О.Г. Казанцева

ВЫЧИСЛИТЕЛЬНЫЕ ОСНОВЫ КОМПЬЮТЕРНОЙ ГРАФИКИ.

Учебное пособие

2005

УДК 004.4 (075.8)
ББК 32.973.26-018.2
К14

Автор: старший преподаватель кафедры прикладной математики и механики УО «ВГУ им. П.М. Машерова» ***О.Г. Казанцева***

Рецензенты: : ***А. И. Бочкин,*** зав. кафедрой информатики и информационных технологий УО «ВГУ им. П.М. Машерова», доцент, кандидат физико-математических наук.

Научный редактор: ***Л. В. Маркова,*** зав. кафедрой прикладной математики и механики УО «ВГУ им. П.М. Машерова», доцент, кандидат физико-математических наук.

Учебное пособие подготовлено в соответствии с учебной программой по курсу «Избранные главы информатики: Вычислительные основы машинной графики.»

Пособие предназначено тем, кто только начинает осваивать компьютерную графику. В нем содержится информация, необходимая при разработке приложений компьютерной графики: цветовые модели, форматы графических файлов. Рассмотрены базовые алгоритмы растровой графики: построение отрезка, окружности, кривой Безье. Отдельный раздел посвящен рассмотрению геометрических преобразований на плоскости и в пространстве.

Пособие предназначено для студентов специальности прикладная математика, а так же для широкого круга читателей, использующих компьютерную графику для решения прикладных и специальных задач.

УДК 004.4 (075.8)
ББК 32.973.26-018.2
К14

СОДЕРЖАНИЕ

Введение

Интерес к компьютерной графике проявляют представители различных специальностей: программисты, конструкторы, технологи, геофизики, биологи, медики, дизайнеры, художники. В настоящее время машинная графика стала необходимым инструментом в работе любого специалиста.

Многие из книг по компьютерной графике глубоко исследуют узкоспециализированные области, такие как разработка библиотек подпрограмм для реализации метода обратного хода лучей или скоростных методов изображения трехмерных сцен которые используются в компьютерных играх. При этом, например, для студентов, только начинающих вникать в эту область, часто недостает информации общеознакомительного плана, позволяющей сориентироваться в стремительно расширяющейся области компьютерной графики. Данный материал призван, хотя бы отчасти, восполнить указанный пробел.

Данное издание ориентировано на студентов, которые изучают программирование. Занятие программированием облегчает восприятие компьютерных информационных технологий, что позволяет глубже заглянуть в мир компьютеров, получить ответы на многие вопросы типа «почему именно так».

Материал пособия в значительной мере соответствует содержанию части курса «Вычислительные основы машинной графики». Здесь рассмотрены базовые методы и алгоритмы современной компьютерной графики.

§ 1. Основные понятия
1.1. История развития компьютерной графики

Отправной точкой в развитии компьютерной графики можно считать 1930 год, когда в США 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 современных телевизионных кинескопов и компьютерных мониторов.

Началом эры собственно компьютерной графики можно считать декабрь 1951 года, когда в Массачусеттском технологическом институте (МТИ) для системы противовоздушной обороны военно-морского флота США был разработан первый дисплей для компьютера “Вихрь”. Изобретателем этого дисплея был инженер из МТИ Джей Форрестер.

Одним из отцов-основателей компьютерной графики считается Айвен Сазерленд (Ivan Sotherland), который в 1962 году все в том же МТИ создал программу компьютерной графики под названием “Блокнот” (Sketchpad). Эта программа могла рисовать достаточно простые фигуры (точки, прямые, дуги окружностей), могла вращать фигуры на экране. После этой программы некоторые крупные фирмы, такие как “Дженерал моторз”, “Дженерал электрик”, приступили к разработкам в области компьютерной графики. В 1965 году фирма IBM выпустила первый коммерческий графический терминал под названием IBM-2250. В конце 70-х годов для космических кораблей “Шаттл” появились летные тренажеры, основанные на компьютерной графике. В 1982 году на экраны кинотеатров вышел фильм “Трон”, в котором впервые использовались кадры, синтезированные на компьютере. Еще в 1979 году Джордж Лукас, глава фирмы “Lucasfilm” и создатель сериала “Звездные войны”, организовал в своей фирме отдел, который занимался внедрением последних достижений компьютерной графики в кинопроизводство.

Существуют фирмы, специализирующиеся на разработке компьютеров для графических приложений, такие как “Silicon Graphics”, “Evans&Sotherland”.

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 автоматизированного проектирования (CAD – Computer Aided Design) которые позволяют разработать полный проект здания, основываясь на методах компьютерной графики.
- Химики изучают сложные молекулы белков, пользуясь средствами компьютерного отображения данных.
- В телевидении и кинематографии вместо «живых» актеров и реального реквизита используются компьютерные технологии для создания персонажей, спецэффектов, имитации стихийных бедствий и т.д.
- В математике развитие фракталов было бы невозможно без компьютеров с соответствующими средствами графического отображения данных.
- Средства мультимедиа привели к появлению новых 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.

1.2. Обработка графической информации

Самая важная функция компьютера – обработка информации. Особо можно выделить обработку графической информации.

При обработке информации, связанной с изображением на мониторе, принято выделять три основных направления:

- *распознавание образов,*
- *обработку изображений,*
- *машинную графику.*

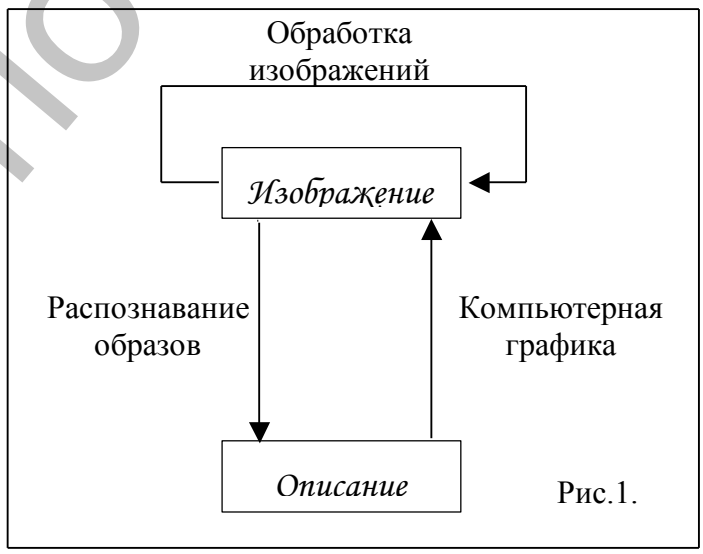


Рис.1.

Основная задача распознавания образов состоит в преобразовании уже имеющегося изображения на формально понятный язык символов. ***Распознавание образов или система технического зрения (computer vision)*** – совокупность методов, позволяющих получить описание изображения, поданного на вход, либо отнести заданное изображение к некоторому классу (так поступают, например, при сортировке почты). Одной из задач распознавание образов является так называемая скелетизация объектов, при которой восстанавливается некая основа объекта, его «скелет». Например, FineReader (система распознавания текста), установление личности по отпечаткам пальцев и др.

Обработка изображений (image processing) рассматривает задачи, в которых и входные и выходные данные являются изображениями. Например, передача изображения с устранением шумов и сжатием данных, переход от одного вида изображения к другому (от цветного к черно–белому) и т.д. Например, редактирование изображений в Adobe Photoshop.

Компьютерная (машинная) графика (computer graphics) воспроизводит изображение в случае, когда исходной является информация неизобразительной природы. Например, визуализация экспериментальных данных в виде графиков, гистограмм или диаграмм, вывод информации на экран в компьютерных играх, синтез сцен на тренажерах (имитация трехмерной реальности).

Компьютерная графика – технология создания и обработки графических изображений средствами вычислительной техники. Компьютерная графика изучает методы получения изображений полученных на основании невизуальных данных или данных, созданных непосредственно пользователем.

В том случае, если пользователь может управлять характеристиками объектов, говорят об интерактивной компьютерной графике. Этим особо подчеркивается способность компьютера создавать графику и вести диалог с человеком. В настоящее время почти любую программу можно считать системой интерактивной компьютерной графики.

1.3. Растровая и векторная графика

Растровые рисунки.

Чтобы компьютер смог обрабатывать рисунки, они должны быть представлены в числовой форме, или, как принято говорить, закодированы. ***Для кодирования рисунок разбивают на небольшие одноцветные части. Все цвета, использованные в изображении, нумеруют, и для каждой части записывают номер ее цвета.*** Запомнив последовательность расположения частей и номер цвета для каждой части, можно однозначно описать любой рисунок. Однако, количество цветов в природе бесконечно, и приходится похожие цвета нумеровать одинаковыми числами.

В зависимости от количества используемых цветов, можно закодировать более или менее реалистичное изображение. Понятно, что, чем меньше цветов в рисунке, тем меньше номеров приходится использовать, и тем проще закодировать изображение. В самом простом случае используется только черный и белый цвет.

Рисунки, закодированные описанным способом, называются ***растровыми изображениями***, ***растрами*** или ***битмапами***, от английского слова *bitmap* - карта бит. Части, на которые разбиваются изображения, называют ***пикселями*** (*PICture ELement* - элемент рисунка). Пиксели часто называют точками. Рисунок из множества пикселей можно сравнить с мозаикой. Из большого количества разноцветных камешков собирается произвольная картина (рис. 2).

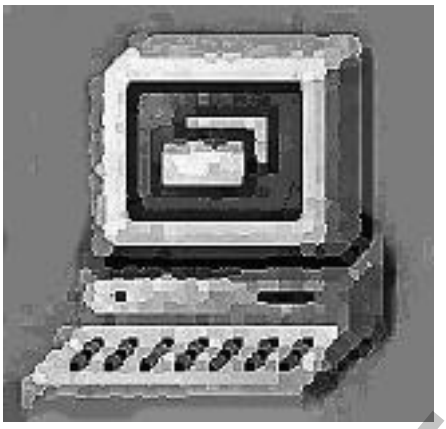


Рис. 2. Увеличенное растровое изображение.

Если для представления каждого пикселя в черно-белом рисунке достаточно одного бита, то для работы с цветом этого явно недостаточно. Однако подход при кодировании цветных изображений остается неизменным: *любой рисунок разбивается на пиксели, то есть небольшие части, каждая из которых имеет свой цвет*.

Растровые изображения достаточно широко используются в вычислительной технике. Фотографии и рисунки, введенные в компьютер, хранятся именно в виде растровых изображений. Большинство рисунков во всемирной компьютерной сети Internet представляют собой растровые файлы. Имеется множество программ, предназначенных для работы с растровыми рисунками.

Растровые изображения обладают одним очень существенным **недостатком**: их т*рудно увеличивать или уменьшать, то есть масштабировать*. При уменьшении растрового изображения несколько соседних точек преобразуются в одну, поэтому теряется разборчивость мелких деталей изображения. При увеличении - увеличивается размер каждой точки, поэтому появляется ступенчатый эффект. Кроме того, растровые изображения *занимают много места в памяти и на диске*. Чтобы избежать указанных проблем, изобрели, так называемый, векторный способ кодирования изображений.

В векторном способе кодирования геометрические фигуры, кривые и прямые линии, составляющие рисунок, хранятся в памяти компьютера в виде математических формул и геометрических абстракций, таких как круг, квадрат, эллипс и подобных фигур. Например, чтобы закодировать круг, не надо разбивать его на отдельные пиксели, а следует запомнить его радиус, координаты центра и цвет. Для прямоугольника достаточно знать размер сторон, место, где он находится и цвет закраски.

С помощью математических формул можно описать самые разные фигуры. Чтобы нарисовать более сложный рисунок, применяют несколько простых фигур. Например, взяв прямоугольник и закрасив его в черный цвет, добавив белый и серый прямоугольники и еще один черный, мы можем получить рисунок трехдюймовой дискеты:

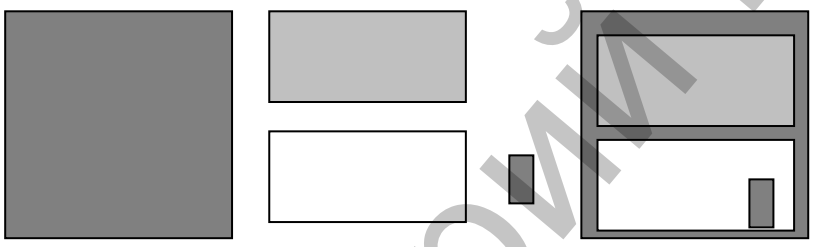


Рис. 3. Векторный рисунок из составных частей.

Любое изображение в векторном формате состоит из множества составляющих частей, которые можно редактировать независимо друг от друга. Эти части называются *объектами*. С помощью комбинации нескольких объектов, можно создавать новый объект, поэтому объекты могут иметь достаточно сложный вид. Для каждого объекта, его размеры, кривизна и местоположение хранятся в виде числовых коэффициентов. Благодаря этому появляется возможность масштабировать изображения с помощью простых математических операции, в частности, простым умножением параметров графических элементов на коэффициент масштабирования. При этом качество изображения остается без изменений. Используя векторную графику, можно не задумываться о том, готовите ли вы миниатюрную эмблему или рисуете двухметровый транспарант. Вы работаете над рисунком совершенно одинаково в обоих случаях. *В любой момент вы можете преобразовать изображение в любой размер без потерь качества.*

Важным преимуществом векторного способа кодирования изображений является то, что *размеры графических файлов векторной графики имеют значительно меньший размер, чем файлы растровой графики*. Однако есть и **недостатки работы** с векторной графикой. Прежде всего, *некоторая условность получаемых изображений.* Так как все рисунки состоят из кривых, описанных формулами, трудно получить реалистичное изо-

бражение. Для этого понадобилось бы слишком много элементов, поэтому рисунки векторной графики не могут использоваться для кодирования фотографий. Если попытаться описать фотографию, размер полученного файла окажется больше, чем соответствующего файла растровой графики.

1.4. Геометрические характеристики растра

Основные характеристики растровых изображений:

- Геометрические характеристики растра:
  - разрешающая способность (разрешение). Определяется в dpi.
  - размер. Определяется в пикселях.
  - форма пикселя.
- Глубина цвета (количество памяти в битах для воспроизведения определенной палитры цветов):
  - двухцветные изображения,
  - полутоновые изображения,
  - цветные изображения.

Объем информации, описывающий цвет пикселя, определяет *глубину цвета*. Чем больше информации определяет цвет каждой точки в рисунке, тем больше вариантов цвета существует.

