, следующее отсечение должно быть Clip2Top, после которого отрезок принимается;

Код отрезка $0x1A$ (рис. 5.11, отрезки AF, BH и BI) Отрезки точно пересекают левую границу области видимости, поэтому вначале надо выполнить Clip1Left. После проверки координаты y_1 отрезок AF будет отброшен. Отрезки точно пересекают верхнюю границу области видимости. Поэтому следующее отсечение — Clip2Top, после которого проверяется координата x_2. Для отрезка BH это дает $x_2 < x_{max}$, поэтому он принимается. Для отрезка BI будет выполняться $x_2 > x_{max}$, поэтому он выходит из окна видимости с правой стороны и, следовательно, нужно выполнить Clip2Right, после чего отрезок принимается;

Код отрезка $0x10$ (рис. 5.11, отрезок BK) Отрезки точно пересекают левую границу области видимости, поэтому вначале надо выполнить Clip1Left, после чего отрезок принимается;

Код отрезка $0x12$ (рис. 5.11, отрезок BJ) Сначала надо выполнить Clip1Left, затем Clip2Right, после чего отрезок принимается;

Код отрезка $0x55$ (рис. 5.12, отрезок X_1Y) Случай простого отбрасывания;

Код отрезка $0x56$ (рис. 5.11, отрезок PS) Случай простого отбрасывания;

Код отрезка $0x50$ (рис. 5.11, отрезки PQ и PR) Вначале надо выполнить Clip1Left

и проверить значение y_1. Для отрезка PQ будет выполняться $y_1 > y_{\min}$, этот отрезок просто принимается. Для отрезка PR будет $y_1 < y_{\min}$. Для него нужно выполнить Clip1Bottom и принять отрезок;

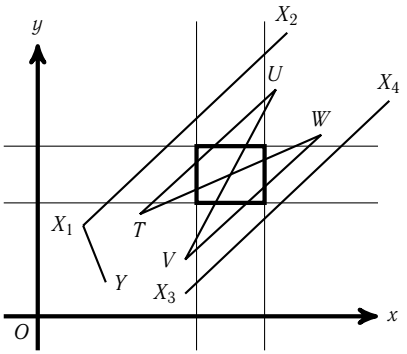


Рис. 5.12. Варианты расположения отрезков

Код отрезка $0x5A$ (рис. 5.12, все отрезки, кроме X_1Y) Вначале надо выполнить Clip1Left и проверить значение y_1. Для отрезка X_1X_2 будет выполняться $y_1 > y_{\max}$, этот отрезок отбрасывается. Для отрезков VU, V_W, X_3X_4 будет $y_1 < y_{\min}$. Для них нужно выполнить Clip1Bottom и проверить x_1. У отрезка X_3X_4 будет выполняться $x_1 > x_{\max}$, этот отрезок отбрасывается. Остальные отрезки должны быть видимы. Для них выполняем сначала Clip2Top и проверяем значение x_2. Для отрезков TU и VU будет выполняться $x_2 < X_{\max}$, они принимаются. Для остальных отрезков нужно выполнить Clip2Right, после чего принять.

Для остальных случаев достаточно либо продублировать условия обработки, либо провести предварительное преобразование поворота отрезка (вместе с окном отсечения) и/или его зеркального отражения для применения одного из вышеперечисленных случаев рассмотрения с последующим обратным преобразованием. Например для того, чтобы отсечь отрезок с кодом $0x4A$ можно повернуть систему координат на 90 градусов против часовой стрелки, после чего провести зеркальное отражение системы относительно оси Oy. В результате получим новые координаты концов отрезка

$$x'_1 = y_1; \qquad y'_1 = x_1;$$
$$x'_2 = y_2; \qquad y'_2 = x_2.$$

При этом область видимости изменит значения своих параметров:

$$x'_{\min} = y_{\min}; \qquad y'_{\min} = x_{\min};$$
$$x'_{\max} = y_{\min}; \qquad y'_{\max} = x_{\max}.$$

В новой системе координат отрезок будет иметь код $0x5A$ и его можно отсечь по уже предложенному алгоритму. После чего, если отрезок не будет отброшен, нужно сделать обратное преобразование для результата отсечения.

Подобную последовательность действий для сведения случая расположения отрезка к одному из перечисленных можно найти для каждого кода отрезка.

5.1.6. Алгоритм Николь—Ли—Николь

Алгоритм Николь—Ли—Николь подобен FC-алгоритму в том, что в нем перебираются всевозможные случаи расположения отрезков и области видимости, и для каждого случая определяется последовательность действий для отсечения. Причем для отсечения любого отрезка находятся максимум две точки пересечения.

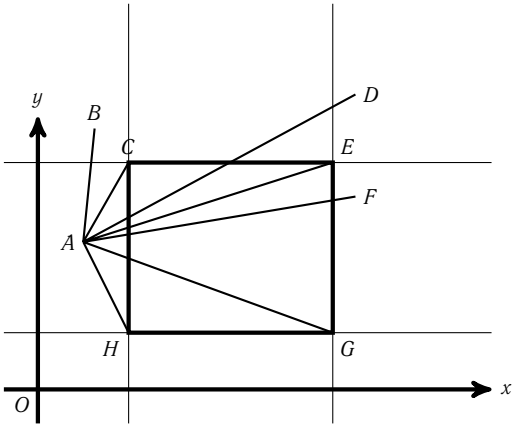


Рис. 5.13. Варианты расположения отрезков

Рассмотрим, например, точку A — начало отрезков, находящуюся левее области видимости (рис. 5.13). Отрезок, исходящий из точки A может быть частично видим, если угол его наклона находится в диапазоне от угла наклона отрезка AH до угла наклона отрезка AC. Точки H и C — соответственно точки левого нижнего и левого верхнего угла области видимости. Так, угол наклона отрезка AB больше, чем угол

наклона AC, то можно сделать вывод, что AB невидим и его отбросить. По аналогичным соображениям можем выявить, что так как угол наклона AD меньше чем угол наклона AC, но больше, чем угол наклона AE, и из-за того, что значение координаты x точки D больше $x_{\max}$, отрезок AD может выходить из области видимости только через верхнюю границу.

Вместо угла наклона целесообразнее сравнивать тангенс угла наклона $(y_2-y_1)/(x_2-x_1)$ отсекаемого отрезка и тангенс угла наклона отрезка, соединяющего точку начала отсекаемого отрезка с углом области видимости, $(y_a-y_1)/(x_a-x_1)$ (через (x_a, y_a) здесь обозначаем координаты некоторой угловой точки области видимости). Чтобы исключить необходимость деления, сравнение тангенсов угла наклона лучше привести к сравнению величин $(y_2-y_1)(x_a-x_1)$ и $(y_a-y_1)(x_2-x_1)$.

Так же как и в FC-алгоритме рассмотрим основные случаи работы алгоритма Николь—Ли—Николь.

Будем использовать следующие обозначения.

- (x_1, y_1), (x_2, y_2) — координаты точки начала и конца отрезка соответственно;
- $\Delta x = x_2 - x_1$, $\Delta y = y_2 - y_1$;
- $\Delta x_L = x_{\min} - x_1$, $\Delta x_R = x_{\max} - x_1$, $\Delta y_B = y_{\min} - y_1$, $\Delta y_T = y_{\max} - y_1$;
- Clip1Left, Clip1Bottom обозначают алгоритмы переноса начальной точки отрезка в точку пересечения отрезка с левой или нижней границей соответственно;
- Clip2Right, Clip2Top обозначают алгоритмы переноса конечной точки отрезка в точку пересечения отрезка с правой или верхней границей соответственно.

Пусть точка начала отрезка лежит не правее и не выше точки конца отрезка, т. е.

$$\Delta x \geqslant 0; \tag{5.12}$$
$$\Delta y \geqslant 0. \tag{5.13}$$

Тогда

если $x_2 < x_{\min}$, $x_1 > x_{\max}$ отрезок полностью невидим, случай простого отбрасывания.
если $y_2 < y_{\min}$, $y_1 > y_{\max}$ отрезок полностью невидим, случай простого отбрасывания.
если $x_1 \geqslant x_{\min}$, $x_2 \leqslant x_{\max}$, $y_1 \geqslant y_{\min}$, $y_2 \leqslant y_{\max}$ отрезок видим, случай простого принятия.

В остальных случаях отрезок полностью или частично лежит за пределами области видимости. Рассмотрим семейство случаев расположения отрезка, когда его начало лежит левее области видимости.

Пусть для точки начала отрезка выполняется $x_1 < x_{\min}$, $y_{\min} \leqslant y_1 \leqslant y_{\max}$ (см. рис. 5.14, точка A). Тогда, из (5.12) и (5.13) следует, что отрезок располагается в од-

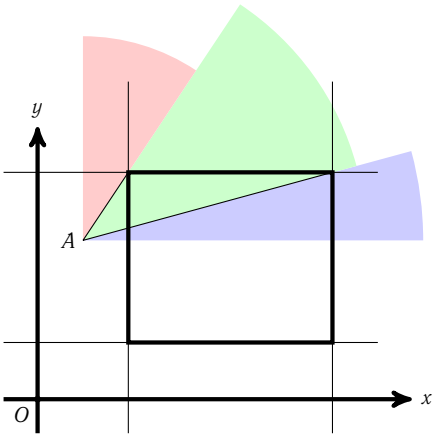


Рис. 5.14. Начальная точка и регионы расположения отсекаемого отрезка. Случай боковой области

ной из трех областей, отмеченных цветом на рисунке 5.14. Области разграничиваются прямыми, на которых лежат отрезки соединяющие точку (x_1, y_1) с точками $(x_{\min}, y_{\max})$ и $(x_{\max}, y_{\max})$ (см. рисунок).

На рисунке 5.15 изображены примеры расположения отрезков, начинающихся слева от области видимости.

Для того, чтобы определить область, в которую попадает отрезок, достаточно проверить:

если $\Delta y \Delta x_L > \Delta y_T \Delta x$ то отрезок попадает в розовую область: он полностью невидим (рис. 5.15, отрезок AB). Случай отбрасывания;

иначе, если $\Delta y \Delta x_R > \Delta y_T \Delta x$ то отрезок попадает в зеленую область: он частично видим (рис. 5.15, отрезки AD и AF). Он точно пересекает прямую, ограничивающую область видимости слева. Выполняем CLIP1LEFT. Если верно условие $y_2 \leqslant y_{\max}$, то полученный отрезок принимается (отрезок AF). В противном

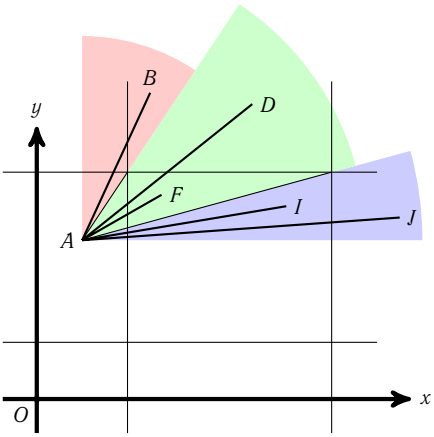


Рис. 5.15. Варианты расположения отсекаемых отрезков. Случай боковой области

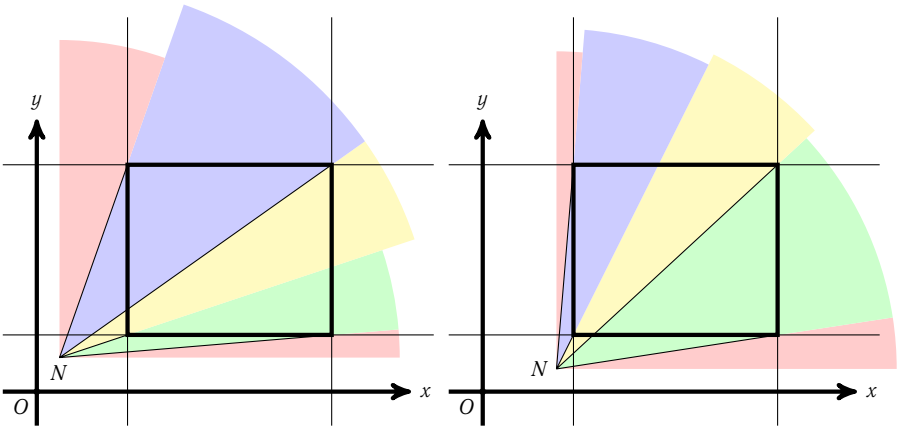
случае отрезок выходит из области видимости через верхнюю границу (отрезок AD). Выполняем Clip2Top и принимаем отрезок.

