акДЕЛАЙТЕ ВСЕ АККУРАТНО ПЖ, НЕ УДАЛЯЙТЕ ТО, ЧТО УЖЕ ЕСТЬ

В содержании кликабельные ссылки на эти вопросы из другого файлика, где все расписано

Форматирование:

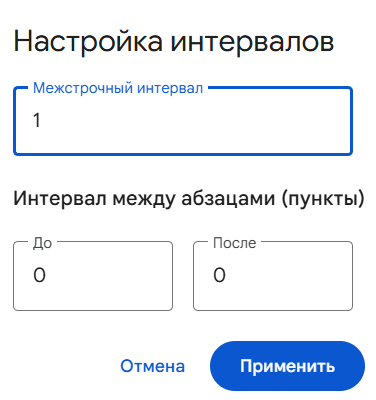
шрифт Montserrat

Обычный текст - размер шрифта 5

Название билета - размер шрифта 6 и жирный

Цвет текста - черный, в крайнем случае синий (всякий желтый не будет видно в чб печати)

Интервалы (важно тыкнуть применить, потому что когда открываешь посмотреть он и так по умолчанию 00 типо стоит, но это обманка):



Давайте по возможности писать дано, найти и асимптотику для каждого алгоритма(ну и ограничения само собой)

**Распределение(предположительно):**

| Имя | Вопросы |
| --- | --- |
| лизОК | 1,11,21,31,41 |
| димуля я | 2,12,22,32,42 |
| полик | 3,13,23,33,43 |
| юля | 4,14,44 |
| сеня | 5,45 |
| настя | 6,16,26,36,46 |
| XUXA | 7,17,27,37,47 |
| ДИДИ | 8,18,28,38,48 |
| Игорь | 9,19,29,39 |
| Сочурка | 10,20,30,(должен пива) |
| Паша | 40 (+ можете еще 4 накидать) |
| Влад | 15, 24 |
| Дима Г | 25, 34 |
| Артём | 35, 49 |

**Легенда:**

* Тема готова
* Тема делается
* Сомнительно, но окей

[1 вопрос. Машинная графика. Задачи, области применения. 4](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.30j0zll)

[2 вопрос. Классификация применения машинной графики. 6](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1fob9te)

[3 вопрос. Основные понятия машинной графики 7](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3znysh7)

[4 вопрос. Графические пакеты. Взаимодействие графического пакета с ОС и прикладными программами. Блок-схема графического пакета и графической системы. 9](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.2et92p0)

[5 вопрос. Алгоритмы растровой машинной графики. Развертка отрезка (алгоритм Брезенхема). 12](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.tyjcwt)

[6 вопрос. Алгоритмы растровой машинной графики. Развертка отрезка (алгоритм Брезенхема). 15](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3dy6vkm)

[7 вопрос. Развертка многоугольников (алгоритм с упорядоченным списком ребер) 18](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1t3h5sf)

[8 вопрос. Алгоритм заполнения гранично-заданных областей. Построчный алгоритм, рекурсивный алгоритм 20](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.4d34og8)

[9 вопрос. Геометрические преобразования. Однородные координаты, композиция преобразований (масштабирование, поворот, перенос на плоскости и в пространстве). 22](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.2s8eyo1)

[10 вопрос. Конические сечения. Параметрическое задание окружности, эллипса, параболы, гиперболы. Построение наилучшего приближения с этим кривым. 26](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.17dp8vu)

[11 вопрос. Представление пространственных форм. Полигональные сетки. 31](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3rdcrjn)

[12 вопрос. Методы построения наилучшего приближения к окружности, эллипсу, параболе, гиперболе 34](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.26in1rg)

[13 вопрос. Параметрические кубические кривые (формы Эрмита) 35](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.lnxbz9)

[14 вопрос. Параметрические кубические кривые (форма Безье, В-сплайн) 37](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.35nkun2)

[15 вопрос. Параметрические кубические поверхности (формы Эрмита, Безье, В-сплайны) 40](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1ksv4uv)

[16 вопрос. Отсечение отрезков. Алгоритм Сазерленда-Коэна. Алгоритм разбиения средней точкой 44](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.44sinio)

[17 вопрос. Алгоритм отсечения отрезка выпуклым окном. Алгоритм Кируса-Бека. Методы вычисления внутренних нормалей 45](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.2jxsxqh)

[18 вопрос. Алгоритмы отсечения отрезков в пространстве. Трехмерный алгоритм разбиения средней точкой. Трехмерный алгоритм Кируса-Бека 51](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.z337ya)

[19 вопрос. Разбиение невыпуклых многоугольников. Разбиение невыпуклых тел. 55](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3j2qqm3)

[20 вопрос. Проекционные преобразования. Классификация плоских геометрических проекций. Ортографические проекции 59](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1y810tw)

[21 вопрос. Аксонометрические проекции. Триметрия, диметрия, изометрия 63](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.4i7ojhp)

[22 вопрос. Косоугольные проекции. Проекции Кавалье и Кабине 67](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.2xcytpi)

[23 вопрос. Последовательное отсечение многоугольника. Алгоритм Сазерленда-Ходжмана 71](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1ci93xb)

[24 вопрос. Поиск пересечений многоугольников. Постановка задачи поиска пересечений многоугольников. Базовые оценки сложности 74](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3whwml4)

[25 вопрос. Алгоритм построения пересечения двух выпуклых многоугольников. Теорема о времени построения пересечения двух выпуклых многоугольников 81](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.2bn6wsx)

[26 вопрос. Алгоритм Вейлера-Азертона пересечения произвольных многоугольников 87](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.qsh70q)

[27 вопрос Геометрический поиск. Постановка задачи, оценки сложности построения решения задач регионального поиска, локализации. 89](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3as4poj)

[28 вопрос. Алгоритмы решения задач: нахождение ближайшей пары, нахождение ближайшего соседа. 93](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1pxezwc)

[29 вопрос. Региональный поиск, 2d дерево. Локализация точек. Метод полос, метод цепей. Метод триангуляции 97](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.49x2ik5)

[30 вопрос. Построение выпуклых оболочек. Постановка задачи, оценка сложности построения решения. Алгоритм Грехема. Алгоритм Джарвиса 110](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.2p2csry)

[31 вопрос. Быстрые алгоритмы построения выпуклых оболочек. Построение выпуклой оболочки простого (произвольного) многоугольника. Открытые алгоритмы 113](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.147n2zr)

[32 вопрос. Динамические алгоритмы построения выпуклой оболочки. Поддержка динамической выпуклой оболочки 115](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3o7alnk)

[33 вопрос. Диаграмма Вороного: свойства, алгоритм построения, оценки решения задач близости 121](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.23ckvvd)

[34 вопрос. Алгоритмы удаления скрытых линий и поверхностей. Алгоритм плавающего горизонта, алгоритм с использованием Z буфера 122](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.ihv636)

[35 вопрос. Удаление невидимых линий. Алгоритм Варнока, Алгоритм Вейлера-Айзертона 125](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.32hioqz)

[36 вопрос. Удаление невидимых линий, алгоритм Робертса 131](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1hmsyys)

[37 вопрос. Алгоритм, использующий список приоритетов. Алгоритм построчного сканирования 135](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.41mghml)

[38 вопрос. Конические сечения. Параметрическое задание окружности, эллипса, параболы, гиперболы. Построение наилучшего приближения с этим кривым. 140](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.2grqrue)

[39 вопрос. Алгоритм определения видимых поверхностей путем трассировки лучей. Интервальный алгоритм построчного сканирования 148](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.vx1227)

[40 вопрос. Построение реалистических изображений. Простая модель освещения для одного источника. Определение нормали к поверхности и вектора отражения 155](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3fwokq0)

[41 вопрос. Закраска методом Гуро и методом Фонга 167](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1v1yuxt)

[42 вопрос. Модель отражения от шероховатых поверхностей Торренса-Спэрроу. Прозрачность, закон Снеллиуса, расчет интенсивности света для многоугольников и криволинейных поверхностей 173](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.4f1mdlm)

[43 вопрос. Тени, методы построения, алгоритмы Вейлера-Азертона и трассировки лучей. Фактура, нанесение рисунка на поверхность, формирование фрактальной поверхности 188](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.2u6wntf)

[44 вопрос. Глобальная модель освещения с трассировкой лучей, формула вычисления значения наблюдаемой интенсивности. Алгоритм трассировки лучей 194](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.19c6y18)

[45 вопрос. Цвет в машинной графике. Основные характеристики света и человеческого восприятия цвета. Аддитивная и субтрактивная системы смешения цветов, законы Грассмана. трехмерное цветовое пространство 197](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3tbugp1)

[46 вопрос. Цветовой график МКО, цветовое пространство XYZ, стандартные источники МКО, цветовые охваты, вычисление координат цветности МКО для смеси двух цветов. 200](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.28h4qwu)

[47 вопрос. Переход от цветового пространства RGB к пространству МКО и обратно. Цветовые системы YIQ, HSV, HLS, цветовой шестигранный конус, алгоритмы преобразования цветового пространства (HSV в RGB, RGB в HSV, HLS в RGB, RGB в HLS) 203](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.nmf14n)

[48 вопрос. . Фракталы. Множество Кантора; фракталы Серпинского; кривая Коха; кривые, заполняющие плоскость; фрактальные кривые и рекурсии. 213](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.37m2jsg)

[49 вопрос. . Множества Жюлиа и Мандельброта и их компьютерное построение. Динамические процессы. Бифуркации. 215](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1mrcu09)