Однако, не определив размер пикселя, невозможно построить изображение на основе закодированных данных. Если же мы зададим размер, то без проблем восстановим закодированный рисунок. *На практике размер пикселей не используют, а задают две другие величины:* ***размер рисунка*** и его ***разрешение.***

Размер описывает физические габариты изображения, то есть его высоту и ширину. Можно задать размеры в метрах, миллиметрах, дюймах или любых других величинах. Но в компьютере чаще всего размер задается в пикселях. При отображении на мониторе и печати на принтере каждый пиксель представляется отдельной точкой, если оборудование не делает специальных преобразований. На старых мониторах, с крупным зерном кинескопа, рисунок получится большим, а на современном принтере, в котором используются мельчайшие точки, рисунок получится очень маленьким. А каким он должен быть на самом деле? Для этого задается разрешение изображения.

Разрешение - это плотность размещения пикселей, формирующих изображение, то есть количество пикселей на заданном отрезке. Чаще всего разрешение измеряется в количестве точек на дюйм (1”=2.54см.) – **dpi** (*Dot Per Inch*). При отображении рисунков на мониторе, используют разрешение от 72 dpi до 120 dpi. При печати самым распространенным разрешением является 300 dpi, но для получения высококаче-

ственных отпечатков на современных цветных принтерах используются большие разрешения.

Форма пикселей определяется устройством вывода графической информации. Например, растр монитора на жидких кристаллах – *прямоугольный* или *квадратный*, растр электронно-лучевой трубки кинескопа – *гексагональный*, а растр в струйных и матричных принтерах – *круглый*.

В качестве растрового элемента возможно использование прямоугольника, квадрата, круга, равностороннего треугольника (рис. 4), правильного шестиугольника –гексаэдра (рис. 5), и т.д.. Можно строить растры, используя неправильные многоугольники, но практический смысл в подобных растрах отсутствует.

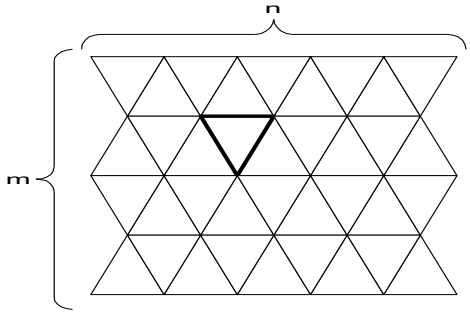


Рис. 4. Треугольный растр.

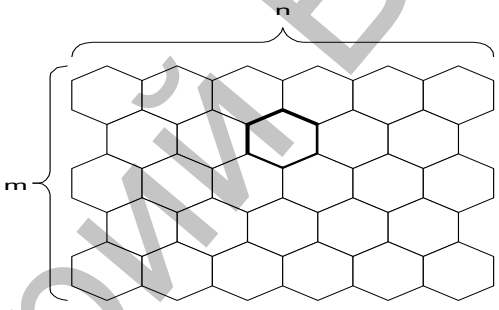


Рис. 5. Гексагональный растр.

1.5. Построение линий в прямоугольном и гексагональном растре

Поскольку экран растрового дисплея можно рассматривать как матрицу дискретных элементов (пикселей), каждый из которых может быть подсвечен, нельзя непосредственно провести отрезок из одной точки в другую.

Процесс определения пикселей, наилучшим образом аппроксимирующих заданный отрезок, называется разложением в растр. В сочетании с процессом построчной визуализации изображения он известен как ***преобразование растровой развертки***.

Для горизонтальных, вертикальных и наклоненных под углом 45° отрезков выбор растровых элементов очевиден. При любой другой ориентации выбрать нужные пиксели труднее, что показано на рис. 6.

Общие требования к изображению отрезка.

- концы отрезка должны находиться в заданных точках;
- хороший зрительный эффект;
- яркость вдоль отрезка должна быть постоянной и не зависеть от длины и наклона;
- алгоритм рисования должен работать быстро.

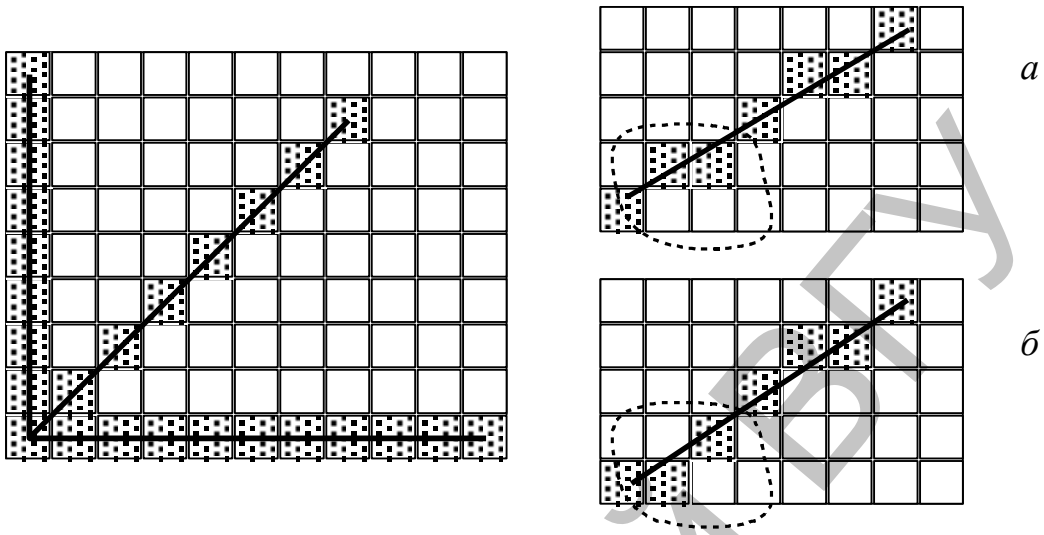


Рис. 6. Разложение в растр отрезков.

Ни одно из этих условий не может быть точно выполнено на растровом дисплее в силу того, что изображение строится из пикселов конечных размеров, а именно:

- концы отрезка в общем случае располагаются на пикселях, лишь наиболее близких к требуемым позициям и только в частных случаях координаты концов отрезка точно совпадают с координатами пикселов;
- отрезок аппроксимируется набором пикселов и лишь в частных случаях вертикальных, горизонтальных и отрезков под 45° они будут выглядеть прямыми, причем гладкими прямыми без ступенек, только для вертикальных и горизонтальных отрезков (рис. 6);
- яркость для различных отрезков и даже вдоль отрезка в общем случае различна, так как, например, расстояние между центрами пикселов для вертикального отрезка и отрезка под 45° различно (см. рис. 6).

Обеспечение одинаковой яркости вдоль отрезков разной длины и ориентации требует извлечения квадратного корня, а это замедляет вычисления. Ширина линий, проведенных под различным углом, колеблется.

Объективное улучшение аппроксимации достигается увеличением разрешения дисплея. В силу существенных технологических проблем, приемлемым разрешением является разрешение порядка 1280×1024.

Субъективное улучшение аппроксимации основано на психофизиологических особенностях зрения и, в частности, может достигаться просто уменьшением размеров экрана. Другие способы субъективного улучшения качества аппроксимации основаны на различных программных ухищрениях по "размыванию" резких границ изображения.

Рассмотрим способы построения линий в прямоугольном и гексагональном растрах.

В прямоугольном растре построение линии осуществляется двумя способами:

1) Соседние пиксели линии могут находиться в одном из восьми возможных (см. рис. 7.а) положениях. Результат – *восьмисвязная линия*. Недостаток – слишком тонкая линия при угле 45°.

2) Соседние пиксели линии могут находиться в одном из четырех возможных (см. рис. 7.б) положениях. Результат – *четырехсвязная линия*. Недостаток – избыточно толстая линия при угле 45°.

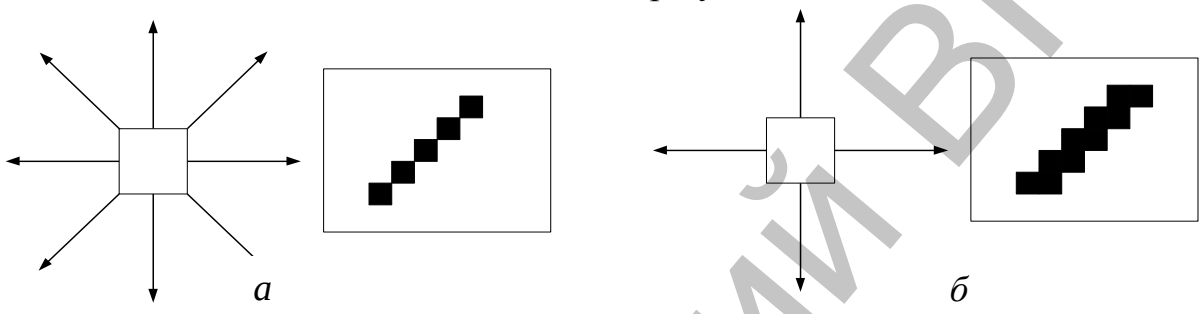


Рис. 7. Построение линии в прямоугольном растре.

В гексагональном растре линии *шестисвязные* (рис. 8) такие линии более стабильны по ширине, т.е. дисперсия ширины линии меньше, чем в квадратном растре.

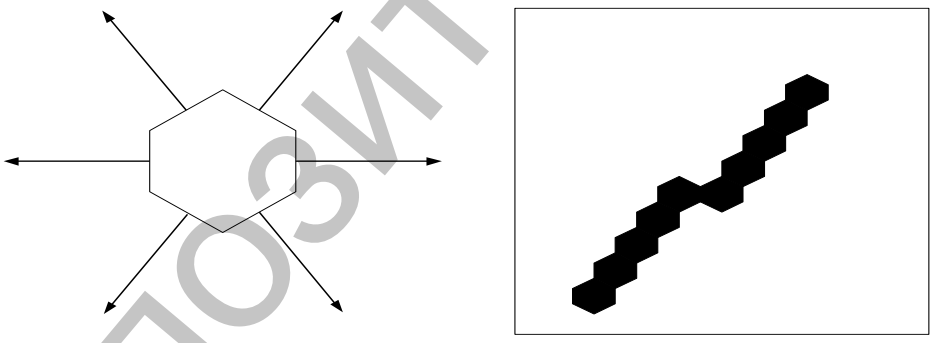
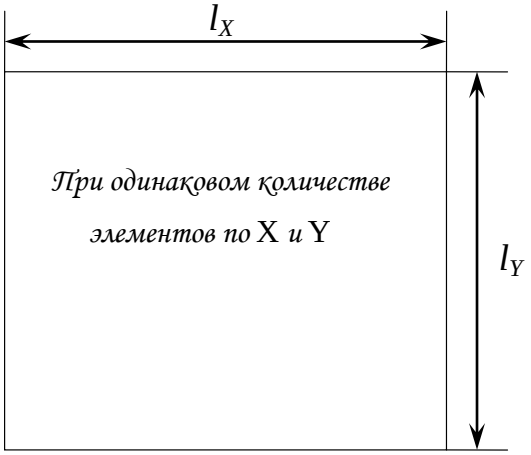


Рис. 8. Построение линии в гексагональном растре.

Каким образом можно оценить, какой растр лучше? Одним из способов оценки является передача по каналу связи кодированного, с учетом используемого растра, изображения с последующим восстановлением и визуальным анализом достигнутого качества. **Экспериментально и математически доказано, что гексагональный растр лучше,** т.к. обеспечивает наименьшее отклонение от оригинала. Но разница не велика.

Моделирование гексагонального растра возможно на основе прямоугольного. Для этого гексаугольник представляют в виде прямоугольника. Определим, какие пропорции при этом должно иметь гексагональное изображение (см. рис.9)?



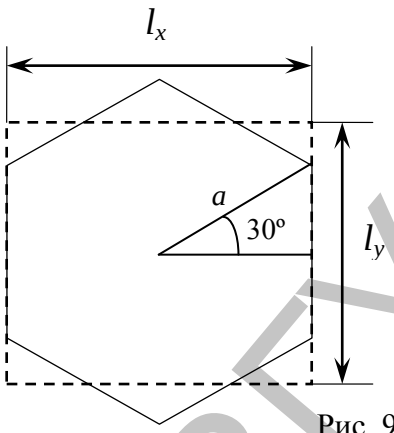


Рис. 9.

$$\frac{l_x}{l_y} = \frac{\cos(30^\circ)\cdot 2a}{3a/2} = \frac{2}{\sqrt{3}};$$

Техническая реализация гексагонального растра не сложна. *Модель гексагонального растра получают из прямоугольного* (рис. 10)*, задержав на 1 пиксель каждую нечетную строчку изображения, и растянув изображение на экране таким образом, чтобы* $\frac{l_X}{l_Y} = \frac{2}{\sqrt{3}}$.

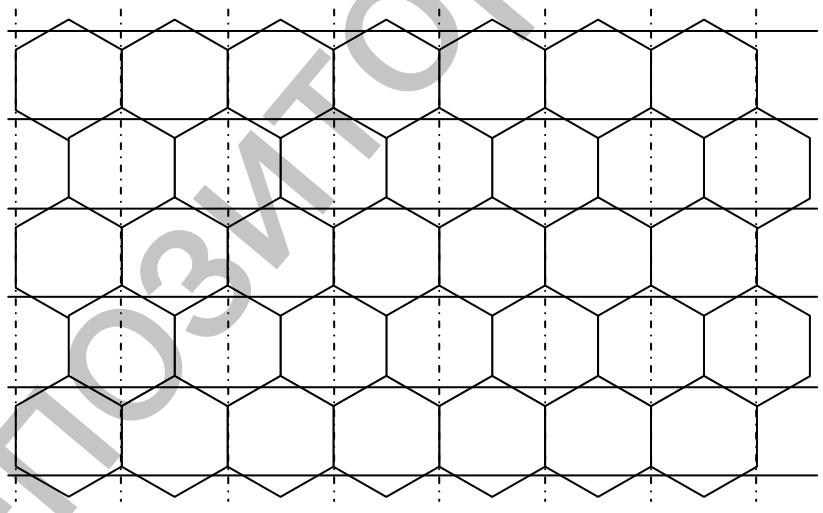


Рис. 10. Построение гексагонального растра.

Тот факт, что гексагональный растр не используется, объясняется следующими причинами:

1. некоторое усложнение алгоритмов;

2. преимущество гексагонального растра не очень велико;

3. историческая ориентация на прямоугольный растр.

1.6. Цветовые модели

Как уже отмечалось, каждый пиксель растрового изображения содержит информацию о цвете. Любой векторный объект также содержит информацию о цвете его контура и закрашенной области. Информация может занимать *от одного до тридцати двух бит*, в зависимости от глубины цвета.

Если мы работаем с черно-белыми изображениями, то цвет кодируется нулем или единицей. Никаких проблем в этом случае не возникает. Для несложных рисунков, содержащих 256 цветов или столько же градаций серого цвета, нетрудно пронумеровать все используемые цвета. Но, для изображений в истинном цвете, содержащих миллионы разных оттенков, простая нумерация не подходит. Для них разработаны несколько моделей представления цвета, помогающих однозначно определить любой оттенок.