иначе отрезок попадает в голубую область: он частично видим (рис. 5.15, отрезки AI и AJ). Он точно пересекает прямую, ограничивающую область видимости слева. Выполняем Clip1Left. Если верно условие $x_2 \leqslant x_{\max}$, то полученный отрезок принимается (отрезок AI). В противном случае отрезок выходит из области видимости через правую границу (отрезок AJ). Выполняем Clip2Right и принимаем отрезок.

Если точка начала отрезка лежит в нижней левой угловой области, т. е. выполняется $x_1 < x_{\min}$, $y_1 < y_{\min}$ (см. рис. 5.16, точка N) то имеем пять регионов расположения отрезка, образованных, как и ранее, с помощью прямых соединяющих точку (x_1, y_1) с углами области видимости. Области, закрашенные на рисунке в розовый цвет, задают варианты расположения отрезка, при которых он отбрасывается. Если отрезок попадает в один из остальных регионов, то к нему нужно применить максимум два отсечения, чтобы он стал видим. Разграничение областей зависит от расположения начальной точки отрезка относительно прямой проходящей через точки $(x_{\min}, y_{\min})$ и $(x_{\max}, y_{\max})$. Это расположение можно определить, проверив неравенство

$$\Delta y_B \Delta x_R \leqslant \Delta y_T \Delta x_L.$$

Если неравенство выполняется, значит начальная точка отрезка лежит выше прямой и

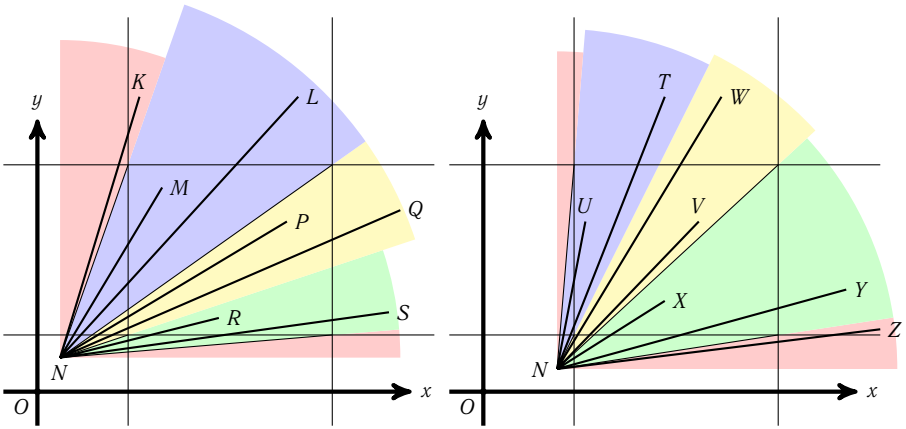


(*а*). $\Delta y_B \Delta x_R \leqslant \Delta y_T \Delta x_L$ (*б*). $\Delta y_B \Delta x_R > \Delta y_T \Delta x_L$

Рис. 5.16. Начальная точка и регионы расположения отсекаемого отрезка. Случай угловой области

имеет место случай, показанный на рисунке 5.16, *а*. В противном случае — начальная точка лежит ниже прямой и имеет место случай, показанный на рисунке 5.16, *б*.

На рисунке 5.17 приведены примеры расположения отрезков, начинающихся в нижней левой угловой области.



(*а*). $\Delta y_B \Delta x_R \leqslant \Delta y_T \Delta x_L$ (*б*). $\Delta y_B \Delta x_R > \Delta y_T \Delta x_L$

Рис. 5.17. Варианты расположения отсекаемых отрезков. Случай угловой области

Порядок отсечения отрезка может проводиться следующим образом:

если $\Delta y \Delta x_L > \Delta y_T \Delta x$ то отрезок попадает в левую розовую область: он полностью невидим (рис. 5.17, *а*, отрезок NK). Случай отбрасывания;

иначе, если $\Delta y \Delta x_R < \Delta y_B \Delta x$ то отрезок попадает в нижнюю розовую область: он полностью невидим (рис. 5.17, *б*, отрезок NZ). Случай отбрасывания;

иначе, если $\Delta y_B \Delta x_R \leqslant \Delta y_T \Delta x_L$

если $\Delta y \Delta x_R > \Delta y_T \Delta x$ то отрезок попадает в голубую область: он частично видим (рис. 5.17, *а*, отрезки NM и NL). Он точно пересекает прямую, ограничивающую область видимости слева. Выполняем Clip1Left. Если верно условие $y_2 \leqslant y_{\max}$, то полученный отрезок принимается (отрезок NM). В противном случае отрезок выходит из области видимости через верхнюю границу (отрезок NL). Выполняем Clip2Top и принимаем отрезок.

иначе, если $\Delta y \Delta x_L > \Delta y_B \Delta x$ отрезок попадает в желтую область: он частично видим (рис. 5.17, *а*, отрезки NP и NQ). Он точно пересекает прямую, ограничивающую область видимости слева. Выполняем Clip1Left. Если верно условие $x_2 \leqslant x_{\max}$, то полученный отрезок принимается (отрезок NP). В противном случае отрезок выходит из области видимости через правую границу (отрезок NQ). Выполняем Clip2Right и принимаем отрезок.

иначе отрезок попадает в зеленую область: он частично видим (рис. 5.17, *а*, отрезки NR и NS). Он точно пересекает прямую, ограничивающую область видимости снизу. Выполняем Clip1Bottom. Если верно условие $x_2 \leqslant x_{\max}$, то полученный отрезок принимается (отрезок NR). В противном случае отрезок выходит из области видимости через правую границу (отрезок NS). Выполняем Clip2Right и принимаем отрезок.

иначе (если $\Delta y_B \Delta x_R > \Delta y_T \Delta x_L$**)**

если $\Delta y \Delta x_L > \Delta y_B \Delta x$ то отрезок попадает в голубую область: он частично видим (рис. 5.17, *б*, отрезки NU и NT). Он точно пересекает прямую, ограничивающую область видимости слева. Выполняем Clip1Left. Если верно условие $y_2 \leqslant y_{\max}$, то полученный отрезок принимается (отрезок

NU). В противном случае отрезок выходит из области видимости через верхнюю границу (отрезок NL). Выполняем Clip2Top и принимаем отрезок.

иначе, если $\Delta y \Delta x_R > \Delta y_T \Delta x$ отрезок попадает в желтую область: он частично видим (рис. 5.17, *б*, отрезки NV и NW). Он точно пересекает прямую, ограничивающую область видимости снизу. Выполняем Clip1Bottom. Если верно условие $y_2 \leqslant y_{\max}$, то полученный отрезок принимается (отрезок NV). В противном случае отрезок выходит из области видимости через верхнюю границу (отрезок NW). Выполняем Clip2Top и принимаем отрезок.

иначе отрезок попадает в зеленую область: он частично видим (рис. 5.17, *б*, отрезки NX и NY). Он точно пересекает прямую, ограничивающую область видимости снизу. Выполняем Clip1Bottom. Если верно условие $x_2 \leqslant x_{\max}$, то полученный отрезок принимается (отрезок NX). В противном случае отрезок выходит из области видимости через правую границу (отрезок NY). Выполняем Clip2Right и принимаем отрезок.

Для остальных случаев расположение отсекаемого отрезка и области видимости можно составить подобные условия обработки, либо провести предварительный набор преобразований поворота и зеркального отображения отрезка (вместе с окном отсечения) для применения одного из вышеперечисленных случаев рассмотрения с последующим обратным преобразованием.

5.1.7. Проверка многоугольника на выпуклость и нахождение вектора внутренней нормали

Проверка на выпуклость может производиться анализом знаков векторных произведений смежных ребер, в порядке обхода (рис. 5.18).

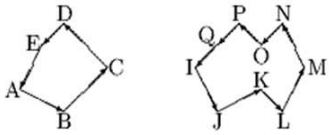


Рис. 5.18. Проверка многоугольника на выпуклость.

Если знак векторного произведения равен 0, то вершина вырождена, т. е. смежные ребра лежат на одной прямой (см. рис. 5.18, вершина Q).

Если все знаки равны 0, то многоугольник отсечения вырождается в отрезок.

Если же векторные произведения имеют разные знаки, то многоугольник невыпуклый (см. рис. 5.18, многоугольник $IJKLMNOPQ$).

Если все знаки неотрицательные, то многоугольник выпуклый, причем обход вершин выполняется против часовой стрелки (см. рис. 5.18, многоугольник $ABCDE$), т. е. внутренние нормали ориентированы влево от контура. Следовательно вектор внутреннего перпендикуляра к стороне к стороне может быть получен поворотом ребра на $90°$ (в реализации алгоритма вычисления нормалей на самом деле вычисляется не нормаль к стороне, а перпендикуляр, так как при вычислении значения t по соотношению (5.11) длина не важна).

Если все знаки неположительные, то многоугольник выпуклый, причем обход вершин выполняется по часовой стрелке, т.е. внутренние нормали ориентированы вправо от контура. Следовательно вектор внутреннего перпендикуляра к стороне может быть получен поворотом ребра на $-90°$.

5.2. Алгоритмы отсечения многоугольников

Многоугольники особенно важны в растровой графике как средство задания поверхностей. Будем называть многоугольник, используемый в качестве окна отсечения, отсекателем, а многоугольник, который отсекается, — отсекаемым.

Алгоритм отсечения многоугольника должен в результате отсечения давать один или несколько замкнутых многоугольников (рис. 5.19). При этом могут быть добав-

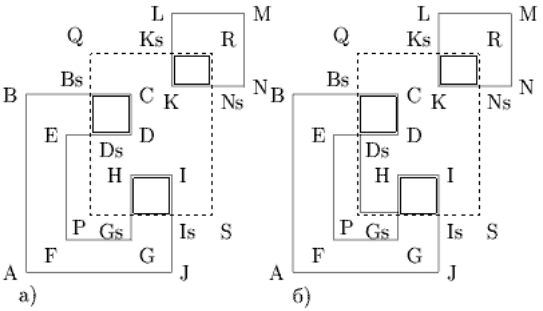


Рис. 5.19. Получение результата отсечения многоугольников.