| **1.** [**Машинная графика. Задачи, области применения.**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.30j0zll)  **Машинная графика** – создание, хранение и обработка объектов и их изображений с помощью ЭВМ.  **Интерактивная машинная графика** - раздел, когда пользователь имеет возможность динамически управлять содержимым изображения, его формой, размерами и цветом на поверхности дисплея с помощью интерактивных устройств взаимодействия, например, клавиатуры, рычага и т.д.  Конечным продуктом компьютерной графики является изображение. Это изображение может использоваться в различных сферах, например, оно может быть техническим чертежом, иллюстрацией с изображением детали в руководстве по эксплуатации, простой диаграммой, архитектурным видом предполагаемой конструкции или проектным заданием, рекламной иллюстрацией или кадром из мультфильма.  **Компьютерная графика** — это наука, предметом изучения которой является создание, хранение и обработка моделей и их изображений с помощью ЭВМ, т.е. это раздел информатики, который занимается проблемами получения различных изображений (рисунков, чертежей, мультипликации) на компьютере.  (Слова деда) **Компьютерная графика** - область информатики, которая направлена на обработку 2-х и 3-х мерных изображений. Сегодня никто не работает без графического представления данных.  **Проблемы**:   * необходимо быстро обрабатывать большие объемы данных * должно быть красиво - качество изображения   **Задачи**:   * представление изображения в компьютерной графике; * подготовка изображения к визуализации; * создание изображения; * осуществление действий с изображением.   **Графическая информация** - модели объектов и их изображения.  **Примеры предметов, связанных с машинной графикой:**   * Скоростные печатающие устройства ударного действия, используемые для получения портретов людей или персонажей из мультфильмов, часто с наложением литер при повторной печати в той же строке. * Планшетные или барабанные графопостроители для получения рисунков, состоящих из линий: двумерных и трехмерных графиков, круговых диаграмм, гистограмм, блок-схем, архитектурных чертежей и электрических схем. * Кинокамеры и видеомагнитофоны для создания высококачественных цветных изображений существующих или воображаемых объектов в виде слайдов, кинофильмов или телевизионных фильмов, например статических диаграмм, заставок в телевизионных передачах, сцен из научно-фантастических фильмов. * Игровые (домашние) ЭВМ с черно-белым или цветным телевизором, позволяющие работать с текстами и графическими изображениями. * Визуальные игровые процессоры, в которых используются микропроцессоры. Простейшие из них представляют собой черный ящик, который может генерировать незатейливые тексты и рисунки, а его рычаги или ручки используются как интерактивные устройства.   **Примеры применения машинной графики:**   * Черчение графиков. * Картография. * Автоматизация чертежных и конструкторских работ. * Моделирование и мультипликация. * Управление процессами. * Автоматизация канцелярских работ и электронная публикация. * Искусство и реклама. | **2. Классификация применения машинной графики.**  Для классификации применений можно использовать ряд критериев.  **Первым критерием** является тип объекта и тип выводимого изображения. Диапазон возможностей включает линейные рисунки двумерных объектов, линейные рисунки трехмерных объектов, часто называемые каркасными, линейные рисунки трехмерных объектов с удалением скрытых линий, двумерные черно-белые тоновые изображения, двумерные цветные изображения и трехмерные тоновые изображения тел перечисленных объектов являются полностью абстрактными, другие — реальными; их изображения могут быть как чисто символическими (например, простой двумерный график), так и весьма реалистичными (например, натюрморт).  **Вторым критерием** служит тип интерактивности и уровень возможностей при управлении изображением, доступный пользователю. В этом случае в рассматриваемый диапазон включаются:  автономное вычерчивание на графопостроителе по готовой базе данных, полученной с помощью прикладной программы или сформированной путем съема координат с физической модели  интерактивное вычерчивание(циклами вида «подать параметры, вычертить, изменить параметры, вычертить повторно»).  задание объекта и «полет» вокруг него в реальном времени под управлением пользователя (типичная задача в летных тренажерах).  интерактивное проектирование, при котором пользователь, начав с пустого экрана, задает объект (чаще всего из готовых составных частей), а затем видоизменяет его по мере необходимости приближаясь или отдаляясь от него.  **Третьим критерием** является роль изображения, то есть в какой степени само изображение является средством для достижения цели или же самой целью. В таких применениях, как картография, изготовление рабочих чертежей и растровая живопись, само изображение является результатом, тогда как во многих видах машинного проектирование полученное изображение служит лишь для визуализации геометрических свойств рассматриваемого объекта (электронной схемы, моста, системы трубопроводов, крыла самолета или автомобильного крыла и т. д.). В этих применениях фаза получения изображения — важная, но относительно небольшая часть гораздо более обширного процесса, целью которого является формирование и последующая обработка общей базы данных с помощью комплекса прикладных программ.  **Четвертым критерием** служат логические и временные соотношения между объектами и изображениями. Например, пользователь может в данный момент времени иметь дело только с одним изображением (обычная ситуация при работе графопостроителя), с изменяющейся во времени последовательностью взаимосвязанных изображений (передается динамика движения или динамика изменения) или же, наконец, со структурной совокупностью объектов (как во многих применениях машинного проектирования, где существует иерархия общих сборочных чертежей, чертежей узлов и т.д.). |
| --- | --- |
| **3. Основные понятия машинной графики**  **Компьютерная графика** - это область информатики, направленная на обработку 2d и 3d изображений в объеме данных. Совокупность алгоритмов компьютерной графики, имеющих разную алгоритмическую сложность и аппаратно реализующихся на различных платформах и устройствах.  **Графическая система** - программно-аппаратный комплекс, ориентированный на узкую предметную область.  В зависимости от способа формирования изображений компьютерную графику принято подразделять на **растровую и векторную.**  **Пиксель** — элемент рисунка(точка). Изображение представляет собой множество цветных точек, расположенных на сетке. Сама сетка называется растровой картой (bitmap).  Если в **растровой(точечной)** графике базовым элементом изображения является **точка**, то в векторной графике — **вектор**(направленный отрезок). Вектор описывается математически как единый, и потому объем данных для отображения объекта средствами векторной графики существенно меньше, чем в растровой графике.  **Графические примитивы** - простейшие фигуры и структуры, с помощью которых можно описать графические данные(синтезировать изображения для заданной предметной области)  **Точка** - этот объект на плоскости представляется двумя числами (x, y), указывающими его положение относительно начала координат. Атрибуты: координаты, цвет, видимость/невидимость, мерцание, яркость  **Прямая линия.** Ей соответствует уравнение y=kx+ b. Указав параметры k и b, всегда можно отобразить бесконечную прямую линию в известной системе координат. Атрибуты: толщина, цвет, начертание (сплошная, пунктирная).  **Отрезок прямой**. Он отличается тем, что требует для описания ещё двух параметров – например, координат х1 и х2 начала и конца отрезка. Атрибуты: длина, толщина, цвет, яркость, мерцание.  **Ломаная линия** - множество отрезков. Атрибуты: толщина, яркость, мерцание, концы одного отрезка - начало другого(скорее св-во).  **Многоугольник** - замкнутая ломаная. Атрибуты: тип закраски, + атр.ломаной, не имеет самопересечений, любое направление обхода(тож св-ва).  **Кривые второго порядка**: параболы, гиперболы, эллипсы, окружности, то есть все линии, уравнения которых содержат степени не выше второй.  Изображение выглядит как Шрифт, белый, каллиграфия, текст  Автоматически созданное описание**Кривая третьего порядка.** Отличие этих кривых от кривых второго порядка состоит в возможном наличии точки перегиба.    К графическим примитивам относится **текст**: шрифт, кол-во символов, направление шрифта. | **4. Графические пакеты. Взаимодействие графического пакета с ОС и прикладными программами. Блок-схема графического пакета и графической системы**  **Графические пакеты** - прикладные программы, предназначенные для создания и обработки цифровых изображений. Они подразделяются на::   | **Пакеты 2-мерной графики** | **Пакеты 3-мерной графики** | | --- | --- | | Ориентированы на работу с 2-мерными изображениями. Изображения рассматриваются как набор полупрозрачных слоев. У каждого слоя есть степень прозрачности. Пример: обычный редактор изображений Adobe Photoshop | Ориентированы на моделирование 3-мерных объектов и сцен. Есть пакеты, ориентированные на САПР(системы автоматизированного проектирования) - например AutoCAD. А также для рендеринга сцен - например Blender3D |   **Пакеты 2-мерной графики** делятся на пакеты растровой и пакеты векторной графики.   | **Пакеты растровой графики** | **Пакеты векторной графики** | | --- | --- | | Рассматривают изображение как сетку пикселей. Можно редактировать отдельные пиксели или их группы. Изменение размеров изображения ухудшает качество.  Пример: Adobe Photoshop | Изображение состоит из геометрических объектов (примитивов). Можно редактировать формы, масштабировать, вращать. Изменение размеров НЕ изменяет качество.  Пример: Adobe Illustrator |   **Графическая система** - программно-аппаратный комплекс, ориентированный на узкую предметную область.  **Блок-схема:**    **Описание**: на схеме: 1) Прикладная структура данных - описывает объекты, которые должны отобразиться на экране. 2) Прикладная программа - обрабатывает вх. данные и управляет графическим выводом. В ее задачи входит задание атрибутов(цвет, размер, шрифты), генерация граф. примитивов(точки, линии), задание выводной операции - какие данные передавать граф. пакету.  3) Графический пакет - промежуточный слой между прикл. программой и аппаратным обеспечением. Содержит дисплейную программу (код, который отвечает за отображение объектов на экране) и дисплейный буфер (временно хранит граф. данные перед отправкой на диспл. процессор).  4) Дисплейный процессор - обрабатывает данные из диспл. буфера, управляет выводом на дисплеи (устройства вывода), принимает сигналы от устройств ввода(клавиатура, планшет). 5) ОС - выделяет ресурсы для граф. пакета и буфера, управляет выполнением прикладных программ, передает сигналы от устройств ввода в прикладную программу.  **Взаимодействие**: 1) Прикл. программы отправляет запрос в граф. пакет. 2) Граф. пакет преобразует команды в универсальный формат и передает их через системные вызовы в ОС. 3) ОС управляет памятью, выделяет буферы и передает команды диспл. процессору через драйвер видеокарты.  4) Дисплейный процессор обрабатывает данные и обновляет содержимое дисплея. |

| **5. Алгоритмы растровой машинной графики. Развертка отрезка (алгоритм Брезенхема).**  **Постановка задачи:**  ● **Проблема**: отрезок может проходить между точками растровой сетки и необходимо вычислить ближайшие координаты точки сетки.  ● **Дано:** начальная и конечная точки отрезка, двумерная растровая сетка  ● **Найти:** вычислить координаты пэлов, лежащих вблизи отрезков на двумерной растровой сетке. При решении этой задачи будет предполагаться, что начальная и конечная точки отрезка имеют целочисленные координаты.  **Простейший пошаговый алгоритм, идея:** Через начальную и конечную точку находим тангенс угла наклона прямой, обозначим его как m = (y\_2-y\_1)/(x\_2-x\_1). Теперь из начальной точки мы можем двигаться вправо, наращивая значения x, смещение по x обозначим Δx = 1, по y смещение будет Δy =Δx\* m=m. Теперь заметим, что каждое последующее значение x\_(i+1) и y\_(i+1) вычисляется из предыдущих значений x\_i и y\_i <= x\_(i+1) = x\_i + 1, y\_(i+1) = y\_i+m. Если y\_i’ - округлённое до ближайшего целого y\_i, то на каждом i шаге требуется закрашивать пэл с координатами (x\_i,y\_i’). В случае, если m > 1, то x и y следует поменять местами.  Проблема алгоритма в том, что он использует округление до ближайшего целого, а этого дорогостоящая операция, при это работа идёт с вещественными переменными или двоичными дробями. Поэтому в этом плане более привлекателен алгоритм Брезенхэма.  **Алгоритм Брезенхема:** Алгоритм Брезенхэма используется для отрисовки прямых на растровом экране. Он основан на целочисленной арифметике и не требует операций деления или умножения, что делает его эффективным. Для простоты будем считать, что тангенс угла наклона находится в диапазоне 0-1. В алгоритме используется управляющая переменная d\_i, которая на каждом шаге пропорциональная разности между s и t(рисунок ниже). На рисунке уже установлено, что пэл P\_(i-1) найден как ближайший, теперь требуется определить какой из пэлов T\_i или S\_i будет ближе. Если s-y < 0 выбирается S\_i, иначе – T\_i.    Изображаемый отрезок проводится из точки (x1, y1), в точку (x2, y2). Пусть первая точка находится ближе к началу координат, тогда перенесём обе точки при помощи T(-x1, -y1) так, чтобы начальной точкой отрезка стала точка(0, 0), а конечной (dx, dy), где dx = x2-x1 и dy = y2-y1. Уравнение прямой примет вид y = (dy/dx)x. Обозначим координаты Pi-1 после переноса (r, q), как показано на рисунке. Из рисунка следует, что s = (dy/dx)*(r+1) -q, t = q+1 - (dy/dx)*(r+1), поэтому s-t = 2(dy/dx(r+1) - 2q - 1. Если s-t < 0, то выбираем Si. Немного преобразуем последнее равенство => dx(s-t) = 2(r\*dy-q\*dx)+2dy-dx. Т.к. величина dx положительна, то правую часть равенства можно использовать при выборе S\_i. Правая часть неравенства и будет управляющей переменной d\_i. Теперь, т.к. q=x\_(i-1) и q=y\_(i-1), то d\_i = 2x\_(i-1)\*dy-2y\_i-1dx+2dy-dx. Теперь получим d\_(i+1) – d\_i = 2dy(x\_i -x\_(i-1)) - 2dx(y\_i-y\_(i-1)). И т.к. x\_i-x\_(i-1) = 1, то d\_(i+1) = d\_i+2dy-2dx(y\_i-y\_(i-1)). Теперь, если же d\_i >= 0, выбирается T\_i, тогда y\_i=y\_(i-1)+1 и d\_(i+1) = d\_i+2(dy-dx). Иначе выбираем S\_i, при этом y\_i = y\_i(-1) и d\_(i+1)=d\_i+2dy. Для вычисления по этим формулам требуются минимальные арифметические возможности: сложение, вычитание и сдвиг влево(для умножения на 2). В случае, если тангенс лежит вне диапазона 0-1, требуется поменять местами x и y. | **6. Алгоритмы растровой машинной графики. Развертка окружности(алгоритм Брезенхема).**  ● **Проблема:** Преобразование окружности в растровую форму. Существуют неэффективные методы  ● **Дано:** 1) Окружность с центром (x0,y0) и радиусом r  ● **Найти:** Координаты точек окружности в растровом представлении.  **Алгоритмическая сложность:** Время выполнения зависит от размера окружности или эллипса. Обычно она оценивается как O(N), где N - количество точек на окружности или эллипсе.  **Основная идея**: использование алгоритма Брезенхема для построения растровой формы окружности. Данный алгоритм основан на выборе ближайшей к истинной фигуре точки на каждом шаге и учете симметрии фигуры.  Развертка окружности  Существует несколько очень простых, но не эффективных способов преобразования окружностей в растровую форму**. Например**, рассмотрим для простоты окружность с центром в начале координат. Ее уравнение записывается как *x^*2 *+ y^*2 = *R^*2. Решая это уравнение относительно *y,* получим Чтобы изобразить четвертую часть окружности, будем изменять *x* с единичным шагом от 0 до *R* и на каждом шаге вычислять *y*.  **Вторым простым** методом растровой развертки окружности является использование вычислений *x* и *y* по формулам *x* = *R* cos α, *y* = *R* sin α при пошаговом изменении угла α от 0° до 90°.Для упрощения алгоритма растровой развёртки стандартной окружности можно воспользоваться её симметрией относительно координатных осей и прямых *y* = ± *x*; в случае, когда центр окружности не совпадает с началом координат, эти прямые необходимо сдвинуть параллельно так, чтобы они прошли через центр окружности. Тем самым достаточно построить растровое представление для 1/8 части окружности, а все оставшиеся точки получить симметрией  Рассмотрим участок окружности из второго октанта x ϵ [0, R/ ⎷2]. Далее опишем **алгоритм Брезенхейма** для этого участка окружности.  На каждом шаге алгоритм выбирает точку *P\_i* (*x\_i, y\_i*), которая является ближайшей к истинной окружности. Идея алгоритма заключается в выборе ближайшей точки при помощи управляющих переменных, значения которых можно вычислить в пошаговом режиме с использованием небольшого числа сложений, вычитаний и сдвигов.  Рассмотрим небольшой участок сетки пикселов, а также возможные способы (от A до E) прохождения истинной окружности через сетку.  Предположим, что точка *P\_(i-*1) была выбрана как ближайшая к окружности при *x* = *x\_(i-*1). Теперь найдем, какая из точек (*S\_i* или *T\_i*) расположена ближе к окружности при *x* = *x\_(i-*1) + 1.  функция отклонения или решающая функция D(P\_i)=(x\_i)^2+(y\_i)^2−r^2  *D*(*S\_i*)=(*x\_(i*−1)+1)^2+(*y\_(i*−1))^2−*R^*2  D(T\_i​)=[(x\_(i−1​)+1)^2+(y\_(i−1)​−1)^2]−R^2.  Если ∣D(S\_i)∣≥∣D(T\_i)∣ то T\_i​ ближе к реальной окружности, иначе выбирается S\_i​.  Введем d\_i=∣D(S\_i)∣−∣D(T\_i)∣.  T\_i​ будет выбираться при d\_i≥0, в противном случае будет устанавливаться S\_i​.  Опуская алгебраические преобразования, запишем d\_i​ и d\_(i+1)​ для разных вариантов выбора точки S\_i​ или T\_i​:  D1=3−2R. (если посчитаем |S\_(1)|-|T\_(1)|)   * Если выбирается S\_i (когда d\_i<0), то   d\_(i+1)=d\_i+4x\_i+6.   * Если выбирается T\_i (когда d\_i≥0), то   d\_(i+1)=d\_i+4(x\_(i−1)−y\_(i−1))+10. |
| --- | --- |
| **7.**[**Развертка многоугольников (алгоритм с упорядоченным списком ребер)**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1t3h5sf)  Дано:  - дискретная растровая сетка  - V(v1, v2, …, vn) – упорядоченное множество вершин многоугольника  - цвет закраски многоугольника  Надо: изменить цвет точек растровой сетки, находящихся внутри V, на заданный цвет краски  Основные шаги алгоритма:  1. Для каждой сканирующей строки y € [y\_min, y\_max] определяются точки пересечения со сторонами полигона: {x1, …, xk}, где k-четное число. Каждое множество точек пересечения сортируется по возрастанию  2. Вычисляются промежутки строк, соответствующих внутренним частям многоугольника: [x\_(2i-1), x\_(2i)]- все нечетные промежутки  3. Промежутки закрашиваются требуемым цветом  В общем случае при пересечении сканирующей строки с ребрами получается четное число точек, но есть 3 исключительных ситуации:  1. Отдельно стоящая вершина-> учитывается 2 раз  2. Горизонтальное ребро -> игнорируется при обработке  3. Двойное пересечение  **Алгоритм:**  Используем список активных ребер (САР), в котором будет храниться информация обо всех ребрах многоугольника, пересекаемых текущей строкой.  При переходе к новой строке не требуется его полностью переформировывать, достаточно лишь удалить из него «закончившиеся ребра» (чей нижний конец оказался выше нового значения y) и добавить вновь появившиеся.  1. Упорядочить ребра по Y  2. Найти y\_min  3. Определить текущие ребра в САР (для текущей строки Y\_current)  4. Найти пересечения Y\_current с ребрами из САР  5. Построить закрашиваемые интервалы (отсортировать по x, взять нечетные интервалы)  6. Закрасить интервалы  7. Y\_current = Y\_current + 1  Повторять пока Y\_current <= Y\_max, перейти к шагу 3.  **В более частном виде так:**  1. Этап подготовки. Для каждого ребра создаётся структура данных:  ◦ ﻿﻿уо - начальная точка ребра по у координате  ◦ ﻿﻿хо - х координата точки пересечения ребра с сканирующей строкой. В начале задаётся как начальная точка ребра по х координате(рост y-координаты происходит сверху вниз)  ◦ ﻿﻿dx-смещение по х при движении вдоль ребра, соответствующее увеличению у-координаты на 1,  ◦ ﻿﻿dy-число пересекаемых ребром строк (у-составляющая длины).  Каждая структура заносится в соответствующую у группу(в зависимости от первого параметра у0).  2. Этап отрисовки. Для каждого значения у € [Ymin, утах ](для каждой сканирующей строки) неоходимо выполнить следующее:  • Проверить у-группу на наличие рёбер. Если таковые есть, то они заносятся в САР  ◦ ﻿﻿Отсортировать х координаты точек пересечения рёбер хО САР с текущей строкой: отсортировать список активных рёбер по возрастанию х координаты пересечения ребра и текущей сканирующей строки.  ◦ ﻿﻿Выделить пары пересечений из отсортированного по х списка. Как же рассматривать всякие исключительные ситуации? Моменты исключительных ситуаций возникают в начале и конце рёбер, эти ситуации легко найти по добавлению и удалению рёбер из САР и сделать для них свою особую реализацию обработки.  ◦ ﻿﻿Закрасить соответствующие интервалы.  ◦ ﻿﻿Для всех рёбер в САР уменьшить dy на 1: dy=dy-1  ◦ ﻿﻿Проверить dy для всех рёбер из САР. Если dy < 0, то исключить его из САР.  ◦ ﻿﻿Вычислить новые координаты ×0 точек пересечения: x0+ = dx  ﻿﻿Перейти к следующей сканирующей строке: у = у + 1 | **8. Алгоритм заполнения гранично-заданных областей. Построчный алгоритм, рекурсивный алгоритм**  Дано: дискретная растровая сетка, С1 - цвет границы области, С2 - цвет закраски внутри области.  Надо: закрасить точки, ограниченные цветом С1 в цвет С2  **Рекурсивный алгоритм(сложность O(N), N- кол-во пикселей)**  Есть 4 и 8-ми связные области (фон Неймана и Мура). Можно использовать любую в данном алгоритме (изменится только количество запусков рекурсии для одной клетки.  Опишу рекурсивную функцию func(x, y, c1,c2):  Алгоритм начинается в какой-то точке внутри области. В этой точке проверяется цвет.  Если color(x,y) = C1 |color(x,y) = C2 -> завершение функции  иначе:     * перекрашиваем точку p в цвет C2 * запускаем функцию func рекурсивно:   func(x+1,y, c1,c2), func(x-1,y, c1,c2), func(x, y+1, c1,c2), func(x ,y-1, c1,c2)  если окрестность 8 точек, запускаемся еще в 4 точках.  **Построчный алгоритм**   * P(x,y) помещается в стек * Затравочный пиксел на интервале извлекается из стека * Интервал с затравочным пикселом заполняется цветом влево и вправо (пока не встретим C1) * В переменной Xl и Xr храним крайний правый и левый пиксел * В диапазоне Xl <=X<=Xr проверяем строку сверху и снизу над текущей строкой. Определяем, есть ли еще незакрашенные пиксели в этих строках. Если есть, то крайний правый пиксел помещаем в стек и потом используем как затравочный. * Работа окончена при опустошении стека.   В таком виде алгоритм обрабатывает 4связные области.  Каждый пиксель обрабатывается **не более 3 раз** (при заливке и проверке сверху/снизу). Сложность O(N)  Изображение выглядит как прямоугольный, диаграмма, Прямоугольник, План  Автоматически созданное описание |