Цветовая модель определяет способ создания цветов, используемых в изображении. Всего разработано *три основных цветовых модели* и *множество их модификаций.*

Коротко рассмотрим основные модели представления цвета. Из школьного курса физики известно, что солнечный свет можно разложить на отдельные цветные составляющие. В то же время, собрав вместе в нужных пропорциях разноцветные лучи, мы получим луч белого цвета. Изменим немного пропорции - и у нас готов источник света заданного цвета. В телевизорах и компьютерных мониторах используется *люминофор, который светится красным, зеленым и синим цветом.* Смешивая эти три цвета можно получить разнообразные цвета и их оттенки. На этом и основана модель представления цвета ***RGB,*** названная так по начальным буквам входящих в нее цветов: *Red - красный, Green - зеленый, Blue - синий.* Любой цвет в этой модели представляется тремя числами, описывающими величину каждой цветовой составляющей. ***Черный цвет образуется, когда интенсивность всех трех составляющих равна нулю***, а ***белый - когда их интенсивность максимальна.*** Множество компьютерного оборудования работает с использованием модели RGB, кроме того, эта модель очень проста. Этим объясняется ее широкое распространение. К сожалению, в модели RGB теоретически невозможно получить некоторые цвета, например насыщенный сине-зеленый, поэтому работать с моделью цвета RGB не всегда удобно. Кроме того, модель RGB сильно связана с реализацией ее на кон-

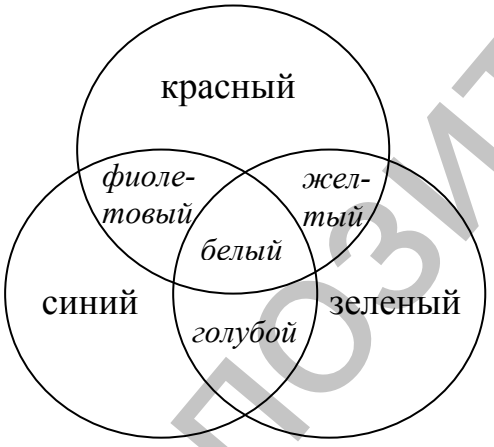


Рис.11. Модель RGB.

кретных устройствах. В настоящее время достаточно распространенным является формат True Color, в котором каждая компонента представлена в виде байта, что дает 256 градаций для каждой компоненты: R=0..255, G=0..255, B=0..255. Количество цветов составляет 256^3= 16 777 216.

Цветовое пространство, образуемое интенсивностями красного, зеленого и синего, представляют в виде цветового куба (см. рис.12). Вершины куба *A, B, C* являются максимальными интенсивностями зеленого, синего и красного соответственно, а треугольник, которые они образуют, называется ***треугольником Паскаля.*** Периметр этого треугольника соответствует максимально насыщенным цветам. *Цвет максимальной насыщенности содержит всегда только две компоненты.* На отрезке *OD* находятся оттенки серого, причем точка *O* соответствует черному, а точка *D* белому цвету.

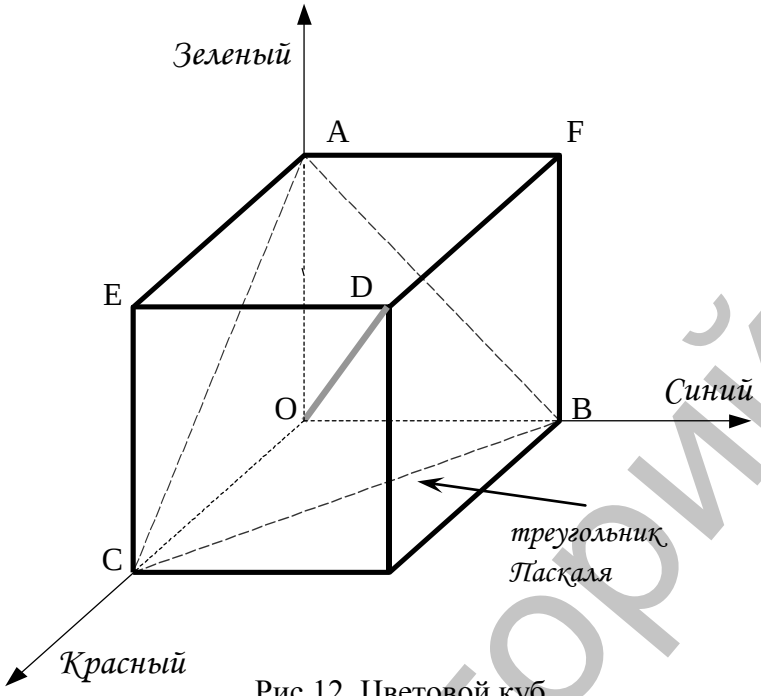


Рис.12. Цветовой куб.

Большинство цветов, которые мы видим в окружающем нас мире, являются следствием отражения и поглощения света. Например, солнечный свет, падая на зеленую траву, частично поглощается, и отражается только его зеленая составляющая.

Рис.13. Модель CMYK.

При печати на принтере, на бумагу наносится цветная краска, которая отражает только свет определенного цвета. Все остальные цвета поглощаются (вычитаются) из солнечного света. На эффекте вычитания цветов построена другая модель представления цвета, называемая ***CMYK.*** Эти буквы также взяты из названий цветов: *Cyan - голубой, Magenta - пурпурный, Yellow - желтый, blacK - черный.* Строго говоря, Magenta не является пурпурным цветом. Точное название этого цвета – фуксин, но в компьютерной литературе и в про-

граммах принято называть этот цвет пурпурным. В разновидности этой модели, называемой ***CMY***, отсутствует черный цвет, но она применяется значительно реже.

Выбор цветов для модели неслучаен, они тесно связаны с цветами модели RGB. *Голубой цвет образуется при поглощении красного, пурпурный при поглощении зеленого, а желтый отраженный цвет получается в результате поглощения синего.* При нанесении большего количества красок разных цветов поглощается больше цвета и меньше отражается. Таким образом, ***при смешении максимальных значений этих трех цветов мы должны получить черный цвет, а при полном отсутствии краски должен получиться белый цвет.*** Однако в действительности при смешении трех красок получается грязно-бурый цвет, так как используемые реальные красители отражают и поглощают цвет не так, как описано в теории. Черный цвет получается только при добавлении черной краски, поэтому в модель CMYK и добавлена черная составляющая.

Система CMYK широко применяется в полиграфии. Типографское оборудование работает исключительно с этой моделью, да и современные принтеры тоже используют красители четырех цветов. При печати на бумагу наносятся несколько слоев прозрачной краски, и в результате мы получаем цветное изображение, содержащее миллионы различных оттенков.

Системы RGB и CMYK удобны при работе с конкретным оборудованием, но не очень удобны для человеческого восприятия. Представив себе желаемый цвет, вы не сможете сказать, сколько в нем составляющих цветов той или иной модели.

Следующая модель цвета основана на восприятии цвета человеком. Все цвета в ней описываются тремя числами. Одно задает собственно цвет, другое – насыщенность цвета, а третье – яркость. *Цвет в этой модели независим от используемых технических средств.* Есть несколько вариантов модели, называемых разными терминами, но означающих одно и то же. Чаще других встречается модель ***HSB,*** в которой каждый цвет описывается *цветовым тоном – Hue* (значение измеряется в градусах от 0 до 360, поскольку здесь цвета радуги располагаются по кругу), *насыщенностью – Saturation* и *яркостью – Brightness* (их значения находятся в диапазоне [0..1]). Например, при S=0, т.е. на оси B – серые тона. Значение B=0 соответствует черному цвету. Белый цвет кодируется как S=0, B=1. Цвета, расположенные по кругу напротив друг друга, т.е. отличающиеся по H на 180°, являются дополнительными.

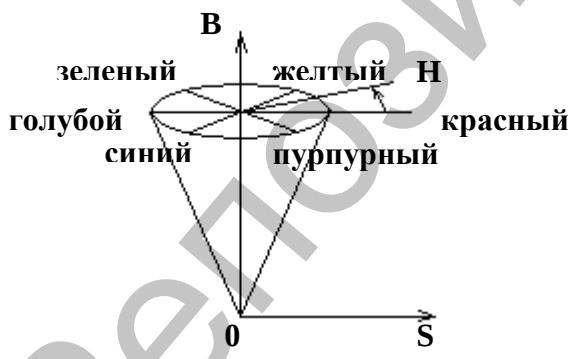


Рис.14. Модель HSB.

Модель HSB не зависит от оборудования и удобна для восприятия человеком, поэтому с ней часто работают различные программы, в дальнейшем преобразуя цвета в модель RGB для показа на экране монитора или в модель CMYK - для печати на принтере. Кроме того, модель HSB удобно использовать при редактировании рисунков. Например, вы хотите заменить зеленый лист на желтый. В редактируемой фотографии достаточно поменять только цветовую составляющую используемых цветов, не меняя яркость и насыщенность. Рисунок при этом не изменится, но примет иной оттенок.

Есть и иные модели представления цвета, но в подавляющем большинстве случаев используется перечисленные выше.

Рассмотренные выше цветовые модели являются ***компонентным*** способом кодирования цвета.

Часто для описания оттенков, используются фиксированные палитры, то есть список заданных цветов. В результате исследований определяют наиболее часто используемые цвета и помещают их в палитру. Каждый цвет изображения, в этом случае, кодируется индексом. Палитру можно воспринимать как таблицу цветов. Она устанавливает взаимосвязь между индексом цвета и его компонентами в выбранной цветовой модели. Поэтому такой способ кодирования цвета называется ***индексным*.**

1.7. Форматы файлов для хранения растровых изображений

Для хранения растра изображений разработано много форматов файлов. Рассмотрим основные из них:

1) *PSD* (PhotoShop Document). Внутренний формат представления данных в PhotoShop. Поддерживает стандарт сжатия RLE (Run Length Encoding): сжатие без потери качества; группа пикселей одного и того же цвета заменяется комбинацией цвет-счетчик. Поддерживает практически все цветовые модели. Имеет возможность использования α-канала (можно назначить несколько цветов прозрачными).
2) *BMP* (Windows BITMAP). Собственный формат графического представления Windows. Поддерживает цветовую схему RGB. Поддерживает индексирование (256 цветов).
3) *DIB* (Device Independent Bitmap). Формат, аналогичный предыдущему, но применяется простейшие алгоритмы сжатия RLE.
4) *GIF* (CompuServe Graphics Interchange Format). Разработан в 1987 г. для обмена графическими данными по сети. В 1989 г. вышла модификация GIF89a. Добавлены возможности сохранения нескольких картинок в один файл и проигрывание их с определенным интервалом. Поддержка

α-канала. Добавлена чересстрочная развертка. Используется сжатие LZW (Lempel, Ziv, Welch) – алгоритм без потери качества; основан на частотной характеристике цвета; работа основана на вычислении частоты, с которой встречается отдельный цвет в изображении, и заменой наиболее часто встречающихся цветов наименьшими индексами.

5) *JPEG* (JPG) (Joint Photographic Extents Group – объединенная группа экспертов в области фотографии). Формат хранения цифровых изображений с потерей качества. При его разработке за основу была взята идея создания стандарта, который существенно уменьшит размер существующих изображений, при этом возможно ухудшение качества иллюстрации. Задача была решена следующим образом: при сжатии графического файла рассматриваются соседние пиксели, а вернее, разница между ними. Создается единое световое пространство, после чего отбрасывается некоторый объем информации о цвете, который обуславливается степенью сжатия. Происходит это неслучайно, человеческий глаз более чувствителен к разнице света (яркости), нежели к цвету. Далее происходит обработка блоков по 8x8 пикселей и формируются цифровые представления. На последнем этапе используется кодирование методом Хафмана для более эффективного сжатия данных. Таким образом, при сжатии JPEG есть риск потерять мелкие детали на изображениях. Аппаратно независим.
6) *PNG* (Portable Network Graphics – переносимая сетевая графика). Разработан как замена формату GIF. Сжатие по усовершенствованному алгоритму LZW. Поддержка чересстрочной развертки (2-ой: по горизонтали и по вертикали). Поддержка α-канала (прозрачность можно задать от 1-99%). Встроенная γ-коррекция (независимость от оборудования). Предназначен для хранения цветового изображения с разрядностью до 48 бит. В этот стандарт заложена возможность отображения картинки на различных операционных системах и платформах с одинаковой яркостью. Специальный байт отвечает за зависимость яркости свечения экрана вашего монитора от напряжения на электродах кинескопа. Для хранения и печати формат PNG не годится, так же как и GIF.
7) *TIFF* (Tagged Image File Format). Индексированный формат графических файлов. Аппаратно-независим. Поддерживают все цветовые модели. Может использоваться сжатие по алгоритму LZW. α-канал.
8) *EPS* (Encapsulated Post Script). Название языка описания файлов, направляемых на печать, и собственно формата, изображение в котором может быть распечатано на PostScript-устройстве. EPS часто используется для передачи векторной и растровой графики в издательских системах. При передачи EPS для печати в них сохраняется цветовая модель. Занимают меньше места, чем JPEG-данные с такой же степенью компрессии. Но некоторые устройства их не воспринимают.

9) *WMF* (Windows Media File). Это внутренний формат операционной системы Windows. Он служит для передачи векторов через буфер обмена (Clipboard). Именно в нем вы передаете графическое изображение между пакетами, копируя изображение в буфер. Цветовая гамма рисунка в формате WMF искажается очень сильно

Для работы с фотографиями стоит использовать форматы *TIFF, JPEG* и *EPS*. Для передачи или хранения файлов наиболее разумно сжимать методами *JPEG*.

Область применения алгоритмов сжатия:

- Фотореалистические изображения.
- Видео изображения.
- Текстуры в играх.
- Изображение с малым количеством цветов и с большими одноцветными областями (деловая графика).

Например, необходимо 1,44 Мб памяти для фотографии 800x600 в формате RGB. Для сохранения 1 часового фильма 30 кадров в секунду – 144 Гб памяти. Это огромные затраты памяти, поэтому необходимо использовать алгоритмы сжатия.

Требования к алгоритму сжатия:

1) Высокая степень компрессии.
2) Высокое качество изображения.
3) Высокая скорость компрессии / декомпрессии.
4) Возможность постепенного вывода изображения (фрагментами в сети).
5) Редактируемость.
6) Невысокая стоимость аппаратной и программной реализации.

Рассмотрим один из самых популярных форматов – формат BMP.

Общая структура BMP файла:

BITMAPFILEHEADER	14 байт,
BITMAPINFOHEADER	40 байт,
Палитра	размер зависит от количества цветов,
Битовый массив растрового изображения	число байт определяется размерами растра и количеством бит на пиксел.