лены новые ребра, а имеющиеся или сохранены или разделены или даже отброшены. Существенно, чтобы границы окна, которые не ограничивают видимую часть отсекаемого многоугольника, не входили в состав результата отсечения. Если это не выполняется, то возможна излишняя закраска границ окна

В принципе задачу отсечения многоугольника можно решить с использованием рассмотренных выше алгоритмов отсечения линий, если рассматривать многоугольник просто как набор векторов, а не как сплошные закрашиваемые области. При этом отрезки, составляющие многоугольник, просто последовательно отсекаются сторонами окна (рис. 5.20).

Если же в результате отсечения должен быть получен замкнутый многоугольник, то формируется вектор, соединяющий последнюю видимую точку с точкой пересечения с окном (рис. 5.21, а). Проблема возникает при окружении отсекаемым многоугольником угла окна (см. 5.21, б).

Здесь мы рассмотрим пять алгоритмов корректно решающие задачу отсечения

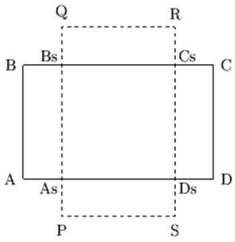


Рис. 5.20. Отсечение набора отрезков.

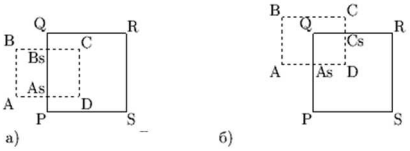


Рис. 5.21. Отсечение набора отрезков.

сплошного многоугольника. Первые четыре алгоритма быстро работают, но генерируют лишние ребра, как это продемонстрировано на рис. 5.19, б. Последний алгоритм свободен от указанного недостатка.

В общем, при отсечении многоугольников возникают два типа задач — отображение части изображения попавшей в окно и наоборот, отображение изображения, находящегося вне окна. Все здесь рассматриваемые алгоритмы могут использоваться в обоих случаях.

5.2.1. Алгоритмы поэтапного отсечения многоугольника ребрами области видимости

Следующие два алгоритма имеют в своей основе один и тот же метод. Область видимости в этих алгоритмах — выпуклый многоугольник.

Как отсекаемый многоугольник, так и многоугольник области видимости, представляются списком вершин в порядке некоторого обхода. В дальнейшем будем считать, что обход осуществляется по часовой стрелке.

Многоугольник последовательно отсекается каждой границей окна, как это показано на рис. 5.22

На каждом этапе отсечения многоугольник отсекается одним ребром выпуклого

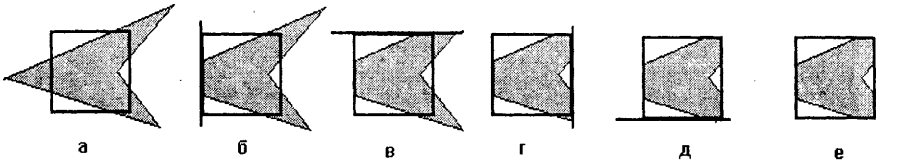


Рис. 5.22. Поэтапное отсечение многоугольника.

окна отсечения. В результате такого отсечения формируется новый многоугольник, который затем отсекается следующим ребром и т. д., пока не будет выполнено отсечение последним ребром окна.

Основная процедура здесь — процедура отсечения отдельным ребром. При выполнении этой процедуры последовательно просматривается каждое ребро многоугольника и решается вопрос о том, какая часть этого ребра будет записана в список многоугольника—результата отсечения.

Возможны 9 различных случаев расположения отсекающего ребра окна и отсекаемой стороны, показанных на рис. 5.23–5.25.

Рис. 5.23. Начальная точка на стороне внешности i-го ребра OB

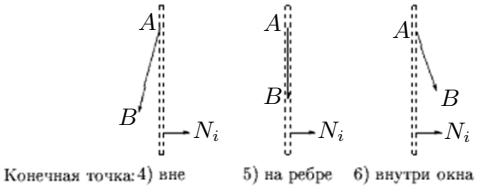


Рис. 5.24. Начальная точка на несущей прямой i-го ребра OB

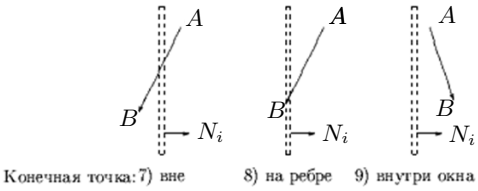


Рис. 5.25. Начальная точка на стороне внутренности i-го ребра OB

Здесь, A и B — начальная и конечная точки отсекаемой стороны многоугольника, соответственно; N_i — нормаль к i-му ребру окна отсечения, направленная внутрь окна.

Из этих рисунков очевиден результат, формирующий список нового многоугольника, зависящий от случая взаимного расположения:

1) Нет выходных данных.
2) В выходные данные заносится конечная точка.
3) Рассчитывается пересечение и в выходные данные заносятся точка пересечения и конечная точка.
4) Нет выходных данных.
5) В выходные данные заносится конечная точка.
6) В выходные данные заносится конечная точка.
7) Рассчитывается пересечение и в выходные данные заносится только точка пересечения.
8) В выходные данные заносится конечная точка.
9) В выходные данные заносится конечная точка.

Общая схема рассматриваемых методов — алгоритма Сазерленда—Ходгмана и алгоритма Кируса—Бека, представлена алгоритмом 8. Данные алгоритмы различаются только способами определения того, какой случай из девяти вышеперечисленных случаев имеет место. В алгоритме 8 эти различия будут иметь место в шагах PosA и PosB. Кроме этого, в алгоритмах разные подходы к поиску точки пересечения на шаге FindC.

Алгоритм 8: Алгоритм Сазерленда—Ходгмана / Кируса—Бека. Общая схема

Вход: список S многоугольников.
Выход: S_1 — список видимых частей многоугольников из S.
начало алгоритма
  Пусть F_i — все вершины области видимости, $1 \leqslant i \leqslant m$;
  $S_1 = \varnothing$;
  цикл пока S *не пуст* **выполнять**
    Взять из S список многоугольника P; n — количество вершин в P;
    Присвоить $i = 1$;
    цикл пока $i \leqslant m$ **выполнять**
      $k = 1$; $A = P_n$;
PosA Определить положение точки A относительно $i-$ границы;
      $P' = \varnothing$; $n_1 = 0$;
      цикл пока $k \leqslant n$ **выполнять**
        $B = P_k$;
PosB   Определить положение точки B относительно $i-$ границы;
        если *AB пересекается с несущей прямой i-ой границы* **то**
FindC   Найти точку пересечения C;
          Занести точку пересечения C в P'; $n_1 = n_1 + 1$;
        если *точка B на i-ой границе или на стороне внутренности* **то**
          Занести точку B в P'; $n_1 = n_1 + 1$;
        Присвоить $A = B$;
      Присвоить $P = P'$; $n = n_1$;
    если $P \neq \varnothing$ **то** положить P в список S_1;
  Выдать S_1;
конец алгоритма

Алгоритм Сазерленда—Ходгмана

Для определения случая расположения ребра многоугольника и прямой, ограничивающей область видимости, используется псевдоскалярное произведение вектора F_iF_{i+1}, проведенного из начальной в конечную точку текущего ребра области видимости, на вектор F_iA из начальной точки текущего ребра окна в очередную вершину A многоугольника (рис. 5.26). Обозначим через $Q_i(A)$ такое произведение.

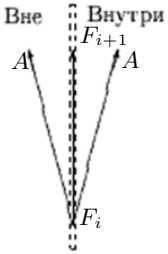


Рис. 5.26. Расположение точки относительно границы.

Если ребра многоугольников обходятся по часовой стрелке (в левой системе координат), то если $Q_i(A) < 0$, — точка A находится на стороне внутренности от i-го ребра области видимости. Если $Q_i(A) = 0$, то точка A находится на несущей прямой i-го ребра.

Для определения, какой из 9-ти случаев имеет место для отрезка AB, нужно найти $Q_i(A)$, $Q_i(B)$ и оценить их знаки.

Подход к нахождению точки пересечения в этом алгоритме такой же, как в алгоритме Скала: находим параметр t для точки пересечения:

$$t = \frac{AF_i \times F_iF_{i+1}}{(B-A) \times F_iF_{i+1}} =$$

$$\frac{AF_i \times F_iF_{i+1}}{(B-A+F_i-F_i) \times (F_iF_{i+1})} = \frac{AF_i \times F_iF_{i+1}}{((F_i-A) \times F_iF_{i+1}) - ((F_i-B) \times F_iF_{i+1})} =$$

$$\frac{AF_i \times F_iF_{i+1}}{(AF_i \times F_iF_{i+1}) - (BF_i \times F_iF_{i+1})} = \frac{Q_i(A)}{Q_i(A) - Q_i(B)},$$

после чего, из параметрического уравнения отрезка AB находим соответствующую точку.

Простой алгоритм отсечения многоугольника (Алгоритм Кируса—Бека)

Данный алгоритм использует те же подпрограммы обработки многоугольного окна отсечения, что и алгоритм Кируса — Бека для отсечения отрезков.

Для определения взаимного расположения, подобно алгоритму отсечения Кируса — Бека, используется скалярное произведение вектора нормали N_i на вектор, проведенный из начала ребра в анализируемую точку.

Таким образом, для определения взаимного расположения начальной A и конечной B точек отсекаемой стороны и ребра отсечения с вектором его начала F_i, надо вычислить:

$$Q_i(0) = (A - F_i)N_i$$

$$Q_i(1) = (B - F_i)N_i.$$

Расчет пересечения, если он требуется, производится аналогично алгоритму Кируса — Бека с использованием параметрического представления линии.

Из равенства 5.10 следует

$$(B - A - F_i + F_i)N_i t - (F_i - A)N_i = 0,$$

откуда

$$((B - F_i)N_i - (A - F_i)N_i)t + (A - F_i)N_i = 0,$$

или, в вышеприведенных обозначениях,

$$(Q_i(1) - Q_i(0))t + Q_i(0) = 0.$$

Таким образом, точке пересечения соответствует значение параметра t, равное:

$$t = \frac{Q_i(0)}{Q_i(0) - Q_i(1)}.$$

Значения координат пересечения находятся из параметрического уравнения отрезка.

5.2.2. Однопроходные алгоритмы отсечения многоугольников

В следующих двух алгоритмах производится отсечение многоугольников относительно прямоугольного окна. В таком случае результат отсечения должен представлять из себя список многоугольника, состоящий из некоторого набора вершин исходного многоугольника, некоторого количества точек пересечения ребер многоугольника

со сторонами области видимости и, возможно, точек—углов области видимости, — так называемых, поворотных точек.

Весь процесс отсечения заключается в одном проходе исходного списка многоугольника, в котором проводится анализ каждого ребра, и в список результата заносятся соответствующие точки. В ходе анализа ребро отсекается. Если после отсечения остаются видимые части, то в список результата заносятся соответствующие вершины и точки пересечения. В любом случае делается заключение о необходимости добавления в результат одной или двух поворотных точек.