| 9 **Геометрические преобразования. Однородные координаты, композиция преобразований (масштабирование, поворот, перенос на плоскости и в пространстве)**  Однородные координаты - система координат (СК), используемая для однородного представления n-мерного объекта (то есть его представление в (n+1)-мерном пространстве, полученном добавлением 1 коорд. - скалярного множителя). (Прим. точка P(x,y) в однородной СК становится P(w1,w2,w3))  (-одновременно не равные нулю): , , где - скалярный множ-ль  В общем виде преобразования:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Перенос  В *афинной СК* на плоскости:  В однородных:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Масштабирование  В *афинной СК* на плоскости:  В однородных:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Поворот  В афинной СК на плоскости:  В однородных: | 10 **Конические сечения. Параметрическое задание окружности, эллипса, параболы, гиперболы. Построение наилучшего приближения с этим кривым**  Общий вид параметрических уравнений: x=x(t); y=y(t). P(t) = [x(t), y(t)]  Проблема: если строить окружность стандартно, используя формулу , имеем проблему неравномерности построения. В первой четверти при равных приращениях отрезки окружности не будут равны.  Параметрическая формула единичной окружности:  Будем использовать приращения углов при построении:    т.к.можно записать рекурсивную формулу:    Параметрическая формула для эллипса:(каноническая в координатах х и у: . Таким образом  Учитывая, формулы для xi и yi:    Параметрическая формула для параболы:  Тогда Реккурентно:    Формула для гиперболы: Параметрически: Используем    (в конспекте тут +в знаменателе, не понимаю поч) |
| --- | --- |
| **11. Представление пространственных форм. Полигональные сетки.**  **Полигональная сетка** - совокупность ребер, вершин и многоугольников. Вершины соединяются ребрами, а многоугольники рассматриваются как последовательности ребер или вершин. Чем более явно выражены зависимости между многоугольниками, вершинами и ребрами, тем быстрее работают операции над ними и тем более памяти требует соответствующее представление. Наружную форму большинства зданий можно легко и естественно описать с помощью полигональной сетки (так же, как мебель и комнаты).  Полигональные сетки применяются также для представления объектов, ограниченных криволинейными поверхностями. Недостаток этого метода - его приблизительность.  **Три способа описания полигональных сеток.**   * **Явное задание многоугольников**   Каждый многоугольник можно представить в виде списка координат его вершин. page30image62715296  P = ((x1, y1, z1), (x2, y2, z2), ... , (xn, yn, zn))  Вершины запоминаются в том порядке, в котором они встречаются при обходе вокруг многоугольника. При этом все последовательные вершины многоугольника (а также первая и последняя) соединяются ребрами.  Для каждого отдельного многоугольника данный способ записи является эффективным, однако для полигональной сетки дает потери памяти вследствие дублирования информации о координатах общих вершин. Полигональная сетка изображается путем вычерчивания ребер каждого многоугольника, однако это приводит к тому, что общие ребра рисуются дважды – по одному разу для каждого из многоугольников.   * **Задание многоугольников с помощью указателей в список вершин**page30image62693088   При использовании этого представления каждый узел полигональной сетки запоминается лишь один раз в списке вершин V =((x1, y1, z1), ... , (xn, yn, zn)). Многоугольник определяется списком указателей (или индексов) в списке вершин. Многоугольник, составленный из вершин 3, 5, 7 и 10 этого списка, представляется как Р = (3, 5, 7, 10).  Такое представление имеет ряд преимуществ по сравнению с явным заданием многоугольников. Поскольку каждая вершина многоугольника запоминается только один раз, удается сэкономить значительный объем памяти. Кроме того, координаты вершины можно легко изменять. Однако все еще не просто отыскивать многоугольники с общими ребрами. Ребра при изображении всей полигональной фигуры по-прежнему рисуются дважды. Эти две проблемы можно решить, если описывать ребра в явном виде.   * **Явное задание ребер**page31image59498000   В этом представлении имеется список вершин V, однако будем рассматривать теперь многоугольник как совокупность указателей на элементы списка ребер, в котором ребра встречаются лишь один раз. Каждое ребро в списке ребер указывает на две вершины в списке вершин, определяющие это ребро, а также на один или два многоугольника, которым это ребро принадлежит. Таким образом, мы описываем многоугольник как P =(E1, ..., E2), а ребро — как Е = (V1, V2, P1, P2). Если ребро принадлежит только одному многоугольнику, то либо P1, либо P2 – пусто. На рис. 13.6 приведен пример такого представления. При явном задании ребер полигональная сетка изображается путем вычерчивания не всех многоугольников, а всех ребер. В результате удается избежать многократного рисования общих ребер. Отдельные многоугольники при этом также изображаются довольно просто.  **Описание полигональной сетки является непротиворечивым, когда:**   * Все многоугольники замкнуты * Все ребра используются по крайней мере один раз, но не более заданного числа * На каждый узел есть ссылка по крайней мере от двух ребер   **Особые требования**   * Полигональная сетка должна быть связной (в любой узел можно попасть из любого другого узла, двигаясь вдоль ребер) * Каждый полигон должен быть плоским (узлы, задаваемые ребрами, связаны бинарным отношением, которое можно представить в виде плоского графа) * Отсутствие “дыр" (существует только одна граница — связная последовательность ребер, каждое из которых принадлежит только одному многоугольнику).   Непротиворечивость имеет важное значение, поскольку сетки часто генерируются в интерактивном режиме, в том числе иногда путем оцифровывания чертежей, поэтому ошибки при этом неизбежны.  Схему с явным заданием ребер наиболее просто проверять на непротиворечивость, поскольку в ней содержится больше всего информации.  С помощью полигональной сетки можно описать многогранник (объем, ограниченный фигурой, составленной из многоугольников). Примерами многогранников являются пирамида и куб. В случае многогранника каждое ребро полигональной сетки должно принадлежать двум соседним многоугольникам. Ребру, которое принадлежит только одному многоугольнику, соответствует дыра в многограннике (чтобы понять это, рассмотрите различие между закрытой и незакрытой коробками для обуви). Если же соседними являются более чем два многоугольника, в этом случае многогранник имеет дополнительные внутренние или внешние многоугольники. | **12. Методы построения наилучшего приближения к окружности, эллипсу, параболе, гиперболе**  В 10 вопросе расписаны определения и подробный вывод.  **Постановка задачи**  **Дано**: уравнение кривой, n - количество точек, которые хотим выдать, [xmin, xmax] для гиперболы/параболы  **Надо**: вычислить координаты (xi,yi), i=1..n с минимальной погрешностью, равномерно расставленные по кривой. План построения наилучшего приближения   1. Строим параметрическое представление x(t), y(t) 2. Дискретизуем параметр dt → Δt (Точки должны быть равномерно распределены по длине кривой, а не по параметру t. Это требует учета скорости изменения длины кривой относительно параметра t.) 3. Строим рекуррентную формулу (координаты следующих точек на основе старых) |