1. BITMAPFILEHEADER – заголовок файла, в нем помещается общее описание файла. *Заголовок имеет следующие поля:*

WORD **bfType** – хранит символы «BM». Это код формата.
DWORD **bfSize** – общий размер файла в байтах.
WORD **bf Reserved1** – зарезервировано, пока что равно 0.
WORD **bf Reserved2** – зарезервировано, пока что равно 0.
DWORD **bfOffBits** – адрес битового массива в данном файле.

2. BITMAPINFOHEADER – еще один заголовок, в котором хранится описание размеров растра и цветового формата пикселов.

```
DWORD   biSize – размер заголовка, равен 40.
LONG    biWidth – ширина растра в пикселах.
LONG    biHeight – высота растра в пикселах.
WORD    biPlanes – должно быть равно 1.
WORD    biBitCount – бит/пиксел,  1, 4, 8, 16, 24 или 32.
DWORD   biCompression – равно нулю.
DWORD   biSizeImage – размер в байтах битового массива растра.
LONG    biXPelsPerMeter – разрешение по X в пикселах на метр.
LONG    biYPelsPerMeter – разрешение по Y в пикселах на метр.
DWORD   biClrUsed – если =0, то используется макс.число цветов.
DWORD   biClrImportant – равно 0, если biClrUsed=0.
```

3. Далее в файле помешается палитра в виде записей RGBQUAD. Каждая запись содержит четыре поля:

```
BYTE    rgbBlue – цветовая компонента B, от 0 до 255.
BYTE    rgbGreen – цветовая компонента G, от 0 до 255.
BYTE    rgbRed – цветовая компонента R, от 0 до 255.
BYTE    rgbReserved – не используется, равно 0.
```

Количество записей RGBQUAD равно количеству используемых цветов. Палитра отсутствует, если число бит на пиксел равно 24. Также палитра не нужна и для некоторых цветовых форматов 16 и 32 бит на пискел.

Обозначения для типов полей:

```
BYTE  – однобайтовое целое число без знака.
WORD  – двухбайтовое целое число без знака.
DWORD – четырехбайтовое целое число без знака.
LONG  – четырехбайтовое целое число со знаком.
```

После палитры (если она есть) в файле BMP записывается растр в виде битового (а точнее, байтового массива). В битовом массиве последовательно записываются байты строк растра. Количество байт в строке должно быть кратно 4, поэтому, если количество пикселов по горизонтали не соответствует такому условию, то справа в каждую строку дописывается некоторое число битов (выравнивание строк на границу двойного слова).

Пример: Пусть имеется изображение размером 2x3 пикселей. 1 строка: черный (RGB=00 00 00), серый (RGB=80 80 80), белый (RGB=FF FF FF). 2-строка: красный (RGB=FF 00 00), зеленый (RGB=00 FF 00), синий (RGB=00 00 FF).

Как видно из рис.16, размер файла 0000004E=78 байт. Адрес битового массива – 00000036=54 байт, ширина и высота растра в пикселах 3x2, на каждый пиксел приходится 0018=24 бита (3 байта: BGR), палитра не ис-

пользуется. Размер в байтах битового массива растра `0018=24` байта. Растр записывается по строкам, начиная с последней. Причем, поскольку количество байт в строке должно быть кратно 4, то справа в каждую строку дописывается по 3 нулевых байта (`000000`).

```
00000000: 42 4D 4E 00 00 00 00 00|00 00 36 00 00 00 28 00 | BMNЂЂЂЂЂ6ЂЂЂ(Ъ
00000010: 00 00 03 00 00 00 02 00|00 00 01 00 18 00 00 00 | ЂЂЂЂЂЂЂЂЂЂЂЂЂЂЂЂ
00000020: 00 00 18 00 00 00 00 00|00 00 00 00 00 00 00 00 | ЂЂЂЂЂЂЂЂЂЂЂЂЂЂЂЂ
00000030: 00 00 00 00 00 00 00 00|FF 00 FF 00 FF 00 00 00 | ЂЂЂЂЂЂЂЂяЂяЂяЂЂЂ
00000040: 00 00 00 00 00 80 80 80|FF FF FF 00 00 00      | ЂЂЂЂЂЂЂЂяяяЂЂЂ
```

Рис.16. Дамп bmp файла с изображением размером 2x3 пиксела.

1.8. Методы улучшения растровых изображений

В растровых системах при невысокой разрешающей способности существует проблема ступенчатого эффекта. Основная причина появления лестничного эффекта заключается в том, что изображаемые объекта имеют непрерывную природу, тогда как растровое устройство дискретно Этот эффект особенно заметен для наклонных линий – при большом шаге сетки растра пикселы образуют как бы ступени лестницы.

Устранение контурных неровностей. Для того, чтобы растровое изображение линии выглядело более гладким, можно цвет угловых пикселов «ступенек лестницы» заменить на некоторый оттенок, промежуточный между цветом объекта и цветом фона. Можно вычислять цвет пропорционально части площади ячейки растра, покрываемой идеальным контуром объекта: $C_x = \dfrac{CS_x + C_ф(S - S_x)}{S}$, где C_x – искомый цвет ячейки, C – исходный цвет объекта, S_x – часть площади ячейки, покрываемой контуром, S – площадь всей ячейки, $C_ф$ – цвет фона.

Локальная фильтрация. Она осуществляется путем взвешенного суммирования яркостей пикселов, расположенных в некоторой окрестности текущего обрабатываемого пиксела (окне).

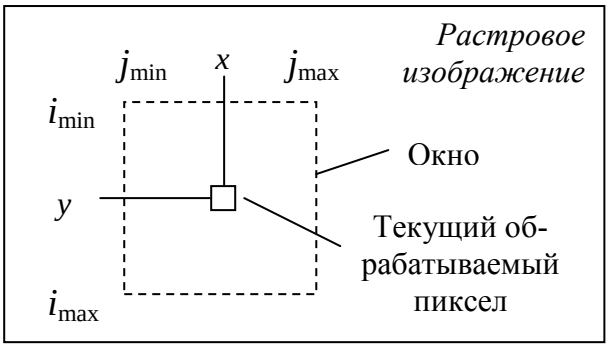


Рис.17. Окресность обрабатываемого пиксела.

$$C_{x,y} = \frac{1}{K} \sum_{i=i_{\min}}^{i_{\max}} \sum_{j=j_{\min}}^{j_{\max}} P_{x+j,y+i} \cdot M_{i-i_{\min},j-j_{\max}},$$

$C_{x,y}$ – новое значение цвета пиксела, P – значение цвета текущего пиксела, K – нормирующий коэффициент, M – двумерный массив (маска) коэффициентов, который определяет свойства фильтра.

На практике часто используется фильтр размером 3x3, $j_{min}=i_{min}= -1$, $j_{max}=i_{max}= +1$. Значение нормирующего коэффициента обычно выбирается равным сумме элементов маски. Этим обеспечивается сохранение масштаба яркости преобразованного растра.

С помощью локальной цифровой фильтрации можно выполнить достаточно разнообразную обработку изображений – повышение резкости, выделение контуров и др.

Метод полутонов. Если достаточно близко расположить маленькие точки различных цветов, то они будут восприниматься как одна точка с некоторым усредненным цветом. В большинстве случаев оттенки цветов имитируются комбинированием, смесью точек. Так если в ячейке размерами $m \times n$ использовать два цвета, то с помощью этой ячейки можно получить $mn+1$ различных цветовых градаций.

Рассмотрим способ преобразования растрового изображения размером $a \times b$ с определенной глубиной цвета в другой растр, предназначенный для отображения с помощью графического устройства, в котором используется ограниченное количество основных цветов.

Данный способ заключается в том, что каждый пиксел растра исходного изображения превращается в ячейку из $m \times n$ пикселов растра отображения. В этом случае размер растра становится равным $ma \times nb$ пикселов. Часто используются квадратные ячейки. Пример ячеек размером 2×2 представлен на рис. 18.

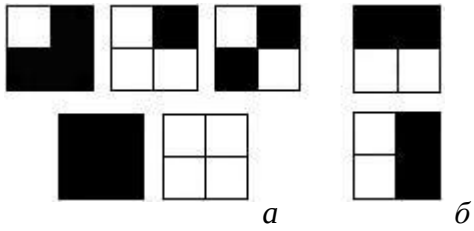


Рис. 18. Каждый пиксел изображения заменяется ячейкой 2×2 пиксела.

Предположим, что графическое устройство способно воспроизвести только два цвета: белый и черный. При такой организации получается пять возможных тонов серого (рис.18.*а*). При выборе конфигураций следует проявлять осторожность, так как иначе могут возникнуть нежелательные мелкомасштабные структуры. Например, не следует применять ни одну из

конфигураций, изображенных на рис.18 *б*, иначе это приведет к тому, что для большой области с постоянной интенсивностью на изображении появятся нежелательные горизонтальные или вертикальные линии.

Алгоритм переноса. Метод дает полутоновые изображения с неплохим качеством. В данном методе разрешение изображения не изменяется. Идея метода: предварительно задается число градаций серого, вычисляется середина этого диапазона и, проходя изображение в направлении слева – направо и сверху – вниз, рассматриваются точки изображения. Если яркость точки ближе к белому, то ставится белая точка, а если к черному, то ставится черная точка, половина полученной ошибки прибавляется к яркости точки справа, а вторая половина к яркости точки снизу (пропорции можно изменять).

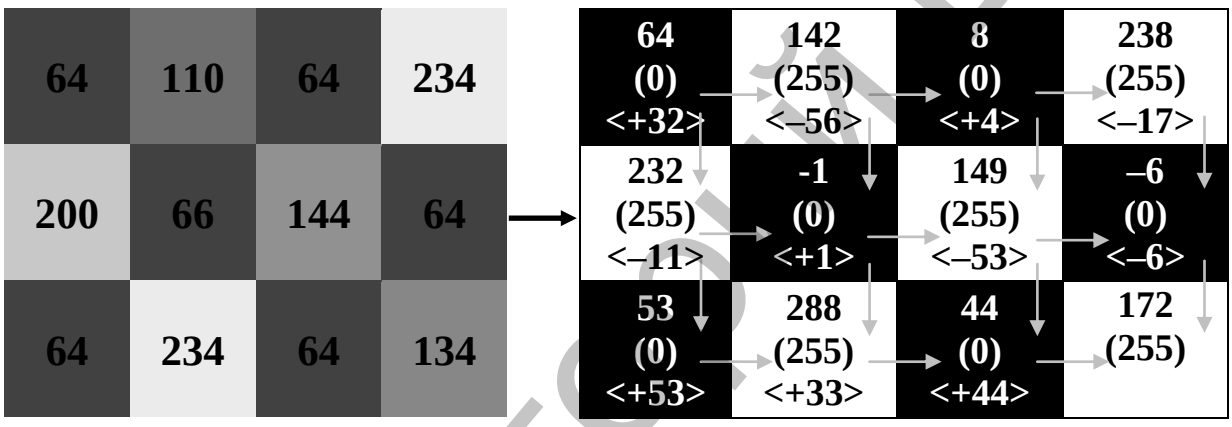


Рис.19. Пример реализации алгоритма переноса.

С помощью подобного метода можно рисовать цветную картинку, а также использовать и в случае большего числа градаций. Если исходных градаций более двух, то текущую яркость сравнивают с ближайшей, после чего выбирают соответствующую яркость, вычисляют ошибку и переносят на следующую точку. Алгоритм используется отдельно для каждого цвета.

§ 2. Базовые растровые алгоритмы
2.1. Параметрический алгоритм рисования линии

Параметрический алгоритм является самым простым способом рисования линии. Однако, у этого метода есть существенные *недостатки*, а именно: необходимость выполнения *операций над вещественными числами* и использования *операции деления*, что значительно усложняет аппаратную организацию и увеличивает время работы алгоритма.

Предположим, что необходимо провести линию из точки (x_1, y_1) в точку (x_2, y_2) с учетом того, что заданы яркости в начальной и конечной точках линии, C_1 и C_2 соответственно, и по яркости произвести линейную интерполяцию (см. рис. 20).

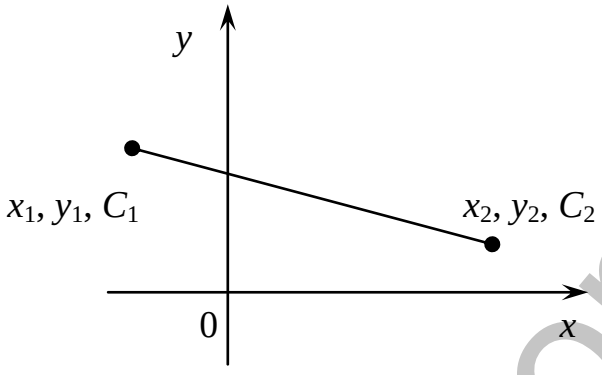


Рис. 20.

Любую точку на этой линии можно представить в виде:

$$x = \lfloor x_1 + t \cdot (x_2 - x_1) \rceil$$
$$y = \lfloor y_1 + t \cdot (y_2 - y_1) \rceil \text{ где}$$
$$C = \lfloor C_1 + t \cdot (C_2 - C_1) \rceil$$

$t \in [0; 1]$, $\lfloor \cdot \rceil$ – знак округления до целого.

Оптимальный шаг приращения по t выбирается как

$$t = \frac{1}{\max\{|x_2 - x_1|, |y_2 - y_1|\}} = \frac{1}{N-1},$$

где N – длина линии в пикселях.

Также можно проводить вычисления через приращение координат x, y, C (с начальным присвоением $x_0=x_1$, $y_0=y_1$, $C_0=C_1$):

$$x_i = x_{i-1} + \Delta x,\ y_i = y_{i-1} + \Delta y,\ C_i = C_{i-1} + \Delta C, \text{ где}$$

$$\Delta x = \frac{x_2 - x_1}{N-1}; \Delta y = \frac{y_2 - y_1}{N-1}; \Delta C = \frac{C_2 - C_1}{N-1}.$$

Значения приращений вычисляются в начале функции и не входят в цикл построения линии на экране, за счет чего повышается быстродействие.

К *достоинствам* алгоритма следует отнести: простоту его программной реализации. Альтернативой приведенному алгоритму является алгоритм Брезенхема.

Следует отметить, что в случае задания цвета точек концов отрезка, линейная интерполяция каждой цветовой компоненты проводится отдельно.

2.2. Алгоритм Брезенхема рисования линии

В 1965 году Брезенхемом был предложен простой целочисленный алгоритм для растрового построения отрезка. Алгоритм выбирает оптимальные растровые координаты для представления отрезка. В процессе работы одна из координат – либо *x*, либо *y* (в зависимости от углового коэффициента) – изменяется на единицу. Изменение другой координаты (либо на нуль, либо на единицу) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние будем называть ошибкой. *Алгоритм построен так, что требуется проверять лишь знак этой ошибки.*

Рассмотрим алгоритм для частного случая – рисования линии в первом октанте, т.е. для отрезка с угловым коэффициентом от нуля до единицы.