Анализ на добавление поворотных точек может осуществляться разными способами. На рисунке 5.27 изображены примеры возможного расположения ребер многоугольника

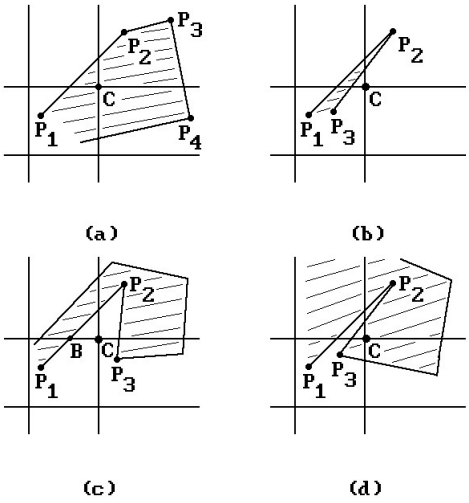


Рис. 5.27. Анализ отрезка на предмет добавления поворотной точки

и точки угла окна. В тех алгоритмах, в которых проводится только один проход многоугольника, в течении этого прохода невозможно предсказать, будет ли многоугольник охватывать угол или нет (например, на рисунке 5.27, анализируя только лишь отрезок P_1P2 невозможно однозначно решить вопрос о добавлении точки C в результат). Поэтому в этих алгоритмах, в целях ускорения работы, вопрос о добавлении поворотной точки решается следующим образом: поворотная точка добавляется если рассматриваемое ребро многоугольника проходит через соответствующую угловую область за пределами области видимости (т.е., для всех примеров на рисунке 5.27 поворотная

точка будет добавлена). Таким образом, в результат могут быть добавлены лишние точки углов окна видимости, что может привести к появлению в многоугольнике—результате областей, вырожденных в отрезок — побочный эффект, который можно легко устранить с помощью дополнительной обработки.

Алгоритм Лианга—Барски

Основная идея алгоритма заключается в следующем. Считаем, что область видимости ограничивают четыре прямые. Две прямые ограничивают область видимости по x и две прямые — по y. Рассматриваемому отрезку—ребру отсекаемого многоугольника придается направление (от начальной точки до конечной). Если ребро не параллельно никакой из прямых, ограничивающих область видимости, то найдутся четыре точки пересечения несущей прямой этого ребра с продолженными границами окна. Пусть для этих точек пересечения найдены параметры t параметрического уравнения отрезка. Причем относительно двух границ отрезок направлен в сторону внутренности области видимости. Обозначим через $t_{1\,in}$ и $t_{2\,in}$ (см. рис. 5.28) параметры для точек пересечения с этими границами (будем называть эти точки «входящие точки пересечения»). Относительно двух других границ отрезок направлен в сторону внешности. Обозначим параметры для этих точек пересечения (для «выходящих точек пересечения») через $t_{1\,out}$ и $t_{2\,out}$ (см. рис. 5.28). Анализируя величины этих парамет-

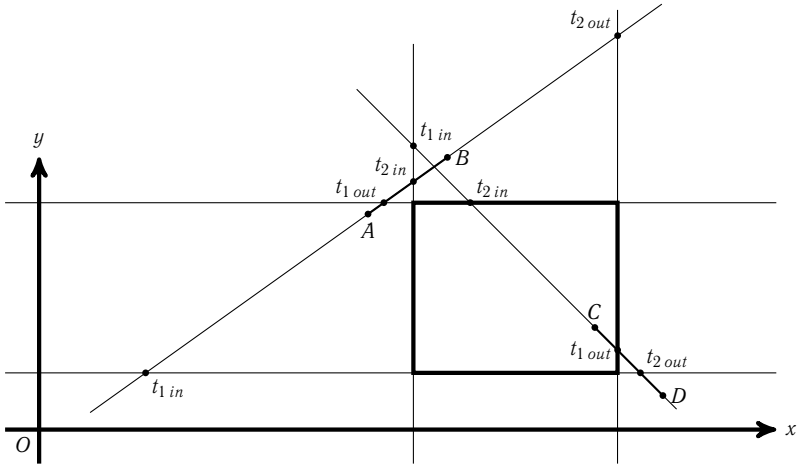


Рис. 5.28. Параметры для точек пересечения несущих прямых с границами окна

ров делается вывод о расположении соответствующих точек пересечения и самого отрезка в целом относительно области видимости.

Если упорядочить по возрастанию параметры для этих четырех точек пересечения, то минимальным параметром будет параметр для одной из входящих точек пересечения. Пусть это будет $t_{1\,in}$. Последний параметр в упорядоченной последовательности — параметр для одной из выходящих точек пересечения. Будем считать, что это $t_{2\,out}$. Параметр $t_{2\,in}$ может быть меньше $t_{1\,out}$ (в этом случае несущая прямая отрезка проходит через область видимости. См. рис. 5.28, отрезок CD), а может быть больше, чем $t_{1\,out}$ (несущая прямая не проходит через область видимости, а следовательно, ребро отсекаемого многоугольника полностью невидимо. См. рис. 5.28, отрезок AB). В первом случае вопрос о видимости или частичной видимости ребра решается сравнением параметров $t_{2\,in}$ и $t_{1\,out}$ с нулем и единицей (параметрами для видимой части отрезка). Если отрезок пересекает одну из границ области видимости или обе границы, то точки пересечения для соответствующих параметров заносятся в список вершин результата, а если отрезок заканчивается в области видимости ($t_{2\,in} \leqslant 1 \leqslant t_{1\,out}$), то в результат заносится и конечная точка отрезка. Во втором случае точек пересечения нет, но отрезок может проходить насквозь через угловую область за пределами окна наблюдения (не начинаться и не заканчиваться в этой угловой области, а проходить насквозь), что выявляется сравнением тех же $t_{2\,in}$ и $t_{1\,out}$ с нулем и единицей. Если это так, то в результат заносится поворотная точка — соответствующая угловая точка области видимости. Наконец, и в случае видимости отрезка, и в случае его нахождения за пределами окна, проводится проверка на окончание отрезка в угловой области за пределами окна. Если конечная точка отрезка располагается в такой области, то в результат заносится соответствующая угловая точка окна.

Общая схема метода представлена в алгоритме 9.

Алгоритм 9: Отсечение многоугольника по Лиангу—Барски. Общая схема

Вход: список S многоугольников.
Выход: S_1 — список видимых частей многоугольников из S.
начало алгоритма
  $S_1 = \varnothing$;
  цикл пока S *не пуст* **выполнять**
    Взять из S список многоугольника P; n — количество вершин в P;
    $k = 1$; $A = P_n$;
    $P' = \varnothing$;
    цикл пока $k \leqslant n$ **выполнять**
      $B = P_k$;
      Определить направление отрезка AB;
      Вычислить $t_{1\,out}$, $t_{2\,out}$;
      если $0 < t_{2\,out}$ **то**
        Вычислить $t_{2\,in}$;
        если $t_{1\,out} < t_{2\,in}$ **то** *// отрезок полностью невидим*
          если $0 < t_{1\,out} \leqslant 1$ **то** *// отрезок заходит в угловую область*
            Добавить в P' точку соответствующего угла окна;
        иначе
          если $0 < t_{1\,out}$ *и* $t_{2\,in} \leqslant 1$ **то** *// отрезок видим*
            если $0 < t_{2\,in}$ **то**
              $A' = A + (B - A)t_{2\,in}$;
              Добавить A' в P';
            если $t_{1\,out} \leqslant 1$ **то**
              $B' = A + (B - A)t_{1\,out}$;
              Добавить B' в P';
            иначе Добавить B в P';
          если $t_{2\,out} \leqslant 1$ **то** *// отрезок заканчивается в угловой области*
            Добавить в P' точку соответствующего угла окна;
      $A = B$; $k = k + 1$;
    если $P \neq \varnothing$ **то** положить P в список S_1;
  Выдать S_1;
конец алгоритма

Алгоритм Мэл'от

Алгоритм Мэл'от (Maillot) является своеобразным продолжением алгоритма Коэна—Сазерленда для отсечения многоугольников. Автор метода предлагает дополнить код области при кодировании, введенном в алгоритме Коэна—Сазерленда, дополнительным разрядом (см. рис. 5.29). Дополнительный разряд будет получать значение «один» если область угловая. В противном случае — значение «ноль».

Для каждого ребра многоугольника проводится отсечение по алгоритму Коэна—Сазерленда обычным образом, без использования дополнительного разряда. Во время отсечения инициализируются два флага: `SEGM` — говорит о том, что после отсечения остались видимые части отрезка, `CLIP` — говорит о том, что начальная точка исход-

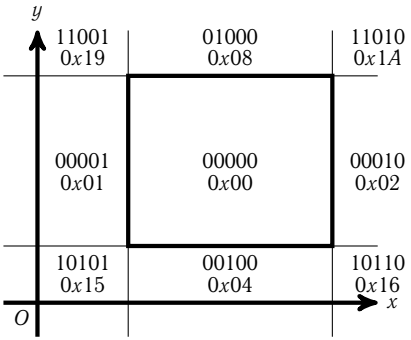


Рис. 5.29. Кодирование областей в алгоритме Мэл'от

ного отрезка была отсечена.

Если после отсечения ребра остались видимые части, то алгоритм Мэл'от проверяет значение флага `CLIP`. Если начальная точка была отсечена, то в результат заносится начальная точка видимой части. В любом случае в результат заносится конечная точка видимой части отрезка.

Если отрезок невидим, производится его анализ чтобы выявить, не пересекает ли отрезок угловую область. Это имеет место, если

1. начало и конец отрезка находятся в разных неугловых областях (рисунок 5.30, отрезок DF);
2. начало отрезка находится в неугловой, а конец отрезка в угловой области по разные стороны от окна видимости (рисунок 5.30, отрезок DE);
3. начало отрезка находится в угловой, а конец отрезка в неугловой области по разные стороны от окна видимости (рисунок 5.30, отрезок AC);
4. и начало и конец отрезка в угловых областях, расположенных по диагонали от окна видимости (рисунок 5.30, отрезки AB и GH).

Все эти случаи выявляются при рассмотрении расширенных кодов концов исходного отрезка. Более того, в первых трех случаях из кодов соответствующих однозначно определяется поворотная точка, которую нужно добавить в результат. В четвертом случае для выявления поворотной точки отрезок делится пополам. Если точка середины отрезка попала не в ту же угловую область, в которой находится один из концов отрезка, то поворотная точка определяется однозначно. В противном случае тест повторяется, но уже для половины отрезка.

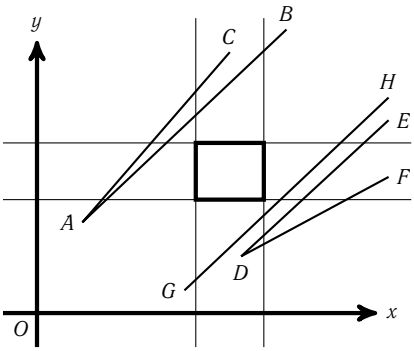


Рис. 5.30. Варианты пересечения отрезком угловой области

Когда поворотная точка возникает в результате наличия отрезка типа DF на рис. 5.30, код угловой области, соответствующий этой поворотной точке вычисляется как сумма кодов областей начала и конца отрезка с добавлением к нему 16 (заполнение единицей добавочного разряда).