| **13. Параметрические кубические кривые (формы Эрмита)**  **Проблема:** построить непрерывную пространственную кривую, которая в графике представляется в виде сегментов (каждый сегмент представляется кубическими многочленами третьей степени).  **Дано**: известны концевые точки и касательные векторы к кривой в этих точках.  **Найти**: f - функции и f’ - производные для функции. (то есть коэффициенты этой функции).  **Погнали:**  Параметрическая кубическая кривая задаётся тремя уравнениями:  x(t) = a\_x t^3 + b\_x t^2 + c\_x t + d\_x  y(t) = a\_y t^3 + b\_y t^2 + c\_y t + d\_y  z(t) = a\_z t^3 + b\_z t^2 + c\_z t + d\_z  Параметр t лежит в пределах от 0 до 1.  **Для куска кривой должны быть известны:**  **Координаты** начальной и конечной точек:P\_1(P\_{1x}, P\_{1y}), P\_4(P\_{4x}, P\_{4y})  **Производные** (по каждой из координат) в начальной и конечной точках - рассматриваются вычисления относительно х:x'(0) = R\_{1x}, x'(1) = R\_{4x}    Найдем выражение для кубической кривой в форме Эрмита, если известны концевые точки и касательные векторы к кривой в этих точках. Зададим точки ( P\_1 ) и ( P\_4) и касательные векторы ( R\_1 ) и ( R\_4 ). Требуется найти коэффициенты ( a\_x, b\_x, c\_x, d\_x ) из выражения  x(t) = a\_x t^3 + b\_x t^2 + c\_x t + d\_x  удовлетворяющего условиям  P\_{1x} = x(0), P\_{4x} = x(1),  R\_{1x} = x'(0), R\_{4x} = x'(1).  Будем описывать функцию ( x(t) ) в матричном виде(1 картинка)  Соединим информацию о граничных условиях с новой формой представления  полинома(2 картинка) | **14. Параметрические кубические кривые (форма Безье, B-сплайн)**   1. **Форма Безье**   Отличается от формы Эрмита заданием касательных векторов в виде конечных точек.  **Дано:** контрольные точки P1, P2, P3, P4 (мб любой размерности)  **Требуется**: построить кривую в форме Безье  **Как строить:**  Касательные векторы **R1, R4** задаются отрезками P1P2 и P3P4.    Переход от геометрического вектора Безье G\_b к геом. вектору Эрмита G\_h определяется  как:  , где M\_hb - матрица перехода от геом. вектора Безье к геом. вектору Эрмита.  Выражаем саму кривую через функцию так:    , где T = [t^3 t^2 t 1] - вектор-строка, M\_h - матрица базисных функций Эрмита (фиксированная), G\_h - геом. вектор Эрмита, G-b - геом. вектор Безье, M\_b - матрица Безье (матрица коэффициентов базисных функций кривой Безье).  Кривая Безье 3-го порядка:    **Свойства:**  1) Кривая всегда располагается внутри фигуры , образованной линиями, соед. контрольные точки  2) Симметрична (т.е. смена местами начальной и конечной точек не влияет на форму кривой)  3) Степень кривой всегда на 1 меньше числа контрольных точек. Например, при 3 контрольных точках форма кривой - парабола (это кривая 2-ого порядка), в нашем билете кубическая кривая.  4) 2 кривые Безье, имеющие общую точку - непрерывны в этой точке.     1. **B-сплайн**   B-сплайн описывается следующей формулой:  , где T = [t^3 t^2 t 1] - вектор-строка, M\_s - матрица базисных коэффициентов B-сплайна, G\_s - вектор  координат x четырех опорных точек.  Матрица M\_s для кубического B-сплайна равна:    Форма B-сплайнов является более гладкой, чем другие формы представления, т.к. первая и вторая производные непрерывны в конечных точках, в отличие от форм Эрмита и Безье, у которых в конечных точках непрерывны лишь первые производные.  В общем случае может проходить через любые управляющие точки. |
| --- | --- |
| **15.** [**Параметрические кубические поверхности (формы Эрмита, Безье, В-сплайны)**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.1ksv4uv)  Перейдем от кубических кривых к бикубическим поверхностям, задаваемым кубическими уравнениями от двух переменных *s* и *t*. Изменяя оба параметра от 0 до 1, можно определить все точки на куске поверхности. Если одному параметру присвоить постоянное значение, а другой – изменять в диапазоне 0-1, то в результате получим кубическую кривую. Так же как и в случае кривых, мы будем рассматривать только уравнение для x:    **1. Форма Эрмита**  Рассмотрим уравнение кубической кривой и заменим t на s:  Сделаем геометрическую матрицу Эрмита функцией от t:  P1 – координаты 1 точки,  P4 – координаты последней.  R1 – производная в т. t = 0,  R4 – производная в т. t= 1.      Подставим в исходное уравнение:  Коэффициенты:  **2. Форма Безье**  Уравнения для кусков Безье выводятся так же, как и для бикубических кусков Эрмита. В результате:  P(t) = TMbGb, где Gb = [P1 P2 P3 P4]T (столбец) – отрезки касательных (P1 и P4 – начальный и конечный).    Геометрическая матрица P состоит из 16 управляющих точек (рис).  Поверхности Безье используются часто при интерактивном проектировании по тем же причинам, что и кривые Безье: управляющие точки позволяют легко изменять форму куска поверхности.  Поверхности Безье, так же как и кривые Безье, обладают свойством выпуклой оболочки.  **3. Форма B-сплайнов**  Куски в форме B-сплайнов представляются в виде:  Здесь, как и для кривых в форме B-сплайнов, достигается C(2)-непрерывность.  P(t) = TMsGs, где Gs = [Pi-1 Pi Pi+1 Pi+2]T (столбец) – для аппроксимации от т. Pi до Pi+1.  **Сравнение форм Эрмита, Безье и B-сплайнов:**   * Форма Эрмита пригодна для аппроксимации уже имеющихся поверхностей. Если необходимо добиться соответствия касательных и концевых точек. * Форма Безье и B-сплайн более пригодны для работы в интерактивном режиме, т.к. требуются только точки. * При этом Безье выгоднее когда надо точно установить концевые точки. B-сплайн пригоден, когда не так критичны конца, но надо сделать кривую гладкой. | **16. Отсечение отрезков. Алгоритм Сазерленда-Коэна. Алгоритм разбиения средней точкой**  Основное применение алгоритмов отсечения - это отбор той информации, которая необходима для визуализации конкретной сцены или вида, как части более обширной обстановки. Но кроме этого отсечение применяется в алгоритмах удаления невидимых линий и поверхностей, при построении теней..  Алгоритм отсечения отрезков средней точкой основан на алгоритме Сазерленда-Коэна, поэтому мы должны сначала ввести последний. Суть алгоритма заключается в том, что концам отрезка присваивается четырёхбитный код: 𝑏0, 𝑏1, 𝑏2,𝑏3. Этот четырёхбитный код содержит информацию о положении точки относительно области вывода. На практике возможны 9 комбинаций: ну тут короче, 0000 есть на экране, остальное выходит за экран При отсечения отрезков, сначала получите кодирование code1 и code2 конечных точек p1 и p2, а затем выполнить двоичную операцию «или» и операцию «и»:   1. Если 𝑐𝑜𝑑𝑒1 | 𝑐𝑜𝑑𝑒2 = 0, значит отрезок целиком лежит внутри окна и должен быть отрисован целиком; 2. Если 𝑐𝑜𝑑𝑒1 & 𝑐𝑜𝑑𝑒2 ≠ 0, отрезок лежит за пределами окна и не будет отрисован; 3. Во всех остальных случаях в окне лежит только часть отрезка, и это значит, что есть необходимость в отсечении.   **3! Шаг 1: Выбрать точку с ненулевым кодом — она вне окна.**  **Шаг 2: Найти, с какой стороной окна отрезок пересекается:**   * Если b3 = 1 → точка **над** окном → пересечение с **верхней** границей y = y\_max * Если b2 = 1 → точка **ниже** окна → пересечение с **нижней** границей y = y\_min * Если b1 = 1 → точка **справа** → пересечение с **правой** границей x = x\_max * Если b0 = 1 → точка **слева** → пересечение с **левой** границей x = x\_min  **Шаг 3: Вычислить координаты точки пересечения** Пример: если мы пересекаем верхнюю границу (y = y\_max), то ищем x:    шаг 4:Заменить старую точку (которая была вне окна) на найденную точку пересечения, и пересчитать её код. Заменить старую точку (которая была вне окна) на найденную точку пересечения, и пересчитать её код. **Метод средней точки** (бинарный поиск): **делит отрезок пополам** и **рекурсивно** проверяет, где его части находятся по отношению к окну. **Недостатки:**  ❌ Может потребовать **много итераций**, особенно если отрезок почти параллелен границе окна  ❌ Медленнее в некоторых случаях, чем прямой расчёт координаты пересечения |

| [**17. Алгоритм отсечения отрезка выпуклым окном. Алгоритм Кируса-Бека. Методы вычисления внутренних нормалей**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.2jxsxqh)  Дано: P\_1P\_2-отрезок, V(h\_1, h\_2, …, h\_k) – многоугольник(он же “окно”), h\_i = (v\_1, v\_2, …, v\_(r\_i), v\_1) – i-я грань многогранника  Надо: найти P’\_1, P’\_2 – точки пересечения отрезка и окна  Ограничения: V – выпуклый, P\_1P\_2 пересекает V.  Параметрическое представление отрезка от P\_1 до P\_2 имеет вид: P(t) = P\_1 + (P\_2-P\_1)t, t € [0; 1], где t – вещественный параметр. Обозначим отсекающую область: R — произвольный выпуклый многогранник. Внутренняя нормаль n в произвольной точке а, лежащей на границе R, удовлетворяет условию n • (b - а) ≥ 0, где b — любая другая точка на границе R. Это условие иллюстрирует рисунок 1, где обозначены внутренняя n\_B и внешняя нормаль n\_H, указанное условие соответствует Q € [-pi/2, pi/2], угол Q — угол между внутренней нормалью n\_B и вектором (b — а).    Если f — граничная точка выпуклой области R, a n — внутренняя нормаль к одной из ограничивающих эту область плоскостей, то для любой конкретной величины t, т. е. для любой точки отрезка P\_1P\_2:  ◦ ﻿﻿из условия n• (P(t) - f) < 0 следует, что вектор P(t) — f направлен вовне области R.  ◦ ﻿﻿из условия n • (P(t) - f) = 0 следует, что P(t) — f принадлежит плоскости, которая проходит через f и перпендикулярна нормали.  ◦ из условия n • (P(t) - f) > О следует, что вектор P(t) — f направлен внутрь R, как показано на рис. 2.      Из всех этих 3 условий следует, что бесконечная прямая пересекает замкнутую выпуклую область ровно в двух точках.  Пусть две эти точки не принадлежат одной граничной плоскости или ребру. Тогда уравнение n\*(P(t) – f) = 0 имеет только одно решение. Если точка f лежит на той граничной плоскости или на том ребре, для которых n является внутренней нормалью, то точка на отрезке P(t), которая удовлетворяет последнему уравнению, будет точкой пересечения этого отрезка с указанной граничной плоскостью.  Подставим определение для P(t\_i) в уравнение n\_i\*(P(t\_i)-f\_i) = 0, тогда получим: n\_i \* (P\_1 + (P\_2 -–P\_1)\*t\_i – f\_i) = 0⇔ n\_i\*(P\_1 – f\_i) + n\_i \* (P\_2-P\_1)\*t\_i – условие пересечения отрезка с i-й границей области. ((P\_1 – f\_i) = w\_i, (P\_2-P\_1) = D != 0). Решая относительно t\_i, получим t\_i = -(n\_i \* w\_i)/(n\_i \* D),i = 1, 2, …, k  Здесь n\_i \* D может быть равно нулю, только при P\_2 = P\_1, либо если отсекаемый отрезок параллелен грани окна.  Если:  1. n\_i \* w\_i < 0, то точка вне окна  2. n\_i \* w\_i = 0, то она на грани окна  3. n\_i \* w\_i > 0, то она внутри окна  Если значение t\_i лежит за пределами интервала t\_i € [0; 1], то можно его проигнорировать. Хотя известно, что отрезок может пересечь выпуклое окно не более чем в двух точках, т. е. при двух значениях t\_i, уравнения могут дать большее число решений в интервале t\_i € [0; 1]. Эти решения следует разбить на две группы: нижнюю и верхнюю, в зависимости от того, к началу или к концу отрезка будет ближе соответствующая точка. Нужно найти наибольшую из нижних и наименьшую из верхних точек. Если n\_i • D > 0, то найденное значение t\_i рассматривается в качестве возможного нижнего предела. Если n\_i • D < 0, то значение t\_i рассматривается в качестве возможного верхнего предела.    Сложность алгоритма Кируса-Бека: O(k), где k – количество граней. Нахождение внутренней нормали к стороне многоугольника  Для вычисления внутренней нормали к любой стороне многоугольника можно использовать следующий метод:  Шаги алгоритма:   1. Представление стороны как вектора:    * Пусть сторона многоугольника представлена вектором:*V\_e*​=(*V\_(e\_x)*​​,*V\_(e\_y)*​​). 2. Вычисление перпендикулярной нормали:    * Нормаль к вектору *V\_e*​ может быть найдена как: *n*=(−*V\_(e\_y)*​​,*V\_(e\_x)*​​).    * Это вектор, перпендикулярный *V\_e*​. 3. Определение внутренней нормали:    * Чтобы определить, является ли найденная нормаль внутренней или внешней, нужно проверить её направление относительно соседних сторон многоугольника.    * Выберем соседнюю сторону *V’\_e*​ и вычислим скалярное произведение: *n*⋅*V’\_e*​.    * Если скалярное произведение положительно , то *n* — внутренняя нормаль.    * Если скалярное произведение отрицательно , то *n* — внешняя нормаль. В этом случае внутреннюю нормаль можно получить, умножив *n* на −1: *n\_*внутренняя​=−*n*. | **18. Алгоритмы отсечения отрезков в пространстве. Трехмерный алгоритм разбиения средней точкой. Трехмерный алгоритм Кируса-Бека.**  Постановка задачи: отсечение объема у отрезка, где отсекатель - 3хмерное тело  Дано: V - точки отсекателя, P1,P2 - точки отсекаемого отрезка. Отсекатель выпуклый  **Трехмерный алгоритм разбиения точкой**  Можно каждой концевой точке присвоить код Cазерленда-Коэна. В трехмерном случае он будет 6тизначный. Единица интерпретируется:   * В первом бите - конец ребра левее объема * Во втором бте - конец ребра правее объема * В третьем бите - конец ребра ниже объема * В 4 бите - конец ребра выше объема * В 5 бите - конец ребра ближе объема * В 6 бите - конец ребра дальше объема   Рассматриваются два выражения:   * code(P1)|code(P2) = 0 - весь отрезок внутри объема * code(P1)& code(P2) != 0 - весь отрезок снаружи   Эти случаи мы отбрасываем, так как в первом случае весь отрезок внутри и результат выполнения алгоритма - весь отрезок. Во втором случае результат - пустое множество.  Если же отрезок внутри объема частично(не выполнилось ни одно из условий выше), делаем так:   * разбиваем рабочий отрезок на 2 части; * для каждой части проверяем 2 условия выше; * если какая то из частей лежит внутри целиком, то мы запоминаем ее в ответ. * Рабочим отрезком становится тот, который частично лежит снаружи;   Повторяем данное упражнение пока отрезок не выродился в точку или пока не найдено пересечение. (На самом деле мне кажется пересечение в данном случае может выглядеть так: один отрезок полностью внутри фигуры а второй полностью снаружи, но тогда наверное нужно проверять не отрезки вида [(x1,y1) (x2,y2)], а не включая граничные точки).  Асимптотика O(logN) где N - длина отрезка в пикселях. (это не пишите, но может спросить  **Трехмерный алгоритм Кируса-Бека**  Пусть у нас есть к граней. Т.к. объем все еще выпуклый то точек пересечения две. Пусть Pi - граничная точка выпуклой области, n - внутренняя нормаль к грани в этой точке. Можем задать отрезок параметрически : P(t) = P1 + (P2-P1)t, 0<=t<=1  Есть три критерия: считаем n\*(P(t) - Pi) = F   * F < 0 - точка вне V * F =0 - точка на границе * F>0 - точка внутри   Точка пересечения для каждой плоскости определяется решением уравнения n(P(t) -Pi) = 0  Всего должно быть 2 таких решения.  В худшем случае работаем за O(k), где k - количество граней (тоже не пишите лишний раз) |
| --- | --- |
| 19 **Разбиение невыпуклых многоугольников. Разбиение невыпуклых тел.** Для разбиения невыпуклых многоугольников используется обобщения метода поворотов и переносов окна. Если вершины многоугольника перечисляются против часовой стрелки, то алгоритм имеет следующий вид:   * Многоугольник переносят так, чтобы его i-я вершина совпадала с началом координат. * Затем многоугольник следует повернуть относительно координат по часовой стрелке так, чтобы (i+1)-я вершина оказалась на положительной полуоси x (). * Проанализировать . * Если , то многоугольник выпуклый в (i+1)-й вершине. * Если , то многоугольник невыпуклый в (i+1)-й вершине. Следует его разбить на два многоугольника вдоль положительной полуоси x. Необходимо найти все его стороны, пересекающиеся с осью x. В результате получается два многоугольника. Первый состоит из вершин, лежащих выше оси x, и точки пересечения, ближайшей к началу координат. Второй — из вершин, лежащих ниже оси x, и точки пересечения.   Данный алгоритм применяется рекурсивно к полученным многоугольникам, пока все многоугольники не станут выпуклыми.  (и тд пока не р/м все вершины)  Разбиение невыпуклых тел.  Для каждой полигональной грани тела:   * Перенести тело так, чтобы одна из вершин выбранной грани совпала с началом координат. * Повернуть тело вокруг начала координат так, чтобы одно из инцидентных ему рёбер совпало с одной из осей координат, например с осью x. * Повернуть тело вокруг выбранной оси координат так, чтобы выбранная грань совпала с одной из координатных плоскостей, например с плоскостью z = 0. * Проверить знаки координаты, которая перпендикулярна выбранной грани (т.е. координаты z), для всех вершин тела, не лежащих на этой грани. * Если все эти знаки совпадают или равны нулю, то тело является выпуклым относительно этой грани. В противном случае оно невыпукло; разрезать тело плоскостью, несущей выбранную грань. * Повторить всю процедуру с каждым из вновь образовавшихся тел. Продолжать работу до тех пор, пока все тела не станут выпуклыми. | 20 **Проекционные преобразования. Классификация плоских геометрических проекций. Ортографические проекции (схему первую срисовать с 22 билета!!)**  (проекционная плоскость = картинная плоскость)  В общем случае проекции преобразуют точки, заданные в системе координат размерностью n, в системы координат размерностью меньше чем n.  Проекторы - проекционные линии, которые соединяют каждую точку проецируемого объекта с поверхностью. Плоскими геометрическими проекциями называется такой класс проекций, где происходит проецирование из трехмерной системы координат в двумерную, так как проецирование производится на плоскость, а не на искривленную поверхность и в качестве проекторов используются прямые, а не кривые линии. Плоские геометрические проекции в дальнейшем будем называть просто проекциями. Они представляют основу описательной геометрии.  Если центр проекции расположен в конечной точке трехмерного пространства, получается перспективная проекция. Если центр расположен в бесконечности, то все проекторы параллельны, и результат является параллельной проекцией. Неплоские и негеометрические проекции также полезны; они широко используются в картографии.  Дано: V - объект в трехмерном пространстве M - проекционная плоскость, на которую будет проецироваться объект V  Проекции делятся на два основных класса: параллельные (аксонометрические) и; центральные. Параллельные проекции делятся на два типа в зависимости от соотношения между направлением проецирования и нормалью к проекционной плоскости: 1) ортогональные – направления совпадают, т. е. направление проецирования является нормалью к проекционной плоскости; 2) косоугольные – направление проецирования и нормаль к проекционной плоскости не совпадают  Изображение выглядит как диаграмма, текст, снимок экрана, линия  Автоматически созданное описание  Центральная проекция любой совокупности параллельных прямых, которые не параллельны проекционной плоскости, будет сходиться в точке схода.  К ортографическим проекциям можно отнести вид спереди, вид сверху, вид сбоку, это такие проекции, где проекционная плоскость перпендикулярна главным координатным осям. То есть проекция на x = 0, y = 0 или z = 0.  Все шесть стандартных видов (спереди, сзади, слева, справа, сверху, снизу) могут быть получены комбинациями отражения, вращения и переноса с последующим проецированием на плоскость z = 0 z=0. Например: Вид сзади: отражение относительно плоскости z = 0 z=0 и проецирование на эту плоскость. Вид слева: вращение вокруг оси y на угол + 90 ∘ +90 ∘ и проецирование на плоскость z = 0.  Матрица проекции на плоскость z=0:Изображение выглядит как линия, диаграмма, Параллельный, зарисовка  Автоматически созданное описание  Подходы к преобразованиям. Для разработки различных преобразований можно использовать два разных подхода:  Фиксированный центр проекции:   * Центр проекции (точка зрения) фиксирован. * Плоскость проекции перпендикулярна каждому проектору. * Объект манипулируется для получения требуемого вида. * Этот подход напоминает изучение небольшого объекта (например, книги), который поворачивают и перемещают для осмотра со всех сторон.   Фиксированный объект:   * Объект фиксирован. * Центр проекции может перемещаться в трехмерном пространстве. * Плоскость проекции не обязательно перпендикулярна направлению взгляда. * Этот подход напоминает изучение большого объекта (например, автомобиля), когда наблюдатель перемещается вокруг него для осмотра с разных ракурсов.   Оба подхода математически эквивалентны. В компьютерной графике выбор подхода зависит от задачи. Например, для статичного изображения подходит первый подход, а для моделирования движения (например, в тренажерах или архитектурных моделях) — второй.  Вспомогательные виды и сечения Для объектов с гранями, не параллельными координатным плоскостям, стандартные ортографические виды не показывают их истинную форму. В таких случаях используются вспомогательные виды: Объект вращают и перемещают так, чтобы нормаль к грани совпала с одной из координатных осей. Результат проецируют на координатную плоскость, перпендикулярную этой оси. Для отображения внутренних деталей сложных объектов применяются сечения: Через объект проводят секущую плоскость. Удаляют часть объекта по одну сторону от плоскости. Оставшуюся часть проецируют на плоскость сечения. |