Будем рисовать линию из точки (x_1, y_1) в точку (x_2, y_2) под углом к оси *Ox* менее 45°, т.е. $(x_2 - x_1) \geq (y_2 - y_1) \geq 0$. Считаем, что координаты концов отрезка целые числа.

Введем обозначения: $\Delta x = (x_2 - x_1)$, $\Delta y = (y_2 - y_1)$.

Поскольку $\Delta x \geq \Delta y$, то на *i*-ой итерации значение x_i рассчитывается так, $x_0 = x_1$; $x_i = x_{i-1} + \Delta x_i$; $\Delta x_i = 1$. По ходу движения по оси *Ox* алгоритм смотрит, насколько пиксел отклонился от реальной линии по оси *Oy*. Если ошибка настолько велика, что соседний по оси *Oy* пиксел оказывается ближе к реальной линии, чем тот, что мы собираемся поставить, то алгоритм переходит на соседний по этой оси пиксел. Таким образом, $y_0 = y_1$,

$$y_i = y_{i-1} + \Delta y_i, \text{ где } \Delta y_i = \begin{cases} 0, \text{ если } h_i < 0.5 \\ 1, \text{ если } h_i \geq 0.5 \end{cases}.$$

В алгоритме используется управляющая переменная h_i, которая называется ошибкой отклонения на каждом *i*-ом шаге и рассчитывается как $h_i = \frac{\Delta y}{\Delta x} \cdot x_i - y_{i-1}$. Заметим, что h_i также имеет смысл углового коэффициента и поэтому, можно считать, что $h_0 = \frac{\Delta y}{\Delta x}$.

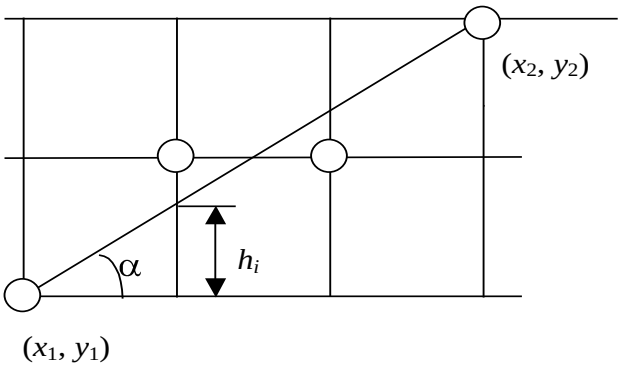


Рис. 21. Текущая ошибка в алгоритме Брезенхема

Согласно рис.21, линия начинается в точке (x_1, y_1), и в этой точке ошибка отклонения равна 0. Следующий пиксел по оси *Ox* будет иметь координату x_1+1. Приращение вертикальной координаты следующего пиксела будет либо 0, либо 1. Текущая ошибка по рисунку больше, чем полпиксела, поэтому вертикальная координата переходит в y_1+1. Следующая точка сдвигается по оси *Ox* на один пиксел, а по оси *Oy* остается в y_1+1. Наконец, последняя точка линии имеет ошибку меньше полпиксела и, следовательно, сдвигается по вертикали на единицу.

Чтобы не сравнивать h_i с 0.5 и не делить на Δx, введем вспомогательную функцию δ_i : $\delta_i = 2 \cdot \Delta x \cdot (h_i - 0.5)$.

Так как $tg\alpha = \dfrac{\Delta y}{\Delta x} = \dfrac{h_i + y_{i-1}}{x_i}$, то $\delta_i = 2(\Delta y \cdot x_i - \Delta x \cdot y_{i-1}) - \Delta x$, $\delta_0 = 2\Delta y - \Delta x$.

Тогда $\Delta y_i = \begin{cases} 0, \text{ если } \delta_i < 0, \\ 1, \text{ если } \delta_i \geq 0. \end{cases}$

От действий с вещественными числами мы избавились, теперь попробуем вычислять значение δ_{i+1} через δ_i и некое приращение $\Delta\delta_{i+1}$.
Пусть $\delta_{i+1} = \delta_i + \Delta\delta_{i+1}$, определим величину приращения:

$$\Delta\delta_{i+1} = \delta_{i+1} - \delta_i = 2\Delta x (h_{i+1} - 0.5) - \delta_i =$$
$$= 2(\Delta y (y_i + 1) - \Delta x\, y_i) - \Delta x - \delta_i = 2\Delta y - 2\Delta x \Delta y_i.$$

Тогда возможные приращения: $\Delta\delta_{i+1} = \begin{cases} 2\Delta y, \text{ если } \delta_i < 0, \\ 2\Delta y - 2\Delta x, \text{ если } \delta_i \geq 0. \end{cases}$

Алгоритм:
начальная инициализация переменных:
$x=x_1$; $y=y_1$;
$\Delta x = x_2 - x_1$; $\Delta y = y_2 - y_1$;
$\delta = 2\Delta y - \Delta x$; $\Delta\delta_1 = 2\Delta y$; $\Delta\delta_2 = 2\Delta x$;
основной цикл:
для *i* от 1 до Δx
{ Рисуем точку (x, y);
пока $\delta \geq 0$ {$y=y+1$; $\delta=\delta-\Delta\delta_2$;}
$\delta=\delta+\Delta\delta_1$; $x=x+1$;}

Такой алгоритм не использует операций с плавающей точкой, не имеет в основном цикле ни делений, ни умножений, и его реализация занимает очень мало места.

Чтобы реализация алгоритма была более полной, необходимо обрабатывать отрезки во всех октантах. Модификацию изложенного выше алгоритма легко сделать, учитывая номер квадранта, в котором лежит отрезок и его угловой коэффициент.

Предполагается, как и раньше, что концы отрезка не совпадают, и являются целыми.

Модифицированный алгоритм Брезенхема:

начальная инициализация переменных:

$x=x_1$; $y=y_1$;

$\Delta x= | x_2 - x_1 |$; $\Delta y=| y_2 - y_1 |$;

$s_1=\text{Sign}(x_2 - x_1)$; $s_2=\text{Sign}(y_2 - y_1)$;

в зависимости от углового коэффициента определяется количество точек отрезка:

если ($\Delta y > \Delta x$), то {$\Delta y <—> \Delta x$; обмен=true;}

иначе обмен=false;

$\delta = 2\Delta y - \Delta x$; $\Delta\delta_1=2\Delta y$; $\Delta\delta_2=2\Delta x$;

основной цикл:

для i от 1 до Δx

{ Рисуем точку (x, y);

пока $\delta \geq 0$ {если (обмен=true), то {$x=x+s_1$;}

иначе {$y=y+s_2$;}

$\delta=\delta-\Delta\delta_2$;}

если (обмен=true), то {$y=y+s_2$;}

иначе {$x=x+s_1$;}

$\delta=\delta+\Delta\delta_1$;}

Линейная интерполяция яркости при построении отрезка по алгоритму Брезенхема получается параметрическим способом.

2.3. Параметрический алгоритм построения окружности

В растр можно раскладывать не только линейную, но и другие, более сложные функции: окружности, эллипсы, параболы, гиперболы, и др. Мы подобно рассмотрим алгоритмы построения окружности.

Рассмотрим окружность с центром в точке (x_0, y_0), и радиусом R.

Каноническое уравнение $(x-x_0)^2+(y-y_0)^2=R^2$.

Тогда $y=y_0 \pm \sqrt{R^2-(x-x_0)^2}$, $x \in [-R + x_0..R + x_0]$.

Параметрическое уравнение окружности:

$$x=x_0+R\cdot \sin \omega t;$$

$$y=y_0+R\cdot \cos \omega t; \text{ где } t = 0\ldots\frac{2\pi}{\omega}$$

Алгоритм рисования окружности:

начальная инициализация переменных:

$x_2=x_0$; $y_2=y_0+R$;

основной цикл:

для α от 1°до 360°

{ $x_1=x_2$; $y_1=y_2$;

x_2=] R sin α [+ x_0; y_2=] R cos α [+ y_0;

Рисуем линию (x_1, y_1) – (x_2, y_2) выбранным цветом;}

Если воспользоваться симметрией окружности, то можно построить более эффективный алгоритм. Если точка (x, y) лежит на окружности, то легко вычислить семь точек, принадлежащих окружности, симметричных этой. То есть, имея функцию вычисления значения y по x=0..$R/\sqrt{2}$ для построения дуги от 0° до 45° можем построить процедуру, которая будет по одной координате ставить восемь точек, симметричных относительно центра окружности.

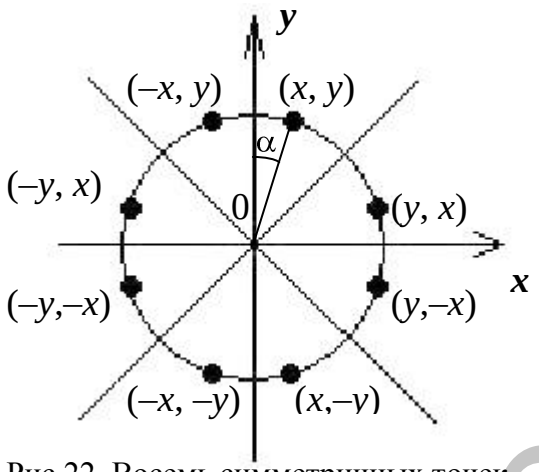


Рис.22. Восемь симметричных точек.

Процедура построения симметричных точек

точки (x_0, y_0, x, y)

{ Рисуем точку (x_0 + x, y_0 + y);

Рисуем точку (x_0 + y, y_0 + x);

Рисуем точку (x_0 + y, y_0 – x);

Рисуем точку (x_0 + x, y_0 – y);

Рисуем точку (x_0 – x, y_0 – y);

Рисуем точку (x_0 – y, y_0 – x);

Рисуем точку (x_0 – y, y_0 + x);

Рисуем точку (x_0 – x, y_0 + y);

}

Алгоритм рисование окружности с центром в точке (x_0, y_0), и радиусом R по точкам:

основной цикл:

для x от 0 до $R/\sqrt{2}$

{ y =] $\sqrt{R^2 - x^2}$ [;

точки (x_0, y_0, x, y);

}

2.4. Алгоритм Брезенхема генерации окружности

Один из наиболее эффективных и простых для понимания алгоритмов генерации окружности принадлежит Брезенхему. Как и в алгоритме Брезенхема для отрезка, для выбора соответствующего пиксела желательно использовать только знак ошибки, а не ее величину.

Как и в предыдущем параграфе, будем рассматривать сегмент окружности, соответствующий x=0.. $R/\sqrt{2}$. На каждом шаге выбираем точку, ближайшую к реальной окружности. В качестве ошибки возьмем величину $D(\mathrm{P}_i)=(x^2_i+ y^2_i) – R^2$.

Рассмотрим рис. 23 , на котором показаны различные возможные способы прохождения истинной окружности через сетку пикселов. Пусть пиксел P_i уже найден как ближайший к реальной изображенной окружности, и теперь требуется определить, какой из пикселов должен быть установлен следующим: T_i или S_i. Для этого определим точку, которой соответствует минимальная ошибка:

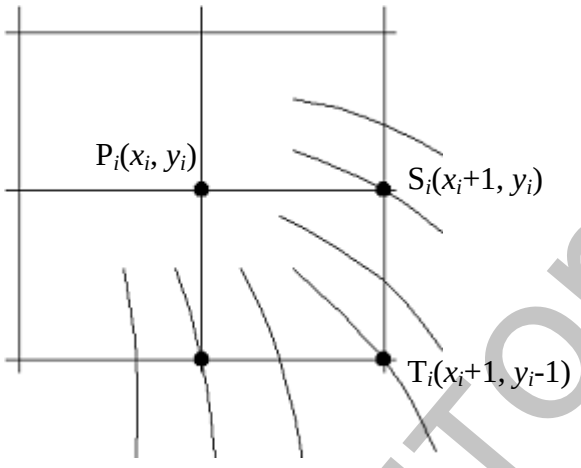


Рис.23.

$$D(\mathrm{S}_i)=((x_i + 1)^2 + y_i^2) – R^2,$$

$$D(\mathrm{T}_i)=((x_i + 1)^2 + (y_i –1)^2) – R^2.$$

Если $|D(\mathrm{S}_i)| < |D(\mathrm{T}_i)|$, то выбираем S_i, иначе – T_i.

Введем величину $d_i = |D(\mathrm{S}_i)| - |D(\mathrm{T}_i)|$.

Если $d_i < 0$, то выбирается S_i , иначе выбирается T_i.

Если рассматривать только часть окружности, дугу от 0° до –45°, то $D(\mathrm{S}_i) > 0$, так как точка S_i лежит за пределами окружности, а $D(\mathrm{T}_i) < 0$, так как T_i находится внутри окружности, поэтому $d_i = D(\mathrm{S}_i) + D(\mathrm{T}_i)$.

Итак, $d_i = D(\mathrm{S}_i) + D(\mathrm{T}_i)$.

Выбираем $\mathrm{S}_i \leftrightarrow d_i < 0$,

$\mathrm{T}_i \leftrightarrow d_i \geq 0$.

Аналогично алгоритму Брезенхема рисования линии найдем приращение: $\Delta d_{i+1}=d_{i+1} – d_i$.

Для нахождения d_{i+1} рассмотрим два случая:

I случай: На i-м шаге мы выбрали точку S_i, т.е. $\mathrm{P}_{i+1}=\mathrm{S}_i$ (см. рис.24).

$$d_{i+1}= D(\mathrm{S}_{i+1}) + D(\mathrm{T}_{i+1}) = ((x_i+2)^2+ y_i^2) – R^2+((x_i +2)^2+ (y_i –1)^2) – R^2.$$
$$d_i = D(\mathrm{S}_i) + D(\mathrm{T}_i) = ((x_i+1)^2+ y_i^2) – R^2+((x_i +1)^2+ (y_i –1)^2) – R^2.$$

$\Delta d_{i+1}=d_{i+1} - d_i = 2((x_i+2)^2 - (x_i+1)^2)=4x_i +6.$

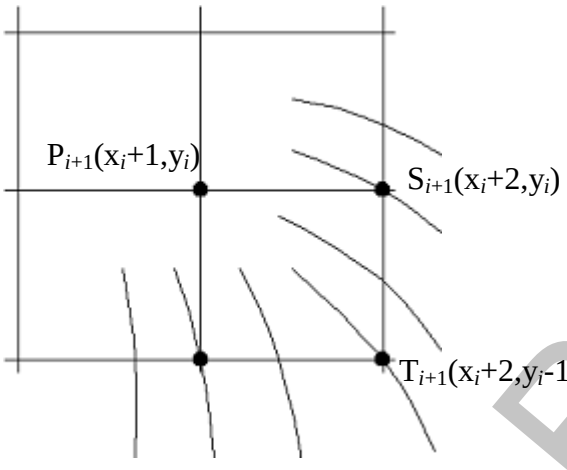


Рис.24.