Код угловой области, соответствующий поворотной точке, возникшей в результате наличия отрезков типа AC, DE на рис. 5.30, предлагается вычислять как сумму кода угловой области, в которой начинается/заканчивается отрезок, со значением элемента массива значений Tcc, в котором для кода боковой области хранится недостающая величина. Так для кодировки областей, представленной на рис. 5.29, массив Tcc будет состоять из следующих величин:

$$Tcc[0x01] = -1;\ Tcc[0x02] = 1;\ Tcc[0x04] = -4;\ Tcc[0x08] = 4.$$

Так, например, для вычисления кода области, соответствующей поворотной точке для отрезка AC нужно сложить код точки A со значением элемента Tcc для кода точки C:

$$0x15 + Tcc[0x08] = 0x15 + 4 = 0x19$$

После выполнения вышеперечисленных действий (в случае видимости отрезка и его невидимости) выполняется простой тест конечной точки исходного отрезка, чтобы выявить необходимость добавления поворотной точки в случае ее расположения в угловой области. Общая схема метода представлена в алгоритме 10. Здесь $ExtCode(A)$ обозначает функцию, возвращающую расширенный код области, в которую попала точка A, в соответствии с рисунком 5.29.

Алгоритм 10: Отсечение многоугольника по Мэл'от. Общая схема

Вход: список S многоугольников.
Выход: S_1 — список видимых частей многоугольников из S.
начало алгоритма
  $S_1 = \varnothing$;
  цикл пока S *не пуст* **выполнять**
    Взять из S список многоугольника P; n — количество вершин в P;
    $k = 1$; $A = P_n$; вычислить $C_{start} = ExtCode(A)$;
    $P' = \varnothing$; ;
    цикл пока $k \leqslant n$ **выполнять**
      $B = P_k$; вычислить $C_{end} = ExtCode(B)$;
      Выполнить алгоритм Коэна—Сазерленда. Получить значения `CLIP`, `SEGM`, и, возможно, точки A' и B';
      $C_2 = C_{end}$;
      если `SEGM` **то**
        если `CLIP` **то** добавить A' в P';
        Добавить B' в P';
      иначе
        если $C_{end} \& 0x10 \neq 0$ **то** *// конечная точка в угловой области*
          если $C_{start} \& C_{end} \& 0x0F = 0$ **то** *// начальная и конечная*
            // точки по разные стороны области видимости
            если $C_{start} \& 0x10 = 0$ **то** $C_1 = C_{end} + Tcc(C_{start})$;
            иначе
              $A' = A$; $B' = B$; $C_{11} = C_{start}$; $C_{12} = C_{end}$;
              $D = A$; $C_1 = C_{start}$;
              цикл пока $C_1 = C_{11}$ *или* $C_1 = C_{12}$ **выполнять**
                если $C_1 = C_{11}$ **то** $A' = D$ **иначе** $B' = D$;
                Вычислить D — точку середины отрезка $A'B'$;
                Вычислить $C_1 = ExtCode(D)$;
              если $C_1 \& C_{12} \neq 0$ **то** $C_1 = C_{11} + Tcc(C_1)$;
              иначе $C_1 = C_{12} + Tcc(C_1)$;
            Определить угол, соответствующий C_1. Добавить его в P';
          иначе
            если $C_{start} \& C_{end} \& 0x0F = 0$ **то**
              если $C_{start} \& 0x10 \neq 0$ **то** $C_2 = C_{start} + Tcc(C_{end})$;
              иначе $C_2 = C_{start} + C_{end} + 16$;
        если $C_2 \& 0x10 \neq 0$ **то**
          Определить угол, соответствующий C_2. Добавить его в P';
        $A = B$; $k = k + 1$; $C_{start} = C_{end}$;
      если $P \neq \varnothing$ **то** положить P в список S_1;
    Выдать S_1;
конец алгоритма

5.2.3. Алгоритм Вейлера — Азертона

Предыдущие алгоритмы отсечения многоугольника отсекают произвольный (как выпуклый, так и невыпуклый) многоугольник относительно выпуклого окна. Зачастую же требуется отсечение по невыпуклому окну. Кроме того, все рассмотренные алгоритмы могут генерировать лишние стороны для отсеченного многоугольника, проходящие вдоль ребра окна отсечения. Далее рассматриваемый алгоритм Вейлера — Азертона свободен от указанных недостатков ценой заметно большей сложности и меньшей скорости работы.

Предполагается, что каждый из многоугольников задан списком вершин, причем таким образом, что при движении по списку вершин в порядке их задания внутренняя область многоугольника находится справа от границы.

В случае пересечения границ и отсекаемого многоугольника и окна возникают точки двух типов:

- входные точки, когда ориентированное ребро отсекаемого многоугольника входит в окно,
- выходные точки, когда ребро отсекаемого многоугольника идет с внутренней на внешнюю стороны окна.

Общая схема алгоритма Вейлера — Азертона для определения части отсекаемого многоугольника, попавшей в окно, следующая:

1. Строятся списки вершин отсекаемого многоугольника и окна.
2. Отыскиваются все точки пересечения. При этом расчете касания не считаются пересечением, т.е. когда вершина или ребро отсекаемого многоугольника инцидентна или совпадает со стороной окна (рис. 5.31).

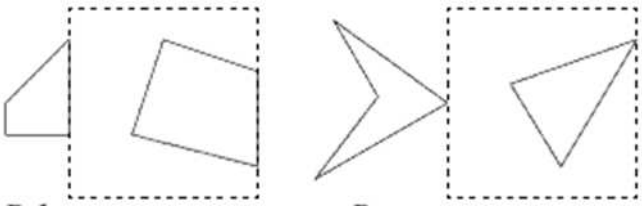


Рис. 5.31. Точки касания.

3. Списки координат вершин отсекаемого многоугольника и окна дополняются новыми вершинами — координатами точек пересечения. Причем если точка пересечения P_i находится на ребре, соединяющем вершины V_i, V_j, то последовательность точек V_i, V_j превращается в последовательность Vi, P_k, V_j. При этом устанавливаются двухсторонние связи между одноименными точками пересечения в списках вершин отсекаемого многоугольника и окна.

   Входные и выходные точки пересечения образуют отдельные подсписки входных и выходных точек в списках вершин.

4. Определение части обрабатываемого многоугольника, попавшей в окно выполняется следующим образом:

   а) Если не исчерпан список входных точек пересечения, то выбираем очередную входную точку. В противном случае завершаем алгоритм.

   б) Помечаем входную точку пересечения.

   в) Начиная от входной точки пересечения двигаемся по вершинам отсекаемого многоугольника пока не обнаружится следующая точка пересечения; все пройденные точки, не включая прервавшую просмотр, заносим в результат; используя двухстороннюю связь точек пересечения, переключаемся на просмотр списка вершин окна.

   г) Начиная от точки пересечения, прервавшей просмотр, двигаемся по вершинам области видимости до обнаружения следующей точки пересечения; все пройденные точки, не включая последнюю, прервавшую просмотр, заносим в результат.

   д) Используя двухстороннюю связь точек пересечения, переключаемся на список вершин обрабатываемого многоугольника.

   е) Если прервавшая просмотр точка пересечения помечена в списке вершин отсекаемого многоугольника, то завершаем построение списка вершин очередного отсекаемого многоугольника. После чего переходим к шагу а), в противном случае переходим к шагу б).

5.3. Трехмерные алгоритмы удаления скрытых линий и поверхностей

Методы удаления невидимых частей сцены можно классифицировать:

- По выбору удаляемых частей:
  - удаление невидимых линий,
  - удаление невидимых ребер,
  - удаление невидимых поверхностей,
  - удаление невидимых объемов.
- По порядку обработки элементов сцены:
  - удаление в произвольном порядке,
  - удаление в порядке, определяемом процессом визуализации.
- По системе координат:
  - алгоритмы работающие в пространстве объектов, когда каждая из n граней объекта сравнивается с остальными $n-1$ гранями (объем вычислений растет как n^2),
  - алгоритмы работающие в пространстве изображения, когда для каждого пиксела изображения определяется какая из n граней объекта видна (при разрешении экрана $m \times m$ объем вычислений растет как $m^2 \times n$).

5.3.1. Алгоритм Варнока (разбиения области)

Алгоритм работает в пространстве изображения и анализирует область на экране дисплея (окно) на наличие в них видимых элементов. Если в окне нет изображения, то оно просто закрашивается фоном. Если же в окне имеется элемент, то проверяется достаточно ли он прост для визуализации. Если объект сложный, то окно разбивается на более мелкие, для каждого из которых повторяется алгоритм. Рекурсивный процесс разбиения может продолжаться до тех пор пока не будет достигнут предел разрешения экрана.

Тесты на сложность визуализации элемента могут быть различными. Здесь приведем классическую схему алгоритма.

Можно выделить 4 случая взаимного расположения окна и многоугольника (рис. 5.32):

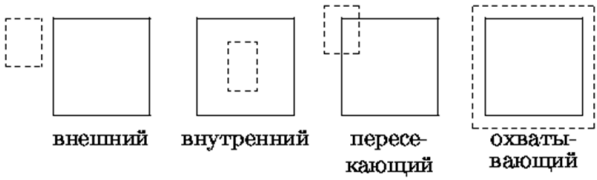


Рис. 5.32. Случаи расположение окна и многоугольника.

- многоугольник целиком вне окна,
- многоугольник целиком внутри окна,
- многоугольник пересекает окно,
- многоугольник охватывает окно.

В четырех случаях можно сразу принять решение о правилах закраски области экрана:

- все многоугольники сцены — внешние по отношению к окну. В этом случае окно закрашивается фоном;
- имеется всего один внутренний или пересекающий многоугольник. В этом случае все окно закрашивается фоном и затем часть окна, соответствующая внутреннему или пересекающему окну закрашивается цветом многоугольника;
- имеется единственный охватывающий многоугольник. В этом случае окно закрашивается его цветом.
- имеется несколько различных многоугольников и хотя бы один из них охватывающий. Если при этом охватывающий многоугольник расположен ближе остальных к наблюдателю, то окно закрашивается его цветом.

В любых других случаях процесс разбиения окна продолжается.

Легко видеть, что при разрешении 1024×1024 и делении стороны окна пополам требуется не более 10 разбиений. Если достигнуто максимальное разбиение, но не обнаружено ни одного из приведенных выше четырех случаев, то для точки с центром в полученном минимальном окне (размером в пиксел) вычисляются глубины оставшихся многоугольников и закраску определяет многоугольник, наиболее близкий к наблюдателю. При этом для устранения лестничного эффекта можно выполнить дополнительные разбиения и закрасить пиксел с учетом всех многоугольников, видимых в минимальном окне.

Первые три случая идентифицируются легко. Последний же случай фактически сводится к поиску охватывающего многоугольника, перекрывающего все остальные многоугольники, связанные с окном. Проверка на такой многоугольник может быть выполнена следующим образом: в угловых точках окна вычисляются z_s-координаты для всех многоугольников, связанных с окном. Если все четыре такие z_s-координаты охватывающего многоугольника ближе к наблюдателю, чем все остальные, то окно закрашивается цветом соответствующего охватывающего многоугольника. Если же нет, то мы имеем сложный случай и разбиение следует продолжить.

Очевидно, что после разбиения окна охватывающие и внешние многоугольники наследуются от исходного окна. Поэтому необходимо проверять лишь внутренние и пересекающие многоугольники.

Из изложенного ясно, что важной частью алгоритма является определение расположения многоугольника относительно окна.

Проверка на то что многоугольник внешний или внутренний относительно окна для случая прямоугольных окон легко реализуется использованием прямоугольной оболочки многоугольника и сравнением координат. Для внутреннего многоугольника должны одновременно выполняться условия:

$$\begin{cases} x_L \geqslant x_{\min}, \\ x_R \leqslant x_{\max}, \\ y_B \geqslant y_{\min}, \\ y_T \leqslant y_{\max}, \end{cases}$$

где x_L, x_R, y_B, y_T — ребра оболочки, $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$ — ребра окна.