| **21** [**Аксонометрические проекции. Триметрия, диметрия, изометрия**](https://docs.google.com/document/d/1iNOTvSUtFaczKkrscoj7jXFpCmRY4edl/edit#heading=h.4i7ojhp)  **Дано:** объект *V* — некоторый объект в трёхмерном пространстве. плоскость проекции *M* — на неё будет проецироваться объект *V*.  Если нормаль к проекционной плоскости параллельна направлению проецирования, но не параллельна осям координат, то такие проекции называются ***аксонометрическими*.** Сохраняется параллельность прямых, а углы изменяются; расстояние можно измерить вдоль каждой из главных координатных осей (в общем случае с различными масштабными коэффициентами).  Представляют интерес три аксонометрические проекции: **триметрическая, диметрическая, изометрическая**. Изометрическая - частный случай диметрической, а диметрическая - частный случай триметрической. Таким образом в изометрической больше всего ограничений, а триметрической - меньше всего.  **Триметрическая проекция** строится произвольными поворотами вокруг любых координатных осей, совершаемыми в произвольном порядке, с последующим проецированием на плоскость z=0. В общем случае для триметрической проекции коэффициенты искажения по каждой из проецируемых главных осей(x, y и z) не равны друг другу. Для любой конкретной триметричейской проекции коэффициенты искажения вычисляются с помощью применения общей матрицы преобразования к единичным векторам главных осей. (Главная ось - ось объекта в исходном пространстве параллельна одной из координатных осей x, y, z)  На картинке: [U] - матрица единичных векторов вдоль нетрансформированных осей x, y, z соответственно, [T] - общая матрица триметрической проекции. Тогда коэффициенты искажения вдоль спроецированных главных осей равны.  **Диметрическая проекция** - триметрическая проекция с двумя одинаковыми коэффициентами искажения, третий коэффициент может иметь любое значение. Диметрическая проекция строится с помощью поворота на угол φ вокруг оси y, затем на угол θ вокруг оси x и проецирования на ось z=0 с центром проекции, расположенным в бесконечности.  **Изометрическая проекция** - триметрическая проекция с тремя одинаковыми коэффициентами искажения. | 22 **Косоугольные проекции. Проекции Кавалье и Кабине**    Косоугольная проекция — это вид параллельной проекции, при которой проекторы (лучи) направлены под углом к плоскости проекции. В отличие от ортографической и аксонометрической проекций, где лучи перпендикулярны плоскости, в косоугольной проекции они наклонены. Центр проекции находится на бесконечности, поэтому все проекторы параллельны.page60image61872432  Косоугольная проекция показывает общую трехмерную форму объекта. Однако истинные размеры и форма сохраняются только для тех граней, которые лежат параллельно плоскости проекции. Углы и длины этих граней не искажаются — как в ортографическом виде спереди. Все остальные части объекта деформируются.  Два самых известных типа косоугольных проекций — это кавалье и кабине :  **1. Кавалье** :Угол между проекторами и плоскостью проекции: β = 45°  Коэффициент искажения f = 1. Это означает, что все три главных направления (x, y, z) изображаются без сокращения. Результат выглядит "утолщённым", потому что глубина передается в полный размер.  Формула связи угла и коэффициента искажения:  β = arcctg(f)  При f = 1: β = arcctg(1) = 45°  Матрица проекции (3D → 2D), если проецируем на плоскость Z=0:    где  a = f · cos(α)  b = f · sin(α)  α - угол между горизонталью и спроецированной осью Z.  f - длина спроецированного единичного вектора на оси Z, т.е. коэф. искажения  β - угол между косыми проекторами и плоскостью  **2. Кабине** : Коэффициент искажения f = 1/2  Угол между проекторами и плоскостью проекции: β = arcctg(1/2) ≈ 63.43°  Глубина сокращается в два раза, поэтому изображение выглядит более реалистично по сравнению с кавалье.  Та же матрица используется, но с другими значениями a и b (формула та же, но другое f):  a = (1/2) · cos(α)  b = (1/2) · sin(α)  Поскольку искажается истинная форма одной грани. косоугольные проекции особенно подходят для иллюстрации объектов с круглыми или иными искривленными гранями. Такие грани должны быть параллельны плоскости проекции, чтобы избежать нежелательных искажений. |
| --- | --- |
| 23. Последовательное отсечение многоугольника. Алгоритм Сазерленда-Ходжмана  **Проблема**: Для выпуклого произвольного многоугольника построить отсечение выпуклым окном  **Дано**: многоугольник P (p1,p2, ... pn), отсекатель или окно Q(q1,q2, .... qm)  **Найти**: С (с1... сp) - точки отсеченного многоугольника.  **Алгоритмическая сложность**: O(n\*m)  Основная идея:  1.Берем первое ребро отсекаемого многоугольника.  2.Рассматриваем его по каждому отсекателю  3.Берем точки пересечения и удаляем остаток  4.Проходимся по всем ребрам  Изначально все вершины многоугольника находятся в списке в порядке обхода по часовой стрелке(!). На каждой итерации из списка последовательно выбираются две идущие подряд вершины и проверяется их расположение относительно определенного ребра.  Существует 4 возможных взаиморасположения ребра отсекателя и двух вершин многоугольника.  При проверке каждой последующей точки в список добавляется от 0(случай 2) до 2(случай 4) точек(в случаях 1 и 3 добавляется по одной точке). – Если ребро из области видимости выходит, то надо определить и занести в результат точку пересечения ребер многоугольника и отсекателя. – Если же ребро входит в область видимости, то следует поступить точно также, но в этом случае в результат необходимо занести и точку Р. Для определения положения точки относительно ребра отсекателя можно составить следующее векторное произведение(л). Для нахождения точки пересечения отрезков можно воспользоваться параметрическими уравнениями отрезков(п). P(s) и P(t) являются векторно-значными функциями, то есть P(s) = [x(s) y(s)], P(t) · [x(t) y(t)].  Приравнение соответствующих компонент дает систему:  Решение t0 и S0 этой системы позволяет найти точку пересечения путем подставления одного из параметра в соответствующее ему уравнение отрезка. Недостатком этого алгоритма является появления ложных ребер на границе отсекателя. Этот алгоритм применим только в случае, если отсекатель является выпуклым многоугольником. | **24. Поиск пересечений многоугольников. Постановка задачи поиска пересечений многоугольников. Базовые оценки сложности.**  ***Дано:*** P и Q – выпуклые многоугольники с L и M вершинами. Положим L <= M.  ***Требуется:***построить их пересечение.  *Теорема.* Пересечением выпуклых L-угольника и M-угольника является выпуклый многоугольник, имеющий не более L + M вершин.  *Док-во:* пересечение P и Q образовано пересечением L + M внутренних полуплоскостей, определяемых рёбрами этих двух многоугольников, ч.т.д..  Граница P ∩ Q состоит из чередующихся цепей вершин этих двух многоугольников (рис. 7.4).  Если использовать простой обход границы P, где шаг за шагом определять пересечение с границей Q, это потребует O(L\*M) времени, поскольку будет проверено пересечение каждого ребра P с каждым ребром Q. Естественно для сокращения вычислительной работы попытаться использовать факт выпуклости многоугольников.  Один из подходов заключается в разбиении плоскости на области, в каждой из которых пересечение этих двух многоугольников можно вычислить легко. В данном случае будет адекватным простейшее разбиение плоскости – индуцированное пучком лучей (алгоритм **Шеймоса-Хоуи**).  1. Возьмём произвольную точку *О* (классически – бесконечно удаленную).  2. Из *О* проведём лучи через все вершины P и Q. Эти лучи разбили плоскость на секторы.  3. Мы хотим упорядочить этот набор лучей по углу вокруг т. *О*:   * Если *О* лежит внутри многоугольника, скажем, P, то лучи к вершинам P упорядочиваются относительно *О* так же, как и вершины, порядок которых задан. * Если же *О* лежит вне P, то вершины P образуют две цепи, а для каждой цепи соответствующие ей лучи из *О* отсортированы по углу.   Поэтому секторы, определяемые вершинами P, упорядочиваются по углу за линейное время от числа вершин.  4. Результирующие куски можно собрать воедино за один обход секторов, на который понадобится линейное время.  5. Очищающий просмотр для удаления ложных вершин, которые возникают на границах между секторами.  **Ключевым является наблюдение**, что пересечение каждого сектора с выпуклым многоугольником образует *четырехугольник* (*трапецию*, если *О* – бесконечно удаленная точка). Поэтому внутри каждого сектора пересечением P и Q будет пересечение двух четырехугольников, которое можно найти за *Const*.  *Теорема*. Пересечение выпуклых L-угольника и M-угольника можно построить за время Θ(L+M) (тета!!). – Оценка и сверху и снизу. Алгоритм будет работать ни быстрее, не медленнее L + M.    Пример. Точка О – бесконечно удалена. Полярный угол относительно О заменен значением абсциссы x. |