II случай: На *i*-м шаге мы выбрали точку T_i, т.е. $P_{i+1}=T_i$ (см. рис.25).

$d_{i+1}= D(S_{i+1}) +D(T_{i+1}) = ((x_i+2)^2+ (y_i - 1)^2) - R^2+((x_i +2)^2+ (y_i -2)^2) - R^2.$
$d_i = D(S_i) + D(T_i) = ((x_i+1)^2+ y_i^2) - R^2+((x_i + 1)^2+ (y_i - 1)^2) - R^2.$
$\Delta d_{i+1}=d_{i+1} - d_i = 2((x_i+2)^2 - (x_i+1)^2) + (y_i - 2)^2 - y_i^2 = 4(x_i - y_i) + 10.$

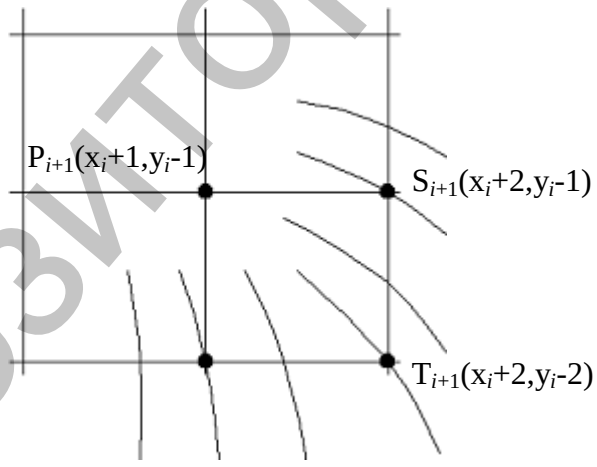


Рис.25.

Учитывая, что P_1 имеет координаты (0, *R*) мы получаем:

$d_1= D(S_1) + D(T_1)=(1^2 + R^2) - R^2 + (1^2 + (R - 1)^2) - R^2=2 - 2R+1=3-2R.$

Когда $d_i < 0$, выбираем S_i , и $\Delta d_{i+1}=4x_i+6$;
Когда $d_i \geq 0$, выбираем T_i , и $\Delta d_{i+1}=4(x_i - y_i) +10$.

Алгоритм хорош тем, что отсутствуют операции с плавающей точкой, а также операции деления и извлечения корня.

2.5. Кривая Безье

Кривая Безье разработана математиком Пьером Безье. Кривые и поверхности Безье были использованы в 60-х годах компанией «Рено» для компьютерного проектирования формы кузовов автомобилей. В настоящее время они широко используются в компьютерной графике.

Кривые Безье описываются в параметрической форме:

$$x = P_x(t)$$

$$y = P_y(t)$$

Значение t выступает как параметр, которому отвечают координаты отдельной точки линии. Параметрическая форма описания может быть более удобной для некоторых кривых, чем задание в виде функции $y = f(x)$. Это потому, что функция $f(x)$ может быть намного сложнее, чем $P_x(t)$ и $P_y(t)$, кроме того, $f(x)$ может быть неоднозначной.

Многочлены Безье для $P_x(t)$ и $P_y(t)$ имеют вид:

$$P_x(t) = \sum_{i=0}^{m} C_m^i t^i (1-t)^{m-i} x_i \ , \quad P_y(t) = \sum_{i=0}^{m} C_m^i t^i (1-t)^{m-i} y_i \ ,$$

где x_i и y_i – координаты точек-ориентиров P_i. $C_m^i = \dfrac{m!}{i!(m-i)!}$ – сочетание m по i (известное по биному Ньютона). Значение m можно рассматривать и как степень полинома, и как значение, которое на единицу меньше количества точек-ориентиров.

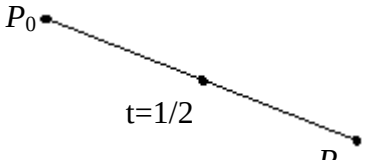


Рис. 26. Кривая Безье (m=1)

При построении сглаживающей кривой по упорядоченному набору точек–ориентиров, ломанная $P_0, P_1 \ldots P_m$ называется *контрольной ломанной*.

Рассмотрим кривые Безье, классифицируя их по значениям m.

m=1 (по двум точкам)

Кривая вырождается в отрезок прямой линии, определяемый концевыми точками P_0 и P_1, как показано на рис. 26.

$P(t)=(1-t)P_0 + t\,P_1$.

m=2 (по трем точкам), рис.27.

$P(t)=(1-t)^2P_0 + 2\,t\,(1-t)\,P_1 + t^2P_2$

Рис. 27. Кривая Безье (m=2)

m=3 (по четырем точкам, кубическая)**.** Используется довольно часто, в особенности в сплайновых кривых.

$P(t)=(1-t)^3P_0+3\,t\,(1-t)^2\,P_1+3\,t\,(1-t)^2\,P_2+\,t^3P_3$.

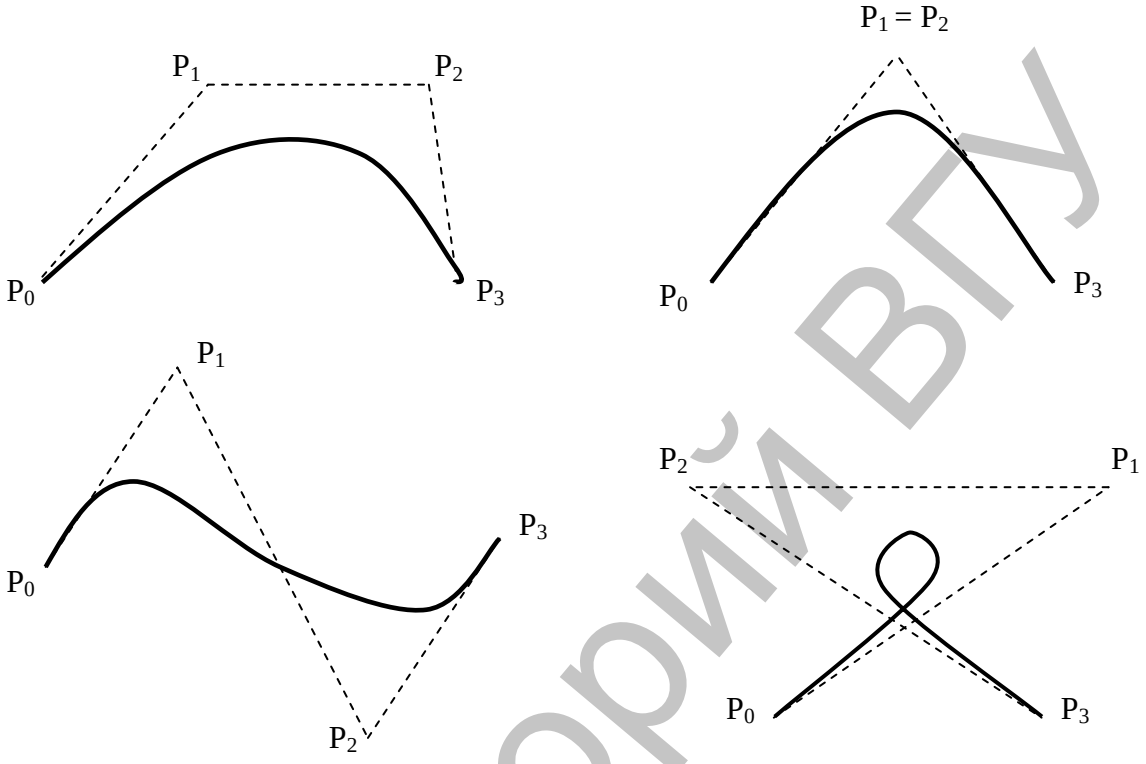


Рис.28. Кубические кривые Безье (m=3)

Алгоритм рисования кривой Безье:

Этот алгоритм позволяет вычислить координаты (*x*,*y*) точки кривой Безье по значению параметра *t*.

1. Каждая сторона контура многоугольника, проходящего по точкам-ориентирам, делится пропорционально значению *t*.

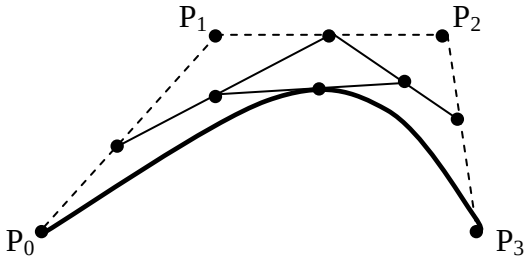


Рис.29. *m*=3; *t* = 0.5

2. Точки деления соединяются отрезками прямых и образуют новый многоугольник. Количество узлов нового контура на единицу меньше, чем количество узлов предыдущего контура.

3. Стороны нового контура снова делятся пропорционально значению *t*. И так далее. Это продолжается до тех пор, пока не будет получена единственная точка деления. Эта точка и будет точкой кривой Безье.

Алгоритм:

цикл по параметру t:

{ *формируем вспомогательный массив R[ ] с исходными точками–ориентирами*:

для i от 0 до m { $R[i]$=P[i];}

находим координаты очередной точки кривой Безье :

для j от m до 0

{

для i от 0 до j {$R[i]=R[i] + t\ (R[i+1] - R[i])$;}

}

}

Результат работы алгоритма – координаты одной точки кривой Безье– записываются в R[0].

Кривая Безье обладает следующими *свойствами*:

1. является гладкой;
2. начинается в точке P_0 и заканчивается в точке P_m, касаясь при этом отрезков P_0P_1 и $P_{m-1}P_m$ контрольной ломанной;
3. функциональные коэффициенты $C_m^i t^i (1-t)^{m-i}$ при вершинах P_i, есть универсальные многочлены Бернштейна, они неотрицательны, и их сумма равна единице.

Поэтому кривая Безье целиком лежит в выпуклой оболочке, порождаемой множеством $\{P_i\}$.

Недостатки кривых Безье:

1. степень функциональных коэффициентов связана с количеством точек–ориентиров;
2. при добавлении хотя бы одной точки в заданный набор, необходимо провести полный пересчет коэффициентов в параметрическом уравнении кривой;
3. изменение хотя бы одной точки приводит к заметному изменению вида всей кривой.

Поэтому на практике удобнее пользоваться кривыми, составленными из элементов кривых Безье (кубических) (необходимо выполнение условия гладкости в точке стыковки).

§ 3. Геометрические преобразования на плоскости

В этом параграфе рассмотрены лежащие в основе компьютерной графики положения математики, необходимые для представления и преобразования точек и линий. При использовании компьютерной графики по желанию пользователя можно менять масштаб изображения, вращать его, смещать и трансформировать для улучшения наглядности перспективного изображения объекта.

3.1. Аффинные преобразования на плоскости

Рассмотрим четыре типа преобразований: параллельный перенос, масштабирование, поворот и отображение.

Полагаем, что на плоскости задана двумерная прямоугольная система координат. Каждой точке ставится в соответствие упорядоченная пара чисел (x, y) ее координат. Вводя на плоскости еще одну прямоугольную систему координат, мы ставим в соответствие той же точке другую пару чисел – (x', y'). Значения (x', y') можно трактовать как координаты точки в новой системе координат.

Аффинное преобразование – преобразование подобия, от лат *affinis* – родственный, подобный.

Аффинное преобразование координат (x, y) описывается формулами:

$$x' = Ax + By + E,$$

$$y' = Cx + Dy + F, \quad \text{где } A, B, C, D, E, F \text{ – константы.}$$

Любое изменение координат, описываемое этими формулами можно представить посредством комбинации операций: параллельного переноса; поворота; зеркального отображения; растяжения (сжатия). На практике (в программировании) покоординатная форма записи элементарных преобразований, как правило, не используется, и ее заменяют на более удобную, матричную запись.

В матричной форме:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} E \\ F \end{bmatrix}$$

Обратное преобразование (x', y') в (x, y) также является аффинным:

$$x = A'x' + B'y' + E',$$

$$y = C'x' + D'y' + F', \quad \text{где } A', B', C', D', E', F' \text{ – константы.}$$

Рассмотрим более подробно некоторые аффинные преобразования.

Параллельный перенос на вектор $\vec{T} = (t_x, t_y)$:

$$x' = x + t_x,$$
$$y' = y + t_y.$$

в векторной форме:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Масштабирование относительно начала координат с коэффициентами k_x, k_y по осям имеет вид:

$$x' = x \cdot k_x,$$
$$y' = y \cdot k_y.$$

в матричной форме:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = K \cdot \begin{bmatrix} x \\ y \end{bmatrix}, \text{ где } K = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}$$

Поворот. Будем считать, что направление *против часовой стрелки* определяет *положительный угол* поворота. Тогда преобразование поворота относительно начала координат на угол α имеет вид:

$$x' = x \cos\alpha - y \sin\alpha,$$
$$y' = x \sin\alpha + y \cos\alpha.$$

в матричной форме:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\alpha) \cdot \begin{bmatrix} x \\ y \end{bmatrix}, \text{ где } R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$$

Отображение. В то время как чистое двумерное вращение в плоскости *Oxy* осуществляется вокруг оси, перпендикулярной к плоскости *Oxy*, *отображение определяется поворотом на 180° вокруг оси, лежащей в плоскости Oxy.*

Вращение около прямой $y=x$ в матричной форме:

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

Вращение около оси ординат (прямой $y=0$):

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

Вращение около оси абсцисс (прямой $x=0$):

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

3.2. Однородные координаты

В предыдущем параграфе были рассмотрены четыре вида преобразований точек на плоскости. Три из них – операции вращения, отображения и масштабирования – описываются в виде произведения матрицы на вектор, а четвертая – операция переноса – описывается как сумма двух векторов. В случае последовательного выполнения любой комбинации

операций вращения, отображения и масштабирования результат легко можно записать в виде произведения матриц соответствующих преобразований. Это будет матрица результирующего поворота и масштабирования. Очевидно, что удобнее применять результирующую матрицу вместо того, чтобы каждый раз заново вычислять произведение матриц. Однако, таким способом нельзя получить результирующую матрицу преобразования, если среди последовательности преобразований присутствует хотя бы один перенос. Матричное произведение в компьютерной графике также называют *композицией*. Было бы удобнее иметь математический аппарат, позволяющий включать в композиции преобразований все четыре выше указанные операции. При этом получился бы значительный выигрыш в скорости вычислений. Однородные координаты и есть этот математический аппарат.

Двумерными однородными координатами произвольной точки плоскости, заданной координатами (x, y), называется любая тройка одновременно неравных нулю чисел X, Y, H, связанных с заданными координатами (x, y) следующими соотношениями:

$$x = \frac{X}{H}; y = \frac{Y}{H}; H \neq 0.$$

Однородные координаты можно представить вложение промасштабированной с коэффициентом H двумерной плоскости в плоскость $Z = H$ в трехмерном пространстве (рис. 30), или как промасштабированные с коэффициентом H значения двумерных координат, расположенные в плоскости $Z = H$: $X=xH$, $Y=yH$: $(x, y) \leftrightarrow [X, Y, H]^T$.