Для внешнего многоугольника достаточно выполнение любого из следующих условий:

$$\begin{cases} x_L > x_{\max}, \\ x_R < x_{\min}, \\ y_B > y_{\max}, \\ y_T < y_{\min}, \end{cases}$$

Таким способом внешний многоугольник, охватывающий угол окна не будет идентифицирован как внешний (см. рис. 5.33).

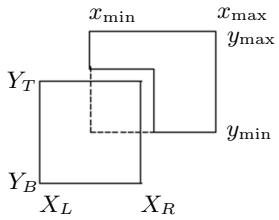


Рис. 5.33. Многоугольник, охватывающий угол.

Проверка на пересечение окна многоугольником может быть выполнена проверкой на расположение всех вершин окна по одну сторону от прямой, на которой расположено ребро многоугольника. Пусть ребро многоугольника задано точками $P_1 = (x_1, y_1, z_1)$ и $P_2 = (x_2, y_2, z_2)$, а очередная вершина окна задается точкой $P_3 = (x_3, y_3, z_3)$. Псевдоскалярное произведение вектора P_1P_3 на вектор P_1P_2, равное $(x_3 - x_1)(y_2 - y_1) - (y_3 - y_1)(x_2 - x_1)$ будет меньше 0, равно 0 или больше 0, если вершина лежит слева, на или справа от прямой P_1P_2. Если знаки различны, то окно и многоугольник пересекаются. Если же все знаки одинаковы, то окно лежит по одну сторону от ребра, т.е. многоугольник может быть либо внешним, либо охватывающим.

5.3.2. Алгоритм удаления поверхностей с Z-буфером

Алгоритм предложен Эдом Кэтмулом и представляет собой обобщение буфера кадра. Обычный буфер кадра хранит коды цвета для каждого пиксела в пространстве изображения. Идея алгоритма состоит в том, чтобы для каждого пиксела дополнительно хранить еще и координату z точки, изображенной в этом пикселе. При попытке изображения точки сцены в некотором пикселе растровой области сравнивается значение её z-координаты с z-координатой пиксела, находящейся в буфере. Если z-координата точки меньше, чем координата уже изображенной точки, т.е. она ближе к наблюдателю, то точка изображается и её z-координата заносятся в буфер, если нет, то ничего не делается.

Дополнительный массив, выполняющий роль буфера, в который заносятся координаты z изображенных точек называют Z-буфером.

Алгоритм с использованием Z-буфера наиболее простой из всех алгоритмов удаления невидимых поверхностей, но требует большого объема памяти. Например, если

необходимо иметь разрядность порядка 20 бит на 1 пиксел, то при изображении нормального телевизионного размера в 768×576 пикселов для хранения z_s-координат необходим объем памяти порядка 1 МБ, а суммарный объем памяти при 3 байтах для значений RGB составит более 2.3 МБ.

Время работы алгоритма зависит от сложности сцены. Многоугольники, составляющие сцену, могут обрабатываться в произвольном порядке. Для сокращения затрат времени нелицевые многоугольники могут быть удалены. По сути дела алгоритм с Z-буфером — некоторая модификация уже рассмотренного алгоритма заливки многоугольника. Если используется построчный алгоритм заливки, то легко сделать пошаговое вычисление z_s-координаты очередного пиксела, дополнительно храня z_s-координаты его вершин и вычисляя приращение Δz z_s-координаты при перемещении вдоль оси Ox_s на $\Delta x = 1$.

Общая схема алгоритма с Z-буфером:

1. Инициализировать растровую область и Z-буфер. Растровая область закрашивается фоном. Z-буфер заполняется максимальным значением z.
2. Для каждой строки растра

   - Пересматривается список активных ребер (*AEL*) на предмет сокращения или пополнения.
   - Для каждой пары активных ребер выполняется возможное заполнение строки растра и Z-буфера (процедура ShowLine, алгоритм 11).
   - Производится обработка списка активных ребер в целях подготовки к следующей строке растра (пересчитываются величины x и z для каждого активного ребра).

При переборе строк растра в порядке увеличения координаты y список активных ребер можно рассматривать как список пятерок: для каждого ребра, заданного координатами (x_1, y_1, z_1) и (x_2, y_2, z_2), где $y_1 < y_2$, в этом списке сохраняются величины $\left(x_1, z_1, y_2, \dfrac{x_2 - x_1}{y_2 - y_1}, \dfrac{z_2 - z_1}{y_2 - y_1}\right)$. Первые два элемента в этой пятерке — значения x и z для проекции ребра на текущую строку растра (первоначально x_1 и z_1). Третий элемент — значение координаты y, при котором данное ребро нужно удалить из списка активных ребер. Два последних элемента пятерки — приращения значений x и y при переходе к следующей строке растра (при увеличении y на 1).

В общем виде метод Z-буфера приведен в алгоритме 12 с использованием процедуры, сформулированной в алгоритме 11.

Алгоритм 11: Процедура SHOWLINE для метода Z-буфера

Вход: y_0 — координата y строки растра, (x_1, z_1) — координаты x и z первой точки, (x_2, z_2) — координаты x и z второй точки. C — цвет соответствующего многоугольника.
Выход: Измененные Z-буфер и область рисования.
начало алгоритма
  $x = x_1$; $z = z_1$; $\Delta z = \dfrac{z_2 - z_1}{x_2 - x_1}$;
  цикл пока $x \leqslant x_2$ **выполнять**
    если $z < Z[x, y_0]$, **то**
      Присвоить $Z[x, y_0] = z$ и закрасить в растровой области точку с координатами (x, y_0) цветом C;
    Присвоить $x = x + 1$, $z = z + \Delta z$;
конец алгоритма

Основной недостаток алгоритма с Z-буфером — дополнительные затраты памяти. Для их уменьшения можно разбивать изображение на несколько прямоугольников или полос. В пределе можно использовать Z-буфер в виде одной строки. Понятно, что это приведет к увеличению времени, так как каждый прямоугольник будет обрабатываться столько раз, на сколько областей разбито пространство изображения. Уменьшение затрат времени в этом случае может быть обеспечено предварительной сортировкой многоугольников на плоскости.

Другие недостатки алгоритма с Z-буфером заключаются в том, что так как пикселы в буфер заносятся в произвольном порядке, то возникают трудности с реализацией эффектов прозрачности или просвечивания и устранением лестничного эффекта с использованием предфильтрации, когда каждый пиксел экрана трактуется как точка конечного размера и его атрибуты устанавливаются в зависимости от того какая часть пиксела изображения попадает в пиксел экрана.

Алгоритм 12: Отсечение невидимых граней с использованием Z-буфера

Вход: $\mathscr{P}$ — список многоугольников трехмерной сцены.
начало алгоритма
- · Заполнить растровую область рисования цветом фона. Определить вещественнозначный двумерный массив Z с размерностями (и индексами элементов) области рисования;
- **цикл пока** *список* $\mathscr{P}$ *не пуст* **выполнять**
  - · Взять из списка $\mathscr{P}$ очередной многоугольник P с цветом C;
  - · Сформировать список S ребер многоугольника. Упорядочить список S по возрастанию значения y_1;
  - · Найти y_{min} и y_{max} — минимальное и максимальное значение координаты y точек вершин многоугольника;
  - · $AEL = \varnothing$, $y_t = y_{min}$, $y_{S\,next} = y_{min}$;
  - **цикл пока** $y_t \leqslant y_{max}$ **выполнять**
    - **если** $y_t = y_{S\,next}$, **то**
      - · Добавить в AEL все пятерки $\left(x_1, z_1, y_2, \dfrac{x_2 - x_1}{y_2 - y_1}, \dfrac{z_2 - z_1}{y_2 - y_1}\right)$, составленные для каждого отрезка из S, у которого $y_1 = y_t$ и $y_1 \neq y_2$;
      - · Для всех ребер в S, у которых $y_1 = y_t$ и $y_1 = y_2$ выполнить ShowLine(y_t, (x_1, z_1), (x_2, z_2));
      - · Удалить из S все ребра, у которых $y_1 = y_t$;
      - · Для отрезков в S найти $y_{S\,next}$ — минимальное значение y_1 у отрезков в S;
      - · Отсортировать AEL по возрастанию первого элемента и по возрастанию четвертого элемента;
      - · Найти $y_{AEL\,next}$ — минimальное значение третьего элемента в пятерках в AEL;
    - · $i = 1$;
    - **цикл пока** $i \leqslant |AEL|$ **выполнять**
      - · Выполнить ShowLine(y_t, (x_i, z_i), (x_{i+1}, z_{i+1}), C), где $(x_i, z_i, y_i, \Delta_i x, \Delta_i z)$ обозначает i-й элемент списка AEL;
      - · $i = i + 2$;
    - · $y_t = y_t + 1$;
    - **если** $y_t \geqslant y_{AEL\,next}$, **то**
      - · Удалить из AEL пятерки с третьим элементом меньшим или равным y_t;
      - · Обновить значение $y_{AEL\,next}$;
    - · В каждой пятерке $(x_j, z_j, y_j, \Delta_j x, \Delta_j z)$ в AEL заменить x_j на $x_j + \Delta_j x$, z_j на $z_j + \Delta_j z$;

конец алгоритма

5.3.3. Алгоритм с использованием S-буфера

Метод S-буфера (или алгоритм построчного сканирования с использованием Z-буфера) является оптимизацией предыдущего метода в целях уменьшения размера Z-буфера до одной строки растра. В данной оптимизации многоугольники обрабатываются не последовательно, а в совокупности. Считаем, что в списке активных многоугольников (*APL*) для каждой строки растра (сканирующей строки) занесены все многоугольники, проецируемые на эту строку. В списке активных ребер заносятся все

ребра активных многоугольников, проецируемые на строку растра. Чтобы отделять в списке активных ребер ребра одного многоугольника от ребер другого, к каждой пятерке элементов, соответствующей ребру многоугольника, добавим идентификатор многоугольника и его цвет. Таким образом в AEL вместо пятерок будем записывать семерки элементов.

Для каждой строки растра в таком случае выполняется следующая последовательность действий:

- Пересматривается список активных многоугольников.
- Пересматривается список активных ребер на предмет сокращения или пополнения.
- Инициализируется S-буфер. Соответствующая строка растра закрашивается фоном. S-буфер закрашивается максимальным значением z.
- Для каждой пары ребер активных многоугольников выполняется возможное заполнение строки растра и S-буфера (процедура ShowLine, алгоритм 11).
- Производится обработка списка активных ребер в целях подготовки к следующей строке растра.

В общем виде метод S-буфера приведен в алгоритме 14 с использованием процедуры, сформулированной в алгоритме 13.