| [**25 вопрос. Алгоритм построения пересечения двух выпуклых многоугольников. Теорема о времени построения пересечения двух выпуклых многоугольников**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.2bn6wsx)**.**  **Дано**: Выпуклые многоугольники P и Q c L и M вершинами.  **Требуется**: Построить пересечение многоугольников.  **Ограничение**: L <= M.  **Теорема**: Пересечение выпуклых L-угольника и M-угольника можно построить за время θ(L+M).  Алгоритма Шеймоса-Хоуи:  1. P и Q выпуклые => их вершины могут быть отсортированы по возрастанию и объединены в один общий список за O(L+M)  2. Проводим через вершины многоугольников вертикальные полосы, образующие L+M-1 полосу  3. Пересечение полосы с многоугольником является трапецией. Полосы рассматриваются последовательно слева направо, для каждой полосы время O(1) вычисляется пересечение трапеций, затем пересечения объединяются в многоугольник(Искомый). Также удаляются ложные вершины, которые возникают между секторами.  **Можем усовершенствовать подход.** Предположим, что многоугольники точно пересекаются. Границей P\* (многоугольника-пересечения) является черед. последовательность участков границ P и Q. Если одним из таких участков является кусок границы P, то некий участок границы Q окружает её во внешней области P\* (след. рис.).    Дальше определяется механизм продвижения. **Идея** заключается в том, чтобы не двигаться по той границе (P или Q), текущее ребро которой ещё может содержать необнаруженную точку пересечения. Движение идет и по q и по p. Тогда в случае (2) мы продвигаемся по P, поскольку текущее ребро вектор q₍i–1₎qᵢ из Q может содержать еще необнаруженную точку пересечения; по тем же причинам в случае (3) мы продвигаемся по границе Q. В случае (4) все пересечения на текущем ребре из P уже обнаружены, в то время как текущее ребро из Q все еще может содержать необнаруженное пересечение; поэтому мы продвигаемся по границе P. Наконец, в случае (1) выбор произволен, и выберем, например, Q. Рассмотрение этих четырех случаев полностью определяет механизм продвижения в данном методе, осуществляемый подпрограммой ДВИЖЕНИЕ. Выражение «ДВИЖЕНИЕ реализует случай (j)» (при j = 1, 2, 3, 4) означает, что продвижение, соответствующее случаю (j), выполняется корректно. Если пренебречь для ясности и простоты вырожденными случаями (когда граница P содержит вершину Q и наоборот), которые требуют особого внимания¹), то можно формализовать алгоритм. Заметим, что pL и qM непосредственно предшествуют p₁ и q₁ соответственно (это означает, что, как обычно, p₀ = pL и q₀ = qM). | **26. Алгоритм Вейлера-Азертона пересечения произвольных многоугольников**  Используется в компьютерной графике для нахождения области пересечения отсекаемогомногоугольника по отсекающему многоугольнику. Алгоритм может обрабатывать многоугольники с окнами.  ● **Проблема:** Найти такое подмножество точек, которое будет находиться и в первом и во втором заданном многоугольнике.  ● **Дано:** первый многоугольник V (v1,v2, ... vn), второй многоугольник M(m1,m2, .... mm)  ● **Найти:** С (с1... сp) - точки отсеченного многоугольника.  Многоугольники задаются **циклическими списками вершин**.  Обход внешней границы — **по часовой стрелке**, внутренних (окна) — **против**.  Алгоритм отслеживает пересечения, чередуя переходы от одной границы к другой, пока не образуется замкнутая фигура.  Границы обрабатываемого и отсекающего многоугольников могут пересекаться или не пересекаться между собой. Если они пересекаются, то точки пересечения образуют пары. Одно пересечение из пары возникает, когда ребро обрабатываемого многоугольника входит внутрь отсекающего многоугольника, а другое — когда оно выходит наружу.      Случай с Окном: |
| --- | --- |
| **27** [**Геометрический поиск. Постановка задачи, оценки сложности построения решения задач регионального поиска, локализации.**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3as4poj)  Пусть имеется набор геометрических данных и нужно узнать, обладают ли они определенным свойством (например, выпуклостью). Когда этот вопрос возникает единожды, то он называется уникальным. Но есть запросы, обработка которых повторяется многократно на том же самом файле. Такие вопросы называются массовыми.  Такие запросы можно выполнять, используя специальную структуру. Для этого нужно знать:  1. Время запроса (сколько времени необходимо как в среднем, так и в худшем случае на 1 запрос)  2. Память (сколько памяти необходимо для структуры данных)  3. Время предобработки (сколько времени необходимо для организации данных перед поиском)  4. Время корректировки (указан элемент данных, сколько времени потребуется на его включение в структуру данных или удаление из нее)    Региональный поиск:  Дано: N точек на плоскости, прямоугольник, стороны которого параллельны координатным осям: [a, b] X [c, d]  Найти: сколько из точек лежит внутри заданного прямоугольника. То есть сколько точек (x, y) удовлетворяют неравенствам a <= x <= b, c <= y <= d?  Наивный алгоритм:  Уникальный региональный запрос может быть обработан за линейное время, так как надо только проверить каждую из N точек, чтобы увидеть удовлетворяет ли она неравенствам, задающим прямоугольник. Аналогично нужна линейная затрата памяти, так как следует хранить только 2N координат. Нет никаких затрат на предобработку, а время корректировки для новой точки равно константе.  Но вот с массовыми запросами начинается сложность. Мы не знаем такое упорядочение точек, чтобы любой новый прямоугольник мог быть с ним согласован. Не можем решить задачу наперед для всех возможных прямоугольников, так как их число бесконечно.  Метод локусов:  Q(p) – число точек, над которым доминирует точка p, то есть таких точек m, у которых: y\_m < y\_p, x\_m < x\_p.  Тогда число точек N(p1p2p3p4) в прямоугольнике p1p2p3p4 определяется следующим образом: N(p1p2p3p4) = Q(p1) – Q(p2)- Q(p4) + Q(p3). (следует из комбинаторного правила включения-исключения: над всеми точками доминирует p1, нужно исключить такие точки, над которыми доминируют p2, p4, но это приведет к тому, что часть точек будет исключена дважды, но эти точки лежат в левом нижнем квадранте вершины p3)  Итак, задача регионального поиска сведена к задаче обработки запросов о доминировании для 4 точек. (Свойство, которое упростит жизнь: на плоскости существуют области удобной формы, внутри которых число доминирования Q является константой)  Предположим, что из наших точек(ну вот которых N штук) на оси x и y опущены перпендикуляры, а полученные линии продолжены в бесконечность. Они создают (N+1)^2 прямоугольников.  Для всех точек p в любом из таких прямоугольников (ячеек) Q(p)-константа, при переходе между ячейками изменяется либо количество точек слева (*x*<*x\_p*​), либо ниже (*y*<*y\_p*​).  Упорядочим исходные точки по исходным координатам, остается только выполнить 2 двоичных поиска для 4 точек, чтобы найти ячейку, содержащую нашу точку. Значит время запроса будет равным O(log N), но нужно O((N + 1)^2) памяти. | **28 Алгоритмы решения задач: нахождение ближайшей пары, нахождение ближайшего соседа.**  Нахождение ближайшей пары  Дано: N точек  Найти: две из них расстояние между которыми наименьшее  Наивный алгоритм: за O(N^2) проверить все пары точек  Алгоритм «Разделяй и властвуй»(O(NlogN))   1. Отсортируем все точки по их x -координате. Это займет O(N log N). 2. Разделим множество точек на две примерно равные половины вертикальной линией x = xmid , где xmid — медианная x -координата. 3. Рекурсивно найдем ближайшую пару в левой и правой половинах. Пусть минимальные расстояния в этих половинах равны d\_L и d\_R соответственно. 4. Обозначим d = min(d\_L, d\_R) . 5. Теперь нужно проверить, есть ли пара точек, одна из которых находится в левой половине, а другая — в правой, расстояние между которыми меньше d. 6. Для этого рассмотрим только те точки, которые находятся в полосе шириной 2d вокруг разделяющей линии. 7. Отсортируем эти точки по y-координате (это можно сделать заранее, чтобы не сортировать на каждом шаге). И пойдем сверху вниз. 8. Для каждой точки в этой полосе достаточно проверить только следующие 6 точек (из-за геометрических соображений), чтобы убедиться, что нет ближайшей пары. Таким образом, проверка займет O(N) времени.   По картинке 1 посмотрим, в одном квадрате не более 4 точек(иначе d было бы меньше чем оно есть). Для каждой точки надо проверить квадрат справа(причем только точки которые ниже самой точки) и квадрат справа снизу. Итого 6 точек.  Нахождение ближайшего соседа  Дано: N точек  Найти: ближайшего соседа для каждой точки множества  Наивный алгоритм работает за O(n^2). Для одной точки надо проверить N-1 соседей.  Можно решить через разделяй и властвуй за O(NlogN) только для каждой точки надо сохранять dmin.( в учебнике инсайдерская информация про O(N)- я в это не верю, это дичь просто) |