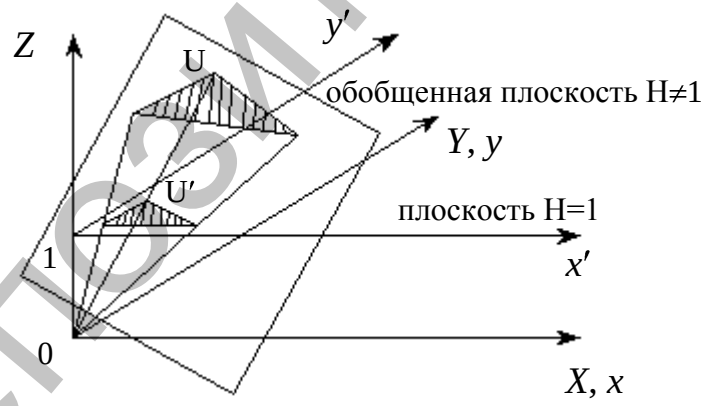


Рис.30. Геометрическое представление однородных координат.

Представление двумерного вектора трехмерным или в общем случае n–мерного (n+1)–мерным называют *однородным координатным воспроизведением*. При однородном координатном воспроизведении n–мерного вектора оно выполняется в (n+1)–мерном пространстве, и конечные результаты в n–мерном пространстве получают с помощью обратного преоб-

разования. Обратное преобразование называется *проекцией* однородных координат.

В силу произвольности значения H в однородных координатах не существует единственного представления точки, заданной в декартовых координатах. Для простоты вычислений выбираем H=1.

Геометрически все преобразования *x* и *y* происходят в плоскости H=1 после нормализации преобразованных координат.

В однородных координатах преобразования параллельного переноса и масштабирования, отображения и поворота относительно центра координат все имеют одинаковую форму: произведение матрицы преобразования на вектор исходных координат.

При помощи троек однородных координат и матриц третьего порядка можно описать любое аффинное преобразование плоскости.

Запишем в виде произведения рассмотренные выше преобразования.

Параллельный перенос определяется произведением матрицы сдвига и вектора исходных координат:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T(t_x, t_y) \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \text{ где } T(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Обратное преобразование:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T^{-1}(t_x, t_y) \cdot \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}, \text{ где } T^{-1}(t_x, t_y) = T(-t_x, -t_y)$$

Масштабирование определяется произведением матрицы масштабирования и вектора исходных координат:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = K(k_x, k_y) \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \text{ где } K(k_x, k_y) = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Обратное преобразование:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K^{-1}(k_x, k_y) \cdot \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}, \text{ где } K^{-1}(k_x, k_y) = K(1/k_x, 1/k_y)$$

Поворот определяется произведением матрицы поворота и вектора исходных координат:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = R(\alpha) \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \text{ где } R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Обратное преобразование:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = R^{-1}(\alpha) \cdot \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}, \text{ где } R^{-1}(\alpha) = R(-\alpha)$$

Отображение относительно прямой $ax+by+c=0$ определяется произведением матрицы отображения и вектора исходных координат:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = S(a,b,c) \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \text{ где } S(a,b,c) = \begin{bmatrix} \frac{b^2-a^2}{b^2+a^2} & \frac{-2ab}{b^2+a^2} & \frac{-2ac}{b^2+a^2} \\ \frac{-2ab}{b^2+a^2} & \frac{a^2-b^2}{b^2+a^2} & \frac{-2bc}{b^2+a^2} \\ 0 & 0 & 1 \end{bmatrix}$$

Очевидно, что $S^{-1}(a, b, c)= S(a, b, c)$.

3.3. Композиция двумерных преобразований

Последовательное выполнение нескольких преобразований можно представить в виде единственной матрицы суммарного преобразования. Умножение на единственную матрицу, естественно выполнять быстрее, чем последовательное умножение нескольких матриц.

Рассмотрим сдвиг точки (x, y) на вектор (x_1, y_1), а затем на вектор (x_2, y_2).

Суммарная матрица сдвига такого преобразования равна

$$T = \begin{bmatrix} 1 & 0 & x_2 \\ 0 & 1 & y_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_1 + x_2 \\ 0 & 1 & y_1 + y_2 \\ 0 & 0 & 1 \end{bmatrix}$$

Получили, что результирующий сдвиг–сдвиг на вектор (x_1+x_2, y_1+y_2). Таким образом, мы получили, что *сдвиг аддитивен*, т.е. последовательное выполнение двух сдвигов эквивалентно одному суммарному сдвигу.

Рассмотрим последовательное выполнение операций масштабирования, первая с коэффициентами (k_{x1}, k_{y1}), вторая – (k_{x2}, k_{y2}).

$$K = \begin{bmatrix} k_{x2} & 0 & 0 \\ 0 & k_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} k_{x1} & 0 & 0 \\ 0 & k_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} k_{x1}k_{x2} & 0 & 0 \\ 0 & k_{y1}k_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Получили, что результирующее масштабирование есть масштабирование с коэффициентами – $(k_{x1}\cdot k_{x2}, k_{y1}\cdot k_{y2})$, т.е. *суммарное масштабирование мультипликативно.*

Покажем, что *два последовательных поворота аддитивны*.

$$R = R(\beta)R(\alpha) = \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} \cos(\alpha+\beta) & -\sin(\alpha+\beta) & 0 \\ \sin(\alpha+\beta) & \cos(\alpha+\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = R(\alpha+\beta).$$

Пример: Рассмотрим задачу поворота на угол α вокруг фиксированной точки (x_0, y_0).

Разделим задачу на три более простых подзадачи:

1. преобразование смещения $\vec{T} = (-x_0, -y_0)$, для совмещения центра поворота с началом координат;
2. преобразование поворота $R(\alpha)$ вокруг начала координат;
3. преобразование смещения $\vec{T} = (x_0, y_0)$, для возвращения центра поворота в прежнее положение.

Рассмотренные выше операции преобразования можно описать в виде произведения:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = M \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \text{где } M = T(x_0, y_0) \cdot R(\alpha) \cdot T(-x_0, -y_0).$$

В результате произведения матриц получаем матрицу преобразования поворота на угол α вокруг фиксированной точки (x_0, y_0):

$$M = \begin{bmatrix} \cos\alpha & -\sin\alpha & -x_0\cos\alpha + y_0\sin\alpha + x_0 \\ \sin\alpha & \cos\alpha & -x_0\sin\alpha - y_0\cos\alpha + y_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

Вернемся к декартовым координатам:

$$x' = m_{11}x + m_{12}y + m_{13},$$
$$y' = m_{21}x + m_{22}y + m_{23}.$$

Очевидно, что матрицы 3 × 3 удобны при вычислении суммарного преобразования. Однако, для нахождение преобразованных однородных координат необходимо выполнить 9 операций умножения и 6 операций сложения. Но так как третья однородная координата может быть равна 1, а третья строка содержать единственный ненулевой элемент, равный 1, то преобразование в декартовых координатах потребует лишь 4 операции умножения и 4 операции сложения. Таким образом, для большей эффектив-

ности, фактическое преобразование координат следует проводить с учетом реальной структуры матрицы преобразований.

3.4. Общий вид матрицы преобразований

Матрицу преобразования для двумерных однородных координат в общем виде можно записать следующим образом:

$$\left[\begin{array}{cc:c} A & B & E \\ C & D & F \\ \hdashline P & Q & W \end{array}\right]$$

Элементы *A, B, C, D* – определяют изменение масштаба, сдвиг и вращение; *E, F* определяют смещение, а *P, Q* – получение проекций. Элемент *W* производит полное изменение масштаба. Чтобы показать это, рассмотрим преобразование:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \xrightarrow{\textit{преобразование}} \begin{bmatrix} X' \\ Y' \\ H' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & W \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ W \end{bmatrix}$$

Проведем нормализацию преобразованных однородных координат:

$$\begin{bmatrix} X' \\ Y' \\ H' \end{bmatrix} = \begin{bmatrix} x \\ y \\ W \end{bmatrix} \xrightarrow{\textit{нормализация}} \begin{bmatrix} x/W \\ y/W \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Таким образом,

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \xrightarrow{\textit{преобразование}} \begin{bmatrix} x/W \\ y/W \\ 1 \end{bmatrix}$$

имеет место однородное изменение масштаба вектора положения. При $W<1$ происходит увеличение, а при $W>1$ уменьшение масштаба.

Рассмотрим влияние элементов *P* и *Q* матрицы преобразования.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \xrightarrow{\textit{преобразование}} \begin{bmatrix} X' \\ Y' \\ H' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ P & Q & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ Px + Qy + 1 \end{bmatrix}$$

Переменная $H' = Px+Qy+1$, которая определяет плоскость, содержащую преобразованные точки, представленные в однородных координатах, теперь образует уравнение плоскости в трехмерном пространстве.

$$\begin{bmatrix} X' \\ Y' \\ H' \end{bmatrix} = \begin{bmatrix} x \\ y \\ Px+Qy+1 \end{bmatrix} \xrightarrow{\text{нормализация}} \begin{bmatrix} x/(Px+Qy+1) \\ y/(Px+Qy+1) \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Покажем, что двумерные однородные координаты для преобразованной точки $[x', y',1]^T$ получаются в результате проецирования плоскости $Z=Px+Qy+1$ на плоскость $Z=1$ с центром проекции в начале координат. Для этого покажем, что прямая, проходящая через точки $U(X',Y',H')$ и $U'(x',y',1)$, проходит и через начало координат (см. рис.30).

Направляющий вектор прямой UU' равен $(X'-x', Y'-y', H'-1)$. Запишем параметрическое уравнение этой прямой:

$$x(t) = x' + (X' - x') \cdot t = x/H' + (x - x/H') \cdot t,$$
$$y(t) = y' + (Y' - y') \cdot t = y/H' + (y - y/H') \cdot t,$$
$$z(t) = 1 + (Px + Qy) \cdot t = 1 + (H' - 1) \cdot t.$$

Из первого уравнения находим, что $x(t)=0$ при $t = 1/(1-H')$. Подставляя это значение в оставшиеся уравнения, получим что и $y(t)=0$ и $z(t)=0$.

Таким образом, мы показали, что элементы P и Q матрицы преобразования определяют проецирование с центром проекции в начале координат.

3.5. Декартовые точки с бесконечными координатами. Параллельные прямые

Рассмотрим в декартовой системе линию, проходящую через начало координат и точку (x, y). Однородные координаты этой точки $[X, Y, H]^T = [xH, yH, H]^T$, где H имеет произвольное значение.

Заметим, что в то время как декартовы координаты стремятся к бесконечности, при $H \to 0$: $x = \frac{X}{H} \xrightarrow[H \to 0]{} \infty$, $y = \frac{Y}{H} \xrightarrow[H \to 0]{} \infty$, предел отношения при этом может быть конечен:

$$\frac{x}{y} \xrightarrow[H \to 0]{} \frac{X}{Y}.$$

Таким образом, *точка с однородными координатами* $[x, y, 0]^T$ *задает в декартовой системе точку на бесконечности для рассматриваемой прямой.* Например, точка с однородными координатами (1, 0, 0) задает бесконечную точку на оси *О*x, а точка с однородными координатами (0,1,0) задает бесконечную точку на оси *Оy*.

Покажем, что *прямые, параллельные в декартовой системе координат, в однородных координатах имеют точку пересечения.* Эта особенность используется при анализе перспективных преобразований.

Пусть две пересекающиеся прямые в декартовой системе координат заданы системой уравнений:

$$A_1x+B_1y+C_1=0,$$
$$A_2x+B_2y+C_2=0.$$

Тогда точка их пересечения в однородных координатах имеет вид:

$$\left[\frac{B_2C_1-B_1C_2}{A_1B_2-A_2B_1}, \frac{A_1C_2-A_2C_1}{A_1B_2-A_2B_1}, 1\right]^T$$

В силу произвольности масштабирующего множителя, умножим значении координат на $A_1B_2–A_2B_1$:

$$\left[B_2C_1-B_1C_2,\ A_1C_2-A_2C_1,\ A_1B_2-A_2B_1\right]^T$$

Если прямые параллельны, то определитель $A_1B_2–A_2B_1$ равен нулю. Учитывая это, и обозначив первые две координаты через X_0 и Y_0, получаем координату точки пересечения параллельных прямых в однородной системе координат: $\left[X_0,\ Y_0,\ 0\right]^T$.

При этом точка пересечения лежит на прямой $–Y_0\cdot x+ X_0\cdot y=0$ на бесконечности.

§ 4. Трехмерные преобразования и проекции
4.1. Геометрические преобразования в пространстве

При рассмотрении трехмерных преобразований, в основном, используется общепринятая в векторной алгебре *правая система координат* (рис. 31, справа). При этом, если смотреть со стороны положительной полуоси в центр координат, то поворот на +90° (против часовой стрелки) переводит одну положительную ось в другую (направление движения расположенного вдоль оси и поворачивающегося против часовой стрелки правого винта и положительной полуоси совпадают). Если же поворот производится по часовой стрелке – то система координат левая (см. рис. 31, слева).

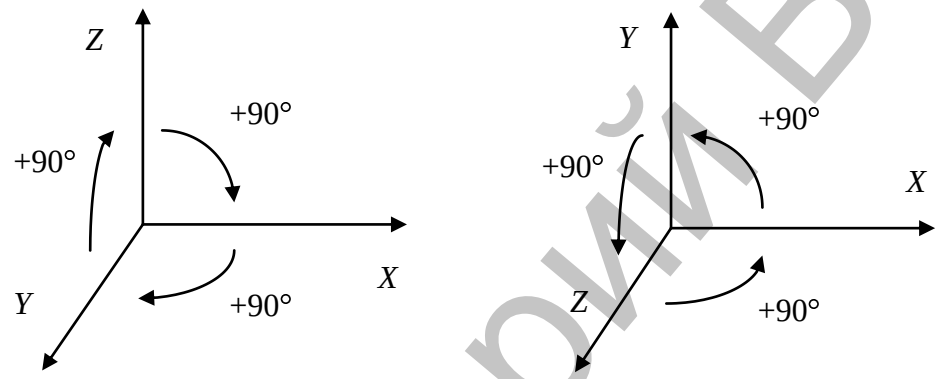


Рис. 31. Левая и правая система координат.

В трехмерной машинной графике более удобной является левая система координат. Так если, например, поверхность экрана совмещена с плоскостью *XY*, то большим удалениям от наблюдателя соответствуют точки с большим значением *Z*.