Алгоритм 13: Процедура ShowLine для метода S-буфера

Вход: y_0 — координата y строки растра, (x_1, z_1) — координаты x и z первой точки, (x_2, z_2) — координаты x и z второй точки. C — цвет соответствующего многоугольника.
Выход: Измененные Z-буфер и область рисования.
начало алгоритма
  $x = x_1$; $z = z_1$; $\Delta z = \dfrac{z_2 - z_1}{x_2 - x_1}$;
  цикл пока $x \leqslant x_2$ **выполнять**
    если $z < Z[x]$, **то**
      Присвоить $Z[x] = z$ и закрасить в растровой области точку с координатами (x, y_0) цветом C;
    Присвоить $x = x + 1$, $z = z + \Delta z$;
конец алгоритма

Алгоритм 14: Отсечения невидимых граней с использованием S-буфера

Вход: $\mathscr{P}$ — список многоугольников трехмерной сцены
начало алгоритма
- Для каждого многоугольника в $\mathscr{P}$ вычислить $y_{P\min}$ и $y_{P\max}$ — значения минимальной и максимальной координаты y для вершин многоугольника;
- Определить значения $y_{P\,next}$, $y_{\max}$ — минимальное значение $y_{P\min}$ и максимальное значение $y_{P\max}$ для многоугольников в $\mathscr{P}$;
- $y_t = y_{P\,next}$;

цикл пока $y_t \leqslant y_{\max}$ **выполнять**
- **если** $y_t = y_{P\,next}$, **то**
  - Занести в список APL все многоугольники из $\mathscr{P}$, для которых $y_{P\min} = y_t$, удалив их из $\mathscr{P}$;
  - $y_{APL\,next} = y_t$;
  - **если** *список* $\mathscr{P}$ *пуст,* **то** присвоить $y_{P\,next} = \infty$;
  - **иначе**
    - Определить значение $y_{P\,next}$ — минимальное значение $y_{P\min}$ для многоугольников в $\mathscr{P}$
- **если** $y_t \geqslant W_{cy}$ **то**
  - Инициализировать строку растра, соответствующую координате y_t;
  - Заполнить строку растра цветом фона. Определить вещественнозначный массив Z с количеством элементов равным количеству пикселов в строке растра;
- **если** $y_t = y_{APL\,next}$, **то**
  - Добавить в AEL все семерки $\left(P, C, x_1, z_1, y_2, \dfrac{x_2 - x_1}{y_2 - y_1}, \dfrac{z_2 - z_1}{y_2 - y_1}\right)$, составленные для ребер многоугольников из APL, у которых $y_1 = y_t$ и $y_1 \neq y_2$;
  - Для всех ребер многоугольников в APL, у которых $y_1 = y_t$ и $y_1 = y_2$ выполнить ShowLine(y_t, (x_1, z_1), (x_2, z_2), C), где C — цвет соответствующего многоугольника; ;
  - Для ребер многоугольников в APL найти $y_{APL\,next}$ — минимальное значение y_1 такое, что $y_1 > y_t$. Если таких ребер нет в APL, то присвоить $y_{APL\,next} = \infty$;
  - Упорядочить AEL по значению 1-го элемента семерки, затем по возрастанию 3-го, и затем по возрастанию 6-го;
  - Найти $y_{AEL\,next}$ — минимальное значение пятого элемента в семерках в AEL;
- $i = 1$;
- **цикл пока** $i \leqslant |AEL|$ **выполнять**
  - Выполнить ShowLine(y_t, (x_i, z_i), (x_{i+1}, z_{i+1}), C_i), где $(P_i, C_i, x_i, z_i, y_i, \Delta_i x, \Delta_i z)$ обозначает i-й элемент списка AEL;
  - $i = i + 2$;
- $y_t = y_t + 1$;
- **если** $y_t \geqslant y_{AEL\,next}$, **то**
  - удалить из AEL семерки с пятым элементом меньшим или равным y_t;
  - Обновить значение $y_{AEL\,next}$;
- В каждой семерке $(P_j, C_j, x_j, z_j, y_j, \Delta_j x, \Delta_j z)$ в AEL заменить x_j на $x_j + \Delta_j x$, z_j на $z_j + \Delta_j z$;

конец алгоритма

5.3.4. Алгоритм Уоткинса

Алгоритм Уоткинса (интервальный метод построчного сканирования) работает с совокупностью проекций активных многоугольников на очередную строку растра, как и предыдущий алгоритм. Но в этом алгоритме не используется дополнительный буфер, а анализируются тот набор отрезков, который определяется списком активных ребер. Данный набор отрезков является результатом пересечения сканирующей плоскости (плоскости, параллельной плоскости xOz и проходящей через текущую строку растра) с многоугольниками трехмерной сцены (см. рис. 5.34).

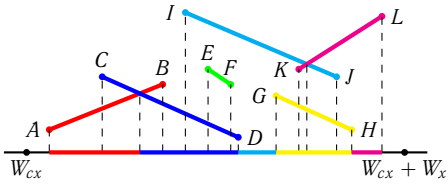


Рис. 5.34. Отрезки, проецируемые на строку растра

Основная процедура в алгоритме — процедура обработки строки растра (процедура ProcessLine, алгоритм 15), в которой выясняется — каким цветом будет закрашиваться каждый из интервалов строки растра. Интервалы определяются точками — концами отрезков, в порядке возрастания координаты x этих точек. Так, например, 12 точек (обозначенных символами латинского алфавита) на рисунке 5.34 соответствуют элементам списка AEL. Эти точки, вместе с начальной и конечной точками строки растра, образуют 13 интервалов.

Интервалы обрабатываются по порядку слева направо. Порядок следования интервалов определяется списком AEL, отсортированным в порядке возрастания текущей координаты x для каждого ребра. Так, для приведенного примера, ребра в списке AEL будут следовать в порядке: A, C, B, I, E, F, D, G, K, J, H, L. Все многоугольники, проецируемые на определенный интервал называются активными по отношению к этому интервалу. Так например для интервала $[K, J]$ активными будут три многоугольника, соответствующие отрезкам GH, IJ и KL; для интервала $[A, C]$ будет активным только один многоугольник, соответствующий отрезку AB; для интервалов [*начало*, A] и [L, *конец*] нет активных многоугольников.

Алгоритм 15: ProcessLine

начало алгоритма
- $x_{left} = -\infty$; $x_{right} = W_{cx}$ $polygonCount = 0$; $i = 1$;
- **цикл пока** *не достигнут конец списка AEL и* $x_{left} < W_{cx} + W_x$ **выполнять**
  - Пусть $(P_i, x_i, y_i, \Delta_i x)$ — очередной элемент AEL;
  - $x_{right} = x_i$;
  - **если** $x_{right} > W_{cx}$ **то**
    - **если** $x_{left} < W_{cx}$ **то** присвоить $x_{left} = W_{cx}$;
    - **если** $x_{right} > W_{cx} + W_x$ **то** присвоить $x_{right} = W_{cx} + W_x$;
    - **если** $polygonCount = 0$ **то** присвоить цвет фона переменной C;
    - **иначе если** $polygonCount = 1$ **то** присвоить $C = C_i$ — цвет многоугольника P_i;
    - **иначе**
      - Присвоить $intersectionStack = \varnothing$;
      - **повторять вычисления**
        - **если** $intersectionStack \neq \varnothing$ **то**
          - Начертить отрезок $[(x_{left}, y_0), (x_{right}, y_0)]$ цветом C;
          - Присвоить $x_{left} = x_{right}$;
          - Извлечь значение из стека $intersectionStack$ и присвоить его переменной x_{right};
        - **цикл пока** HasIntersection$(x_{left}, x_{right}, x_{int})$ **выполнять**
          - Занести x_{right} в стек $intersectionStack$;
          - Присвоить $x_{right} = x_{int}$;
        - Для всех P, таких, что $Active(P)$, найти z_{mid}

$$z_{mid} = -\frac{a(x_{left} + x_{right}) + 2by_t + 2d}{2c};$$

        - Присвоить переменной C цвет многоугольника с минимальным z_{mid};
      - **пока** $intersectionStack \neq \varnothing$;
  - Изменить $Active(P_i) = !Active(P_i)$;
  - **если** $Active(P_i)$ **то** $polygonCount = polygonCount + 1$;
  - **иначе** $polygonCount = polygonCount - 1$;
  - Начертить отрезок $[(x_{left}, y_0), (x_{right}, y_0)]$ цветом C;
  - Присвоить $x_{left} = x_{right}$, $i = i + 1$;
- **если** $x_{left} < W_{cx} + W_x$ **то**
  - **если** $x_{left} < W_{cx}$ **то** присвоить $x_{left} = W_{cx}$;
  - Присвоить $x_{right} = W_{cx} + W_x$;
  - Начертить отрезок $[(x_{left}, y_0), (x_{right}, y_0)]$ цветом фона;

конец алгоритма

Обработка интервалов проводится в следующем порядке:

- Если в интервале нет активных многоугольников, то интервал закрашивается цветом фона (интервалы $[начало, A]$ и $[L, конец]$).
- Если в интервале только один активный многоугольник, — интервал закрашивается цветом многоугольника (интервалы $[A, C]$, $[B, I]$, $[D, G]$, $[H, L]$).
- Если в интервал попадает больше одного многоугольника то:
  - Если в интервале нет пересечений отрезков (интервалы $[I, E]$, $[E, F]$,

$[F, D]$, $[G, K]$, $[J, H]$), то среди всех отрезков ищется отрезок, с минимальным значением координаты z для точки середины интервала.

– Если в интервале есть пересечения отрезков (интервалы $[C, B]$ и $[K, J]$), то: находится координата x точки пересечения; интервал делится точкой пересечения на части; для каждой из частей интервала повторяется алгоритм.

В общем виде интервальный метод построчного сканирования приведен в алгоритме 17 с использованием процедур, описанных в алгоритмах 15 и 16.

Алгоритм 16: HASINTERSECTION Проверка наличия пересечений на интервале

Вход: x_{left}, x_{right}. Параметр x_{int} может использоваться для возвращения одного из результатов.

Выход: $False$ — если на отрезке от x_{left} до x_{right} отсутствуют пересечения активных многоугольников. $True$ — в противном случае. В случае возврата $True$ параметр x_{int} содержит значение координаты x для точки пересечения.

начало алгоритма

цикл для каждого *многоугольника* P_j *такого, что* $Active(P_j)$ **выполнить**

· Вычислить

$$z_{j\,left} = -\frac{a_j x_{left} + b_j y_t + d_j}{c_j}; \quad z_{j\,right} = -\frac{a_j x_{right} + b_j y_t + d_j}{c_j};$$

цикл для каждого *многоугольника* P_k *такого, что* $Active(P_k)$ **выполнить**

· Вычислить

$$z_{k\,left} = -\frac{a_k x_{left} + b_k y_t + d_k}{c_k}; \quad z_{k\,right} = -\frac{a_k x_{right} + b_k y_t + d_k}{c_k};$$

· Вычислить $\Delta z_{left} = z_{j\,left} - z_{k\,left}$; $\Delta z_{right} = z_{j\,right} - z_{k\,right}$;

если $sign(\Delta z_{left}) \neq sign(\Delta z_{right})$ **то**

· Вычислить

$$x_{int} = \frac{x_{right}\Delta z_{left} - x_{left}\Delta z_{right}}{\Delta z_{left} - \Delta z_{right}};$$

· Выдать $True$ и **закончить алгоритм**;

· Выдать $False$;

конец алгоритма

Алгоритм 17: Алгоритм Уоткинса

Вход: $\mathscr{P}$ — список многоугольников трехмерной сцены
начало алгоритма
- Для каждого многоугольника в $\mathscr{P}$ вычислить $y_{P\min}$ и $y_{P\max}$ — значения минимальной и максимальной координаты y для вершин многоугольника;
- Определить значения $y_{P\,next}$, $y_{\max}$ — минимальное значение $y_{P\min}$ и максимальное значение $y_{P\max}$ для многоугольников в $\mathscr{P}$;
- $y_t = y_{P\,next}$;