| 29 **Региональный поиск, 2d дерево. Локализация точек. Метод полос, метод цепей. Метод триангуляции**  1. Задача геометрического поиска.  Дано множество точек M – const;  Регион (область) R(v1, ..., vn, v1) – var;  Вычислить k – число точек, находящихся внутри региона. Сложность (метод локусов или 2d-дерево) – O(log n)  Двумерное пространство (плоскость) рассматривается как обобщённый прямоугольник, который рекурсивно разрезается попеременно вертикальными и горизонтальными прямыми, каждая из которых проходит через выбранную точку и делит множество точек на два подмножества приблизительно равной мощности. Эти разрезы реализуются в виде двоичного дерева, где каждый узел содержит: координату и направление разреза, точку, по которой производится разрез, а также ссылки на левое и правое поддеревья, соответствующие частям пространства по разные стороны от секущей прямой.  Построение дерева:   1. В корневом узле выбирается медиана координаты 𝑥 (или 𝑦) и соответствующая точка. 2. Пространство делится вертикальной (или горизонтальной) прямой, проходящей через эту точку. 3. Процесс продолжается рекурсивно: в каждом узле направление разреза чередуется (x → y → x → ...), и точки делятся в соответствии с медианой соответствующей координаты. 4. Разбиение завершается, когда в подмножестве не остаётся точек (лист дерева).   Региональный поиск  Поиск всех точек, попадающих в заданный прямоугольный регион, осуществляется с помощью стратегии «разделяй и властвуй»:   * В каждом узле проверяется, пересекается ли запросный регион с областью, ассоциированной с узлом. * Если да — осуществляется проверка на принадлежность точки региону, а затем рекурсивный спуск в соответствующие поддеревья. * Поиск прекращается в листах дерева или в поддеревьях, не пересекающихся с запросом.   2. Задача локализации.  Дано множество регионов Ri; точка P;  Найти Ri, содержащий P  Сложность при решении в лоб – O(nm) (по планарному графу), сложность при использовании метода полос или цепей – O(log n)  Локализация точки на планарном подразбиении сводится к преобразованию исходного плоского графа в структуру, удобную для эффективного поиска, в частности — с возможностью применения двоичного поиска.. Метод полос для локализации точки основан на разбиении плоскости на горизонтальные полосы, проходящие через вершины плоского прямолинейного графа (ППЛГ). Каждая полоса содержит отрезки ребер графа, не пересекающиеся внутри полосы, что позволяет упорядочить их слева направо и применять двоичный поиск. Предобработка выполняется методом «заметающей прямой» с использованием сбалансированного дерева, обеспечивая время запроса  𝑂(log^2𝑀), предобработку за 𝑂(𝑁log𝑀) и использование 𝑂(𝑁^2) памяти в худшем случае. Метод эффективен по времени запроса, но затратен по памяти. Метод цепей используется для задачи локализации точки в планарной карте и основан на разбиении исходного планарного графа на так называемые цепи — последовательности связанных рёбер, образующих ломаную линию. Основная идея метода — использование монотонных цепей, которые упрощают проверку принадлежности точки к области.  Монотонная цепь — это такая, которую любая прямая, ортогональная заданному направлению, пересекает не более чем в одной точке. Благодаря этой геометрической упорядоченности дискриминация (определение положения точки относительно цепи) сводится к двоичному поиску и линейной проверке, что обеспечивает логарифмическое время поиска.  Если удаётся построить полное множество монотонных цепей, упорядоченных по положению и покрывающих весь граф, то локализация точки осуществляется с помощью двоичного поиска по этим цепям. В результате метод цепей обеспечивает локализацию точки за время O(log² n).  3. Задача триангуляции.  Дано множество точек M.  Нужно построить триангуляцию – треугольники с вершинами из M так, чтобы треугольники образовывали разбиение рассматриваемого куска плоскости (разбиение выпуклой оболочки множества M)  Сложность для построения триангуляции Делоне (для каждого треугольника можно описать окружность, которая не содержит других точек триангуляции) – O(n log n).  Сложность для построения триангуляции без ограничений – O(n).  Метод детализации триангуляции заключается в построении иерархии триангуляций (разбиений на треугольники), где на каждом уровне удаляются несмежные внутренние вершины малой степени, а образовавшиеся области повторно триангулируются. Полученная структура задаётся в виде направленного ациклического графа, позволяющего локализовать точку за O(log N) времени при O(N) памяти после предобработки. | 30 **Построение выпуклых оболочек. Постановка задачи, оценка сложности построения решения. Алгоритм Грехема. Алгоритм Джарвиса**  Дано: множество точек на плоскости, каждая из которых задана парой чисел — координатами (в декартовой системе координат). Таким образом, дано множество пар вещественных чисел.  Требуется: определить последовательность точек из данного множества, образующих его выпуклую оболочку. Выпуклой оболочкой множества X называется наименьшее выпуклое множество, содержащее X. Здесь «наименьшее множество» означает минимальный элемент по отношению к включению множеств, то есть такое выпуклое множество, которое содержит данную фигуру и при этом содержится в любом другом выпуклом множестве, содержащем эту фигуру.  В рамках задачи требуется упорядочить найденные точки так, чтобы соседние элементы в последовательности были смежными вершинами многоугольника, ограничивающего выпуклую оболочку. Иными словами, полученная последовательность точек должна представлять собой обход этого многоугольника по часовой или против часовой стрелки.  Ограничения: исходное множество точек содержит не менее трёх элементов  Выбор начальной точки: Берётся самая левая нижняя точка p 1 ​ (если таких несколько, выбирается самая левая из самых нижних). Эта точка гарантированно входит в выпуклую оболочку. Поиск выполняется за O ( n )) путём перебора всех точек.  Построение оболочки: Следующая точка p 2 ​ выбирается так, чтобы она имела наименьший положительный полярный угол относительно p 1​ (при условии, что ось направлена вправо от p 1 ​ ). Далее для каждой текущей точки p i ​ ( i ≥ 2) ищется точка p i + 1 , которая образует наибольший угол против часовой стрелки между прямыми p (*i − 1)\*p i и p*( i + 1)\*p i ​​ . Поиск каждой следующей точки выполняется за O ( n ). Процесс повторяется, пока не будет замкнута оболочка (возврат к начальной точке p 1 ​ ).  В процессе работы можно вычислять не угол, а косинус угла между прямыми с помощью скалярного  произведения векторов. Наибольший угол  соответствует наименьшему косинусу  Псевдокод алгоритма:  Jarvis(P):  1) p[1] = найти\_крайнюю\_левую\_нижнюю\_точку(P)  2) p[2] = найти\_точку\_с\_мин\_полярным\_углом(P, p[1])  3) i = 2  4)do:  a) p[i+1] = p[1] # инициализация  b) for j от 1 до |P|:  if p[j] не в оболочке или p[j] == p[1]:  if косинус\_угла(p[i-1], p[i], p[j]) <  косинус\_угла(p[i-1], p[i], p[i+1]):  p[i+1] = p[j]  c) i = i + 1  while p[i] != p[1]  5) return массив p[1..i]  В лучшем случае: O(nh), где h - количество вершин выпуклой оболочки В худшем случае (когда все точки являются вершинами оболочки): O(n²) Каждая итерация внешнего цикла требует O(n) операций Количество итераций равно h (числу вершин оболочки)  Алгоритм Грэхема решает задачу построения  выпуклой оболочки, используя стек точек-  кандидатов. Все точки входного множества  последовательно обрабатываются, при этом  точки, не являющиеся вершинами выпуклой  оболочки, удаляются из стека. В результате в  стеке остаются только вершины оболочки,  упорядоченные против часовой стрелки.  Выбор начальной точки (p0): Находим точку с  минимальной y-координатой. Если таких точек  несколько, выбираем самую левую из них  Сортировка остальных точек: Сортируем точки  {p1, ..., pm} по возрастанию полярного угла  относительно p0. При совпадении углов  сортируем по расстоянию до p0 (ближние точки  сначала)  Инициализация стека: Помещаем в стек p0 и  первую отсортированную точку p1  Построение оболочки: Для каждой точки pi (i = 2..m): Удаляем точки из стека, пока они не образуют левый поворот с текущей точкой Добавляем текущую точку в стек  Результат: Стек содержит вершины выпуклой оболочки в порядке обхода против часовой стрелки  Graham(Q):  1) p0 = точка с минимальной Y-координатой (при равенстве - самая левая)  2) {p1, ..., pm} = сортировка(Q\{p0}) по полярному углу относительно p0  (при равных углах - по возрастанию расстояния до p0)  3) S = новый стек  4) Push(p0, S)  5) Push(p1, S)  6) for i от 2 до m:  while угол между NextToTop(S), Top(S) и pi не является левым поворотом:  Pop(S)  Push(pi, S)  7) return S |
| --- | --- |
| **31. Быстрые алгоритмы построения выпуклых оболочек. Построение выпуклой оболочки простого (произвольного) многоугольника. Открытые алгоритмы**    **1. Быстрые алгоритмы** построения выпуклой оболочки (БЫСТРОБОЛ) вдохновлены принципом работы алгоритма быстрой сортировки (QuickSort / БЫСТРСОРТ). Идея заключается в рекурсивном разбиении множества точек и последующем объединении частичных решений.  **Постановка задачи:**  Дано: множество точек на плоскости.  Найти: выпуклую оболочку, содержащую все заданные точки.  Алгоритмическая сложность:  - В среднем случае: O(N log N)  - В худшем случае: O(N²)  **Описание алгоритма:**  1. Начальное разбиение множества S:  - Выбираются две точки: l — с наименьшей абсциссой, r — с наибольшей абсциссой.  - Через эти точки проводится прямая, которая делит множество S на два подмножества:  - S1 — точки, расположенные выше или на прямой lr.  - S2 — точки, расположенные ниже или на той же прямой lr.  - Заметим, что точки l и r принадлежат обоим подмножествам.  2. Рекурсивная обработка подмножеств:  - Для каждого подмножества (например, S(1)) выбирается точка h, такая что треугольник (h, l, r) имеет максимальную площадь.  - Если таких точек несколько, выбирается та, у которой угол ∠hlr больше.  - Точка h гарантированно принадлежит выпуклой оболочке.  3. Дальнейшее разбиение множества:  - Строятся две прямые: L1 (из l в h) и L2 (из h в r).  - Все точки множества S(1) анализируются относительно этих прямых:  - Точки справа от обеих прямых — внутренние для треугольника (lrh), их можно удалить.  - Оставшиеся точки делятся на два новых множества:  - S(1,1): точки слева от L1 или на ней, но справа от L2.  - S(1,2): аналогично, но относительно другой комбинации прямых.  - Эти новые множества передаются на следующий уровень рекурсии.  4. Примитивные операции: вычисление площади треугольника, определение положения точки относительно прямой. Каждая из этих операций требует нескольких арифметических действий.  5. Обработка начального случая: чтобы корректно начать процесс, используется искусственная точка r0 = (x0, y0 - ε), где ε — малое положительное число. Это позволяет использовать вертикальную прямую как начальную границу разбиения.После завершения алгоритма точка r0 удаляется (ε полагается равным 0).  **Анализ сложности:** - На каждом уровне рекурсии выделение подмножеств S(1) и S(2) занимает O(N) операций. - При условии, что размеры подмножеств уменьшаются не менее чем в константное число раз на каждом уровне, общая сложность составляет O(N log N). - Однако в худшем случае, как и у алгоритма БЫСТРСОРТ, сложность достигает O(N²)    **2. Построение выпуклой оболочки простого (произвольного) многоугольника.**  **Постановка задачи:**  Дано:  - Простой многоугольник P, заданный упорядоченной циклической последовательностью вершин (q1, q2, ..., qm), ориентированной по часовой стрелке.  - Внутренность многоугольника P находится справа при обходе его границы.  - p1 — самая левая вершина многоугольника.  - pm — самая правая вершина многоугольника.  Требуется: построить выпуклую оболочку для цепи (p1, p2, ..., pm) простого многоугольника P, которая состоит из подпоследовательности вершин исходного многоугольника.  Ограничения:  - Многоугольник P является простым (без самопересечений).  - Последовательность вершин задана в порядке обхода по часовой стрелке.  - Алгоритм должен корректно обрабатывать все возможные конфигурации вершин внутри карманов, образуемых текущей оболочкой.  Сложность:  - Временная сложность: O(N), где N = m — количество вершин многоугольника.  - Используется один стек или список критических вершин для хранения промежуточных результатов.  **Th:** Выпуклая оболочка простого многоугольника с N вершинами может быть построена за оптимальное время 0(N) при использовании памяти объемом 0(N).  **Решение:**  1. Инициализация:  - Найти самую левую вершину p1 и самую правую вершину pm.  - Эти точки являются начальной и конечной точками верхней оболочки.  - Обозначить q1 = p1, qr = pm. Начальная оболочка — (q1).  2. Проход по цепи:  - Последовательно просматриваются вершины от p1 до pm.  - Для каждой новой вершины v проверяется её положение относительно последнего ребра текущей оболочки qi−1qi.  - Вершина v может находиться в одной из четырёх областей, определенных положением предыдущих вершин (рисунок 4.10).  3. Обработка вершины v:  - Если v находится в области 1:  - Граница заходит внутрь кармана. Продвигаться дальше по границе до тех пор, пока не найдётся вершина вне кармана (в области 2).  - После этого перейти к обработке как для случая 2.  - Если v находится в области 2:  - Это критическая вершина. Строится опорная прямая из v к текущей оболочке.  - Все вершины между последним подходящим qr и qi удаляются.  - Добавляется новая вершина v как qr+1 (рисунок 4.11б).  - Если v находится в области 3:  - Это также критическая вершина. Проверяется угол qi−1qiv, который должен быть выпуклым.  - Добавляется v как следующая точка оболочки qi+1 (рисунок 4.11в).  - Если v находится в области 4:  - Граница заходит внутрь оболочки. Продвигаться дальше по границе до тех пор, пока не найдётся вершина в области 2 или 3.  - После чего перейти к соответствующему случаю обработки.  4. Завершение:- Процесс заканчивается, когда достигнута вершина pm.- Результатом является последовательность критических вершин (q1, q2, ..., qr), представляющая верхнюю выпуклую оболочку.  **Примечания:**- Каждое ребро qi qi+1 рассматривается как «крышка» некоторого «кармана» Ki, содержащего оставшуюся часть цепи (рисунок 4.8). - Алгоритм гарантирует, что построенная оболочка согласуется с условием простоты многоугольника. - Корректность алгоритма доказана, а его временная сложность равна O(N), что делает его более эффективным, чем общие методы построения выпуклой оболочки.  **3. Открытый алгоритм** - алгоритм, обрабатывающий данные по мере поступления. Закрытый - алгоритм, обрабатывающий всю совокупность целиком. Основной чертой открытых алгоритмов является отсутствие ограничений на время коррекции, что равносильно тому, что новый элемент данных (точка) вводится по запросу сразу же после того, как завершается коррекция, связанная с предыдущим элементом данных. | **32. Динамические алгоритмы построения выпуклой оболочки. Поддержка динамической выпуклой оболочки :(**  **Построение выпуклой оболочки динамическими алгоритмами:**  **Дано**: множество точек p1,...,p\_n на плоскости.  **Найти:** выпуклую оболочку этого множества так, чтобы обработка происходила по мере поступления каждой новой точки.  **Алгоритмическая сложность:**  – глобальная сложность построения оболочки O(NlogN)  – время коррекции для каждой новой точки O(logN)  Основная идея состоит в использовании открытого алгоритма , который работает с данными пошагово, без предварительного знания всего множества точек. Для этого применяется специальная структура данных — балансированное дерево поиска, которое хранит упорядоченную последовательность вершин выпуклой оболочки.    Используется балансированное дерево поиска (например, AVL-дерево или красно-черное дерево), которое обеспечивает эффективные операции вставки, удаления и поиска с временем O(log i), где i-число узлов дерева. Дерево содержит следующие ключевые элементы:  1. m : Самая левая вершина (начало цепочки).  2. M : Корневая вершина (конец цепочки).  Каждая вершина классифицируется как вогнутая, опорная или выпуклая относительно отрезка между точкой p и вершиной v, что позволяет эффективно определять опорные прямые для каждой новой точки.  Процесс построения:  1. Каждая вершина классифицируется как вогнутая, опорная или выпуклая.  2. Для каждой новой точки p находятся левая и правая опорные точки l и r.  3. Если точка p является внешней, вершины между l и r удаляются, а p вставляется на их место.  **Поддержка динамической выпуклой оболочки:**  **Дано**: множество точек p1,...,p\_n на плоскости.  **Найти:** выпуклую оболочку этого множества так, чтобы обработка происходила после каждой операции вставки или удаления вершины.  **Алгоритмическая сложность:**  – глобальная сложность построения оболочки O(log^2(N))  – время коррекции при добавлении или удалении точки O(logN)  Основная идея состоит в разбиении выпуклой оболочки на монотонные ломаные линии. Всё множество точек делится на подмножества, для каждого подмножества строится своя собственная B-оболочка (часть выпуклой оболочки). Это позволяет не пересчитывать всю оболочку целиком каждый раз, а менять только те её части, которые затронуты добавлением или удалением точки . Эти B-оболочки хранятся в узлах дерева.  **Как устроено дерево:**  Листья дерева содержат отдельные точки множества, упорядоченные по их координате x.  Внутренние узлы представляют собой B-оболочки тех точек, которые находятся в соответствующем поддереве.  То есть: чем выше узел в дереве, тем больше точек он "покрывает", и тем больше часть общей выпуклой оболочки он представляет.  Каждый узел дерева содержит:  Сцепляемую очередь Q[v] — список точек, которые находятся в поддереве этого узла, но не входят в его B-оболочку . То есть, это внутренние точки, которые пока не влияют на внешний вид выпуклой оболочки.  Индекс левой опорной точки J[v] — информация о том, какая точка из B-оболочки текущего узла является левой опорной при соединении с соседней частью оболочки.  Когда мы хотим объединить две части оболочки (например, после удаления новой точки), нам нужно найти между ними опорный отрезок — прямую, которая касается обеих частей и не пересекает их.    Вся оболочка изначально разбивается на верхнюю и нижнюю части. На левом рисунке показано, как эти оболочки хранятся в дереве, тут приведена верхняя оболочка, такая же есть для нижней части. Корень дерева содержит все вершины верхней или нижней части оболочки, его дети только соответствующие её части. |