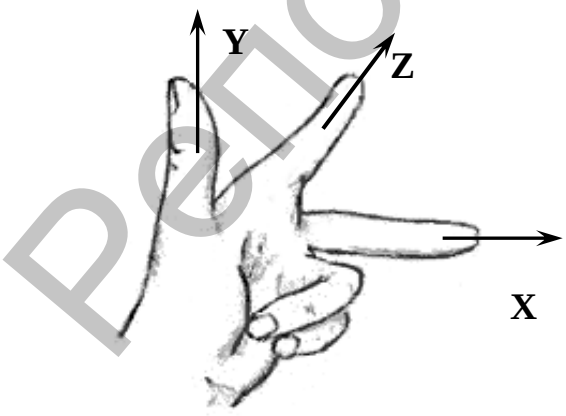


Рис. 32. Определение левосторонней системы координат по левой руке.

Существует также легкий способ определения вида системы координат по правой или левой руке, как показано на рис. 32. Для левой руки большой, указательный и средний пальцы формируют левую тройку ортогональных векторов. То же относится и к их циклическим перестановкам.

Работа с однородными трехмерными координатами и матрицами преобразования (формирование и композиция) аналогична двумерному случаю, поэтому здесь будут рассмотрены только матрицы преобразований сдвига, масштабирования, отображения и поворота.

Подобно тому как в двумерном случае точка в однородных координатах представляется трехмерным вектором $[X, Y, H]^{\mathrm{T}}$, а матрицы преобразований имеют размер 3×3, для трехмерного случая точка представляется четырехмерным вектором $[X, Y, Z, H]^{T}$, где H произвольное, не равное нулю, а матрицы преобразований имеют размер 4×4.

Формулы для преобразования: $x = \frac{X}{H};\ y = \frac{Y}{H};\ z = \frac{Z}{H};\quad H \neq 0;$

Параллельный перенос определяется произведением матрицы сдвига и вектора исходных координат:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = T(t_x, t_y, t_z) \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},\ \text{где}\ T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Обратное преобразование:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = T^{-1}(t_x, t_y, t_z) \cdot \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix},\ \text{где}\ T^{-1}(t_x, t_y, t_z) = T(-t_x, -t_y, -t_z)$$

Масштабирование определяется произведением матрицы масштабирования и вектора исходных координат:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = K(k_x, k_y, k_z) \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},\ \text{где}\ K(k_x, k_y, k_z) = \begin{bmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Обратное преобразование:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = K(k_x, k_y, k_z) \cdot \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix},\ \text{где}\ K^{-1}(k_x, k_y, k_z) = K(1/k_x, 1/k_y, 1/k_z)$$

Отображение относительно плоскости *Oxy* определяется произведением матрицы отображения и вектора исходных координат:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = S_z \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},\ \text{где}\ S_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Вид матрицы отображения очевиден и для отображения относительно плоскостей *Oxz* и *Oyz* поскольку отображение относительно плоскости

Oxy изменяет только знак координаты *z*, отображение относительно плоскости *Oxz* изменяет только знак координаты *y*, отображение относительно плоскости *Oyz* изменяет только знак координаты *x*.

Отображение относительно других плоскостей можно получить путем комбинации сдвига, поворота и рассмотренного выше отображения.

Поворот. Особенностью поворота в 3-х мерном пространстве является то, что один поворот в пространстве следует раскладывать на 3 поворота: вокруг оси абсцисс, вокруг оси ординат, вокруг оси аппликат. Вторая особенность – это направление угла поворота. Существует договоренность, что *положительным направлением угла поворота* для обеих систем координат считается то направление, при котором, если смотреть с положительного конца полуоси в центр координат, осуществляется кратчайший переход от одной положительной полуоси к другой: для *X*: *Y*→*Z*; для *Y*: *Z*→*X*; для *Z: X*→*Y*. В левой системе координат положительными будут повороты по часовой стрелке, для правой– против часовой стрелки (рис. 31).

Ранее рассмотренная для двумерного случая матрица поворота является в то же время трехмерным поворотом вокруг оси *Oz*. Так как при трехмерном повороте вокруг оси *Oz* (поворот в плоскости *XY*) на угол φ_z размеры вдоль оси *Oz* неизменны, то все элементы третьей строки и третьего столбца равны 0, кроме диагонального, равного 1:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = R(\varphi_z) \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \text{ где } R(\varphi_z) = \begin{bmatrix} \cos\varphi_z & -\sin\varphi_z & 0 & 0 \\ \sin\varphi_z & \cos\varphi_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Матрица обратного преобразования: $R^{-1}(\varphi_z) = R(-\varphi_z)$.

При повороте вокруг оси *Ox* (в плоскости *YZ*) на угол φ_x размеры вдоль оси *Ox* не меняются, поэтому все элементы первой строки и первого столбца равны 0, за исключением диагонального, равного 1:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = R(\varphi_x) \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \text{ где } R(\varphi_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi_z & -\sin\varphi_z & 0 \\ 0 & \sin\varphi_z & \cos\varphi_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Матрица обратного преобразования: $R^{-1}(\varphi_x) = R(-\varphi_x)$.

При повороте вокруг оси *Oy* (в плоскости *XZ*) на угол φ_y размеры вдоль оси *Oy* не меняются, поэтому все элементы второй строки и второго столбца равны 0, за исключением диагонального, равного 1:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = R(\varphi_y) \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \text{ где } R(\varphi_y) = \begin{bmatrix} \cos\varphi_y & 0 & -\sin\varphi_z & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi_y & 0 & \cos\varphi_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Матрица обратного преобразования: $R^{-1}(\varphi_y) = R(-\varphi_y)$.

В общем случае любой поворот в пространстве может быть описан с помощью некоторых комбинаций этих трех поворотов. Поскольку повороты описываются умножением матриц, то трехмерные вращения некоммутативны, т.е. порядок умножения влияет на конечный результат.

4.2. Аксонометрическая и перспективная проекции

При отображении трехмерных объектов на экране или бумаге необходимо знать, как происходит преобразование проекции трехмерного пространства в двумерное пространство.

Проецирование выполняется с помощью проецирующих лучей (проекторов), выходящих из центра проекции, пересекающих проекционную (картинную) плоскость, и проходящих через каждую точку трехмерного объекта. Тем самым точки пересечения проекторов с картинной плоскостью образуют проекцию объекта.

Тип проецирования на плоскую, а не искривленную поверхность, где в качестве проекторов используются прямые, а не искривленные линии, называется *плоской геометрической проекцией*. По расположению центра проекции относительно плоскости проекции различают: центральные и параллельные проекции.

Если центр проекции находится на конечном расстоянии от проекционной плоскости, то проекция – *центральная*. Если же центр проекции удален на бесконечность, то проекция – *параллельная*.

При **параллельной проекции** центр проекции находится на бесконечном расстоянии от плоскости проекции. Проекторы представляют собой пучок параллельных лучей. В этом случае необходимо задавать направление проецирования и расположение плоскости проекции. По взаимному расположению проекторов, плоскости проекции и главных осей координат различаются:

- ортогональные,
- прямоугольные аксонометрические,
- косоугольные аксонометрические проекции.

При *ортогональном* проецировании проекторы перпендикулярны плоскости проекции, а плоскость проекции перпендикулярна главной оси.

При *прямоугольном аксонометрическом* проецировании проекторы перпендикулярны плоскости проекции, которая расположена под углом к главной оси.

При *косоугольном аксонометрическом* проецировании проекторы расположена под углом к плоскости проекции, а плоскость проекции перпендикулярна главной оси.

Изображение, полученное при параллельном проецировании, не достаточно реалистично, но передает точные формы и размеры, хотя и возможно различное укорачивание для различных осей.

При **центральной проекции** расстояние от центра проекции до плоскости проецирования конечно, поэтому проекторы представляют собой пучок лучей, исходящих из центра проекции. В этом случае необходимо задавать расположение и центра проекции и плоскости проекции.

Изображения на плоскости проекции имеют перспективные искажения, т.к. размер видимого изображения зависит от взаимного расположения центра проекции, объекта и плоскости проекции. Из-за перспективных искажений изображения, полученные центральной проекцией более реалистичны, но неверно передают форму и размены объекта.

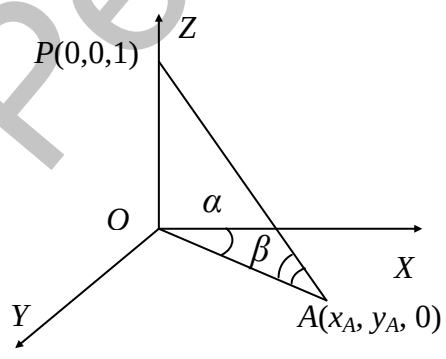
В зависимости от того по скольким осям выполняется перспективное искажение различают

- одноточечные;
- двухточечные;
- трехточечные центральные проекции.

Будем считать, что плоскость экрана монитора совпадает с проекционной плоскостью. Запишем выражения для матриц преобразования.

Ортогональное проецирование, выполняется на плоскость, перпендикулярную какой-либо оси, например на плоскость $Z=Z_0$.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ Z_0 \\ 1 \end{bmatrix}$$



Рассмотрим *косоугольное аксонометрическое проецирование*. В этом случае плоскость проецирования перпендикулярна главной оси, например оси *Z*, а проекторы составляют с плоскостью проецирования угол, не равный 90 °.

Рис. 33. Косоугольная аксонометрическая проекция точки *P* в точку *A*.

Из рис.33 видно, что x_A=|OA|·cos α, y_A=|OA|·sin α, |OA|=ctg β.

Теперь, проектором, параллельным рассмотренному, спроецируем некоторую точку M(x, y, z) в точку N(x_N, y_N, 0).

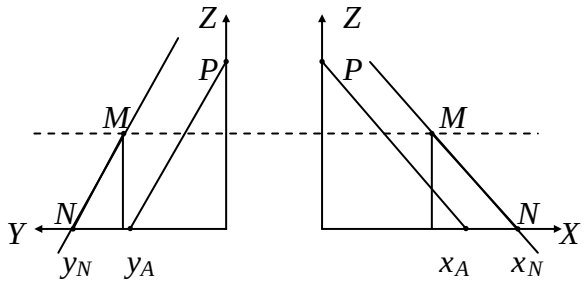


Рис. 34 Косоугольная аксонометрическая проекция точки M в точку N.

Из подобия треугольников получаем: $\frac{x_N - x}{z} = \frac{x_A}{1}$, $\frac{y_N - y}{z} = \frac{y_A}{1}$.

Откуда следует, что $x_N = x + z \cdot x_A$, $y_N = y + z \cdot y_A$.

В матричном виде косоугольное аксонометрическое проецирование на плоскость $Z=Z_0$ имеет вид:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \operatorname{ctg}\beta\cos\alpha & 0 \\ 0 & 1 & \operatorname{ctg}\beta\sin\alpha & 0 \\ 0 & 0 & 0 & Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + z\operatorname{ctg}\beta\cos\alpha \\ y + z\operatorname{ctg}\beta\sin\alpha \\ Z_0 \\ 1 \end{bmatrix}$$

Выведем матрицу, определяющую центральное проецирование для простого случая одноточечной проекции, когда плоскость проекции перпендикулярна оси Z и расположена на расстоянии d от начала координат. Центр проекции (точка просмотра) находится в начале координат.

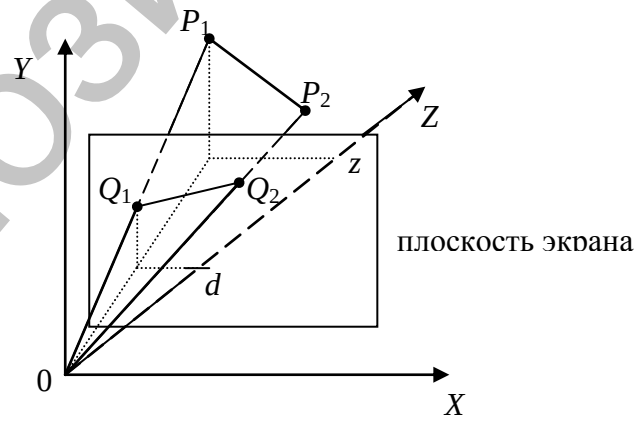


Рис. 35. Центральная проекция отрезка P_1P_2 в отрезок Q_1Q_2.

В результате проецирования точка $P_1(x, y, z)$ переходит в точку $Q_1(x', y', d)$. В этом случае выполняются следующие соотношения:

$\frac{x}{z} = \frac{x'}{d}$, $\frac{y}{z} = \frac{y'}{d}$, откуда $x' = \frac{x}{z/d}$, $y' = \frac{y}{z/d}$.

В матричном виде:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ H' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

Приведенные выше формулы легки для расчета, однако не очень удобны, когда необходимо приближать или удалять от наблюдателя проекционную плоскость.

Предположим, что экран находится в плоскости Z=0 и наблюдатель находится в точке (0, 0, –d).

В результате проецирования точка $P_1(x, y, z)$ переходит в точку $Q_1(x', y', 0)$. Как и выше, из подобия треугольников находим, что:

$$\frac{x}{z+d} = \frac{x'}{d}, \quad \frac{y}{z+d} = \frac{y'}{d}, \text{ откуда } x' = \frac{x}{z/d+1}, \; y' = \frac{y}{z/d+1}.$$

В матричном виде:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ H' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d+1 \end{bmatrix}$$

Это преобразование можно рассматривать как композицию преобразований: перенос начала координат в центр проецирования, собственно проецирование и обратный сдвиг начала координат.

Литература

1. Адамс, М. Математические основы машинной графики / М. Адамс, А. Роджерс. – М.: Мир, 1979.
2. Порев В. Компьютерная графика / В. Пореев. – С.-Пб.: БХВ - Санкт-Петербург, 2002.
3. Препарата Ф. Вычислительная геометрия: введение / Ф. Препарата, М. Шеймос. – М.: Мир, 1989.
4. Роджерс Д. Алгоритмические основы машинной графики / Д. Роджерс. – М.: Мир, 1989.
5. Фоли Д. Основы интерактивной машинной графики / Д. Фоли, А. ван Дэм. – М.: Мир, 1985.
6. Вельтмандер П. В. Машинная графика: Учебное пособие в 3-х книгах. Книга 2. Основные алгоритмы / П. В. Вельтмандер. – Новосибирск: НГУ, 1997.
7. Казанцев А.В. Основы компьютерной графики. Часть 1. Математический аппарат компьютерной графики / А.В. Казанцев. – Казань: КГУ, 2001.
8. Ласло М. Вычислительная геометрия и компьютерная графика на С++ / М. Ласло. – М.: Бином, 1997.
9. Фокс А. Вычислительная геометрия / А. Фокс, М. Пратт. – М.: Мир, 1982.
10. Боресков А.Б. Компьютерная графика: первое знакомство / А.Б. Боресков, Е.В. Шикина, Г.Е. Шикина; под ред. Е.В.Шикина. – М.: Финансы и статистика, 1996.