цикл пока $y_t \leqslant y_{\max}$ **выполнять**
  если $y_t = y_{P\,next}$, **то**
    - Занести в список APL все многоугольники из $\mathscr{P}$, для которых $y_{P\min} = y_t$, удалив их из $\mathscr{P}$;
    - Для каждого нового многоугольника P_j в APL установить значение $Active(P_j) = False$ и определить значения пятерки $(C_j, a_j, b_j, c_j, d_j)$, где C_j — цвет многоугольника, а a_j, b_j, c_j, d_j — коэффициенты уравнения несущей плоскости для многоугольника P_j. Если $c_j = 0$ — удалим многоугольник из APL;
    - $y_{APL\,next} = y_t$;

    если *список* $\mathscr{P}$ *пуст,* **то** присвоить $y_{P\,next} = \infty$;
    иначе
      Определить значение $y_{P\,next}$ — минимальное значение $y_{P\min}$ для многоугольников в $\mathscr{P}$

  если $y_t = y_{APL\,next}$, **то**
    - Добавить в AEL все четверки $\left(P, x_1, y_2, \dfrac{x_2 - x_1}{y_2 - y_1}\right)$, составленные для ребер многоугольников из APL, у которых $y_1 = y_t$ и $y_1 \neq y_2$;
    - Добавить в AEL все четверки $(P, x_1, y_2, 0)$, составленные для ребер многоугольников из APL, у которых $y_1 = y_t$ и $y_1 = y_2$;
    - Для ребер многоугольников в APL найти $y_{APL\,next}$ — минимальное значение y_1 такое, что $y_1 > y_t$. Если таких ребер нет в APL, то присвоить $y_{APL\,next} = \infty$;
    - Упорядочить AEL по значению 2-го элемента четверки;
    - Найти $y_{AEL\,next}$ — минимальное значение третьего элемента в четверках в AEL;

  Выполнить процедуру ProcessLine
  - $y_t = y_t + 1$;

  если $y_t \geqslant y_{AEL\,next}$, **то**
    - удалить из AEL четверки с третьим элементом меньшим либо равным y_t;
    - Обновить значение $y_{AEL\,next}$;

  - В каждой четверке $(P_j, x_j, y_j, \Delta_j x)$ в AEL заменить x_j на $x_j + \Delta_j x$;

конец алгоритма

5.3.5. Алгоритм художника с использованием BSP-дерева

Алгоритм художника предполагает, что трехмерная сцена состоит из набора многоугольников, упорядоченных по степени приближения к наблюдателю. То есть трехмерную сцену можно представить упорядоченным списком многоугольников, где на первом месте находится самый удаленный от наблюдателя многоугольник, а на последнем — самый близкий к наблюдателю. В этом случае можно по порядку (от пер-

вого до последнего) изобразить многоугольники на экране, подобно «художнику», наносящему краски на холст. Тогда наиболее приближенный к наблюдателю многоугольник будет изображен в последнюю очередь и окажется полностью видимым. При этом он может перекрывать собой любой из многоугольников, изображенных ранее.

Такой подход является вполне удовлетворительным, если в трехмерной сцене отсутствуют многоугольники «протыкающие друг друга» или набор многоугольников, среди которого нельзя установить какой из многоугольников ближе другого (как на рисунке 5.35), а следовательно, сортировка «по приближенности к наблюдателю» невозможна. Выходом из таких ситуаций является разделение многоугольников,

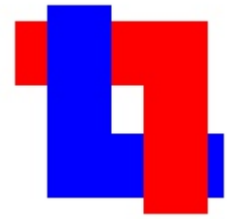


Рис. 5.35. Многоугольники,

входящих в такую группу, на более мелкие части, для которых сортировка станет возможной.

Один из методов, реализующих предварительную сортировку многоугольников для алгоритма художника и организующих разбиения многоугольников в случае необходимости — алгоритм использующий BSP-дерево.

Алгоритм построения BSP-дерева — двоичное разбиение пространства (BSP — binary space partition) — это метод, который впервые был сформулирован в 1969 году. Изначально предлагалось использовать метод для сортировки полигонов в сцене. В то время отсутствовала аппаратная метода Z-буфера, а программная реализация была слишком медленной. Однако метод двоичного разбиения пространства остался актуальным даже после создания аппаратного Z-буфера (поскольку, к сожалению, метод Z-буфера не решает многих проблем, например, отображение полупрозрачных объектов). Кроме сортировки, метод широко применяется и в других областях, к примеру, проверка на столкновение, в некоторых алгоритмах компьютерных сетей, в методе

излучательности.

Скорость работы метода двоичного разбиения пространства достигается за счёт разбиения исходного пространства и проведения предварительных вычислений. На вход метода поступает набор некоторых объектов (в случае сортировки полигонов в сцене этими объектами являются полигоны), а потом с помощью рекурсивного алгоритма создаётся двоичное дерево таким образом, что можно совершать обход дерева в порядке от более удалённых к менее удалённым или от менее удалённых к более удалённым многоугольникам.

Принцип работы алгоритма заключается в том, что все полигоны пространства (в общем случае n-мерного) разбиваются на группы, лежащие в разных выпуклых подпространствах относительно некоторой гиперплоскости (гиперплоскость — это пространство размерности $n - 1$). Не нарушая общности рассуждений будем рассматривать двумерное пространство (далее сделаем некоторые поправки для трёхмерного пространства). В этом случае гиперплоскость будет представлять собой прямую линию. Для наглядности рассмотрим пример расположения многоугольников, представленный на рисунке 5.36.

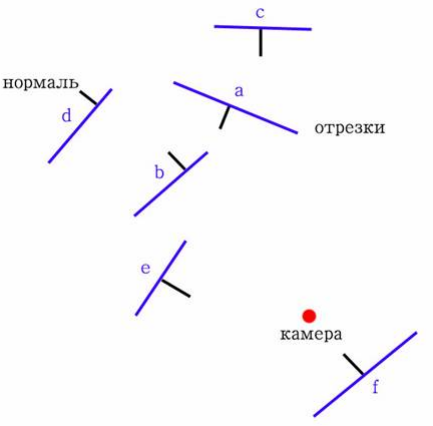


Рис. 5.36. Первоначальное расположение многоугольников

У нас есть плоскость (пространство), на которой расположены отрезки (участки гиперплоскостей) с заданными нормалями. Первым этапом алгоритма является определение разделяющей плоскость прямой. Будем выбирать произвольный отрезок

(например, отрезок **b**) и дополнять его до прямой — это и будет наша гиперплоскость (см. рис. 5.37):

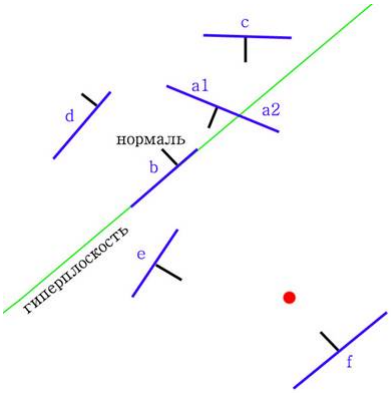


Рис. 5.37. Разделение первоначальной сцены первой гиперплоскостью

Благодаря прямой, мы получили плоскость, разбитую на две полуплоскости. Нормаль к прямой совпадает с нормалью отрезка, через который она проведена. По направлению нормали мы будем определять переднюю и заднюю полуплоскости: если нормаль находится в полуплоскости, то данная полуплоскость — передняя; иначе — задняя. Теперь нужно определить, каким полуплоскостям принадлежат отрезки. Таким образом все отрезки разбиваются на три группы: отрезки, лежащие в передней полуплоскости (**c** и **d**), отрезки, лежащие в задней полуплоскости (**e** и **f**), и отрезки, лежащие на прямой (только **b**). Если отрезок принадлежит обоим полуплоскостям, то он делится на два (так **a** делится на **a1** и **a2**). Если узлу двоичного дерева приписать прямую и все отрезки, лежащие на ней, а две оставшиеся группы приписать его дочерним поддеревьям, то получим образование структуры, представленной на рисунке 5.38.

Теперь нужно рекурсивно повторить алгоритм для каждого поддерева. То есть, мы имеем в качестве пространства переднее полуподпространство (в него входят отрезки **d**, **a1**, **c**). Выбираем любой из этих отрезков, например, **a1** и проводим через него гиперплоскость: получаем в качестве переднего поддерева отрезок **d**, а заднего — **c** (см. рис. 5.39):

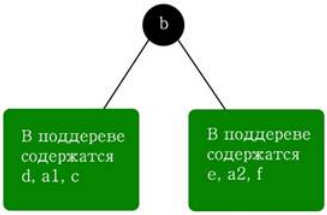


Рис. 5.38. Первый шаг построения BSP-дерева

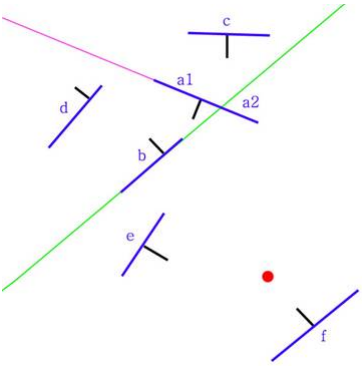


Рис. 5.39. Разделение сцены второй гиперплоскостью

Получаем структуру, представленную на рисунке 5.40.

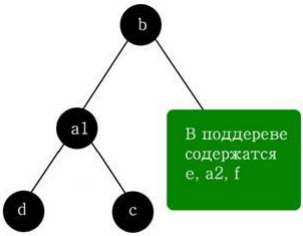


Рис. 5.40. Второй шаг построения BSP-дерева

Аналогичным образом поступаем с задним поддеревом узла **b**. Например, если выбрать за гиперплоскость сначала **a2**, а затем **e**, то получим разбиение исходной

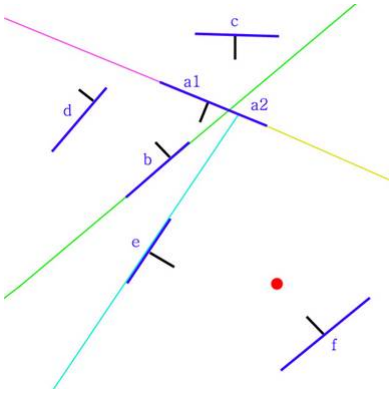


Рис. 5.41. Разбиение сцены

плоскости прямыми, как показано на рисунке 5.41 и в структуре на рисунке 5.42.

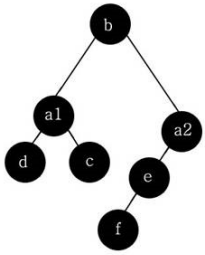


Рис. 5.42. Окончательное BSP-дерево

Итак, мы построили дерево двоичного разбиения пространства. Теперь можно, зная положение камеры, обойти все дерево по полигонам от самого дальнего до самого ближнего к камере. Начинается обход дерева, естественно, с его корня. Порядок обхода каждого узла задается положением камеры относительно прямой, соответствующей данному узлу, следующими правилами:

- Если камера находится в передней полуплоскости относительно прямой, соответствующей данному узлу, то обходим сначала заднее поддерево, потом все полигоны, которые находятся в данном узле, и в последнюю очередь переднее поддерево.
- Наоборот, если камера в задней полуплоскости, то обходим узел в порядке

от переднего поддерева к заднему.

- Если же камера находится на данной прямой, то сначала обходятся поддеревья в любом порядке, а полигоны самого узла не обходятся вовсе (т.к. они, фактически, не видны наблюдателю).