| 33 Диаграмма Вороного: свойства, алгоритм построения, оценки решения задач близости  **Дано**: конечное множество точек на плоскости.  **Найти**: Границы ячеек Вороного (рёбра и вершины диаграммы).  Диаграмма Вороного конечного множества точек S на плоскости представляет такое разбиение плоскости, при котором каждая область этого разбиения образует множество точек, более близких к одному из элементов множества S, чем к любому другому элементу множества.  Свойства:   * Связь с триангуляцией Делоне. Диаграмма Вороного двойственна триангуляции Делоне. То есть если соединить отрезками середины всех локусов (ячеек Вороного) с центрами соседних локусов, то результатом будет триангуляция Делоне. * При условии, что никакие 4 точки множества S не лежат на одной окружности, каждая вершина диаграммы является пересечением в точности трех ребер диаграммы. * Применяется при поиске всех всех ближайших соседей. Ближайшие соседи всех точек множества S – центры соседних локусов.   Существует множество алгоритмов построения диаграммы. Самый “простой” имеет **сложность О(n4)**. Самые быстрые алгоритмы – алгоритмы Форчуны и Чана имеет **сложность О(n log n).**  “Наивный” метод построения диаграммы:   * Строим серединные перпендикуляры всех отрезков, соединяющих точку p со всеми остальными точками. * Попарно пересекаем все перпендикуляры. * Все точки пересечения проверяем на принадлежность каждой из полуплоскостей точки р. То есть точки, которые принадлежат всем полуплоскостям точки р, объявляем вершинами ее локуса. * Повторяем предыдущие шаги для всех точек.   Полуплоскостью вершины р называется называется полуплоскость, полученная разбиением плоскости серединным перпендикуляром, содержащая точку р.  При наличии на плоскости триангуляции Делоне, диаграмма строится за **О(n)**:  Вершины диаграммы Вороного являются центрами описанных окружностей треугольников триангуляции. | **34 Алгоритмы удаления скрытых линий и поверхностей. Алгоритм плавающего горизонта, алгоритм с использованием Z буфера**  **Постановка задачи**: удаление невидимых линий и поверхностей является одной из наиболее сложных задач в компьютерной графике  **Проблема**: Есть линии которые нам необходимо удалить, и нужно вычислить какие точки видны для наблюдателя а какие нет  **Дано**: сцена, набор точек  **Найти**: точки для удаления  **Суть алгоритмов** - удалить те линии ребер, поверхностей, граней или объемово, которые невидны наблюдателю.  **Идея алгоритмов** - чем дальше расположен объект от точки наблюдения, тем больше вероятность, что он будет полностью или частично заслонен другим, более близким к наблюдателю, объектом. После определения расстояний или приоритетов по глубине проводится сортировка по горизонтали и по вертикали, для выяснения, будет ли рассматриваемый объект действительно заслонен объектом, расположенным ближе к точке наблюдения.  **Алгоритм плавающего горизонта**  Используется для для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде: F(x, у, z) = 0.  **Идея** - сведение трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат х, у или z.  В примере параллельные плоскости определяются постоянными значениями z. F(x, у, z) = 0 сводится к последовательности кривых, лежащих в каждой из этих параллельных плоскостей, например к последовательности функций вида у = f(x, z) где z постоянно на каждой из заданных параллельных плоскостей.  Поверхности складываются последовательности кривых, лежащих в каждой из этих плоскостей  Алгоритм сначала упорядочивает плоскости z = const по возрастанию расстояния до них от точки наблюдения. Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней, т. е. для каждого значения координаты х в пространстве изображения определяется соответствующее значение у. Алгоритм удаления невидимой линии заключается в следующем: если на текущей плоскости при некотором заданном значении х соответствующее значение у на кривой больше значения у для всех предыдущих кривых при этом значении х, то текущая кривая видима в этой точке; в противном случае она невидима  *Сложность от ГПТ* - N \* M (Количество точек по x и по y)  **АЛГОРИТМ, ИСПОЛЬЗУЮЩИЙ z-БУФЕР -** алгоритм удаления невидимых поверхностей  Требует большого объема памяти. Другой недостаток алгоритма z-буфера состоит в трудоемкости и высокой стоимости устранения лестничного эффекта, а также реализации эффектов прозрачности и просвечивания. Поскольку алгоритм заносит пикселы в буфер кадра в произвольном порядке, то нелегко получить информацию, необходимую для методов устранения лестничного эффекта, основывающихся на предварительной фильтрации. При реализации эффектов прозрачности и просвечивания пикселы могут заноситься в буфер кадра в некорректном порядке, что ведет к локальным ошибкам.  **Идея**: Z-буфер – отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пиксела в пространстве изображения. В буфер элементы заносятся в произвольном порядке.  В процессе работы глубина или значение z каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и, кроме того, производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по х и у наибольшего значения функции z (х, у).  **Шаги**:   1. Заполнить буфер кадра фоновым значением интенсивности или цвета 2. Заполнить z-буфер минимальным значением z 3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке. 4. Для каждого Пиксел(x,y) в многоугольнике вычислить его глубину z(x,y). 5. Сравнить глубину z(х,у) со значением Zбуфер(х,у), хранящимся в z-буфере в этой же позиции.   Если z(х,у) > Zбуфер (х,у), то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить Zбуфер(х,у) на z(х,у). В противном случае никаких действий не производить.  **Сложность** - O(W×H), - размер сетки экрана(в пикселях) |
| --- | --- |
| **35.Удаление невидимых линий. Алгоритм Варнока, Алгоритм Вейлера-Айзертона**  Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.  Все алгоритмы удаления невидимых линий включают в себя сортировку. Главная сортировка ведтся по геометрическому расстоянию от тела, поверхности, ребра или точки до точки наблюдения.  **Дано:** 3D сцена, состоящая из многоугольников (aka полигонов, aka 2d поверхностей, в том числе под разными углами и вообще как угодно расположенных в 3d).  **Найти:** изображение сцены с удалением невидимых линий или поверхностей (надо нарисовать только видимые ребра и грани полигонов).  **Алгоритм Варнока**  Работаем в **пространстве изображения** (2d пространство картинки вида на сцену)  Сложность: **O(n log n)**, где n - число многоугольников.  Основная идея: рекурсивно разбивать экран на подокна до достижения простого содержимого или предела разрешения.  Разбили экранч на, например, 4 подокна, для каждого подокна возможна ситуация  1. Все полигоны (а точнее их проекции, мы ведь в пространстве изображения) являются внешними по отношению к окну. Тогда заливаем фоновым цветом и дальше окно не разбиваем.  2. В окно попала проекция или часть проекции только одного полигона. Тогда часть окошка пересекающаяся с проекцией заливаем цветом полигона, а остальную часть фоновым цветом. Дальше окно не разбиваем.  3. Окно охвачено проекцией одного полигона, т. е. окно целиком внутри одной проекции. Заливаем всё окно цветом полигона. Дальше окно не разбиваем.  4. Проекции нескольких полигонов охватывают окно целиком. Выбираем полигон, который расположен ближе других к точке наблюдения (для этого и сортировали полигоны) и заливаем окно его цветом.  5. Во всех остальных случаях разбиваем окно на ещё 4 подокна. Можно разбивать и не на 4 части, можно и не на равные части разбивать, но это уже улучшения.  О критерии, на основании которого прекращать разбиение: В качестве очевидного критерия можно взять размер области. Как только размер области станет не больше размера одного пиксела, то производить дальнейшее разбиение не имеет смысла и для данной области ближайшая к ней грань определяется явно, как в методе трассировки лучей. Иногда, для устранения лестничного эффекта, процесс разбиения проводится до размеров, меньших, чем разрешение экрана, на один пиксель. При этом усредняются атрибуты подпикселей, чтобы определить атрибуты самих пикселей.  **Алгоритм Вейлера-Азертона**  Работаем в **пространстве объекта** (то есть с 3d координатами).  Сложность: **O(n^2)**, где n - число многоугольников.  Применим для удаления как невидимых линий, так и поверхностеи. Не работает с проекциями с самопересечениями (надо предварительно разбивать их на несколько). Вейлер и Азертон попытались минимизировать количество шагов в алгоритме разбиения типа алгоритма Варнока путем **разбиения окна вдоль границ многоугольника**. Выходными данными этого алгоритма, который для достижения необходимой точности работает в пространстве объекта, служат многоугольники. Поскольку выходом являются многоугольники, то алгоритм можно легко использовать для удаления как невидимых линий, так и невидимых поверхностей. Алгоритм удаления невидимых поверхностей состоит из четырех шагов.  1. Предварительная сортировка всех граней по глубине (front-to-back).  2. Отсечение по границе ближайшего к наблюдателю многоугольника, называемое сортировкой многоугольников на плоскости.  3. Удаление многоугольников, экранированных многоугольником, ближайшим к точке  наблюдения.  4. Если требуется, то рекурсивное подразбиение и окончательная сортировка для  устранения всех неопределенностей.  **Альтернативное объяснение** с простым примером: - В качестве первого шага производится сортировка всех граней по глубине (front-to-back).  - Затем из списка оставшихся граней берется ближайшая грань A и все остальные грани обрезаются по этой грани. Если проекция грани B пересекает проекцию грани A, то грань B разбивается на части так, что каждая часть либо содержится в грани A, либо не имеет с ней общих внутренних точек.  - Таким образом, получаются два множества граней: F\_in – грани, проекции которых содержатся в проекции грани A (сюда входит и сама грань A), и F\_out – грани, проекции которых не имеют общих внутренних точек с проекцией грани A. Множество F\_in обычно называют множеством граней, внутренних по отношению к A.  - Однако во множестве Fin могут быть грани, лежащие к наблюдателю ближе, чем сама грань A (это возможно, например, при циклическом наложении граней). В этом случае каждая такая грань используется для разбиения всех граней из множества F\_in (включая исходную грань A). Когда рекурсивное разбиение завершится, то все грани из первого множества выводятся из набора оставшихся граней (их уже ничто не может закрывать). Затем из набора оставшихся граней Fout берется очередная грань и процедура повторяется.  **Простейший случай для двух граней A и B**  Будем считать, что грань A расположена ближе, чем грань B. Тогда на первом шаге для разбиения используется именно грань A. Грань B разбивается на две части. Часть B\_1 попадает в первое множество F\_in и, так как она лежит дальше грани A, удаляется.  После этого выводится грань A и в списке оставшихся граней остается только грань B2. Так как кроме нее других граней не осталось,  то эта грань выводится, и на этом работа завершается. | **36. Удаление невидимых линий, алгоритм Робертса**  Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства  Все алгоритмы удаления невидимых линий/поверхностей обязательно включают в себя этапы:  **1**. Сортировка линий/поверхностей по геометрическому расстоянию от тела, поверхности ребра или  точки до точки наблюдения(чем дальше элемент от наблюдателя, тем больше вероятность, что он  будет перекрыт). Расставляются приоритеты по глубине.  **2**. Сортировка по горизонтали и вертикали для определения элементов тела, которые действительно  друг друга перекрывают.  **Алгоритм плавающего горизонта** Главная идея данного метода заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат *х*, *у* или *z*. Итак, поверхность теперь складывается из последовательности кривых, лежащих в каждой из этих плоскостей, как показано на рис. 5.3.    Алгоритм сначала упорядочивает плоскости *z =* const по возрастанию расстояния до них от точки наблюдения. Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней, т. е. для каждого значения координаты *х* в пространстве изображения определяется соответствующее значение *y*. Алгоритм удаления невидимой линии заключается в следующем.  Если на текущей плоскости при некотором заданном значении *x* соответствующее значение *у* на кривой больше значения *y* для всех предыдущих кривых при этом значении *x*, то текущая кривая видима в этой точке; в противном случае она невидима.  **Алгоритм Робертса**  Алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, экранируются этими телами. Поэтому вычислительная трудоемкость алгоритма Робертса растет теоретически, как квадрат числа объектов.  Дано: Сцена, тела на сцене заданные вершинами, точка наблюдения E.  Важно: В алгоритме Робертса требуется, чтобы все изображенные тела или объекты были выпуклыми. Невыпуклые тела должны быть разбиты на выпуклые.Тело в алгоритме задаётся матрицей тела V, состоящее из коэффициентов уравнения вида ax+by+cz+d=0 плоскостей для каждой плоскости, образующей тело.  |V|= ⎡ a₁ a₂ ⋯ aₙ ⎤ Нормали всех поверхностей должны быть направлены наружу тела (нормаль P(a,b,c),  ⎢ b₁ b₂ ⋯ bₙ ⎥ d отвечает за смещение)  ⎢ c₁ c₂ ⋯ cₙ ⎥ Для проверки используется скалярное произведение *S*=[*x*,*y*,*z*,1] на матрицу  ⎣ d₁ d₂ ⋯ dₙ ⎦ коэффициентов P=[a,b,c,d]   * SPᵀ = 0 (точка на поверхности) * SPᵀ > 0 (точка внутри тела) * SPᵀ < 0 (точка снаружи тела)   При необходимости столбцы матрицы умножаются на -1 для корректировки направления нормалей (берут точку, которая точно внутри, умножают и проверяют, если больше нуля, то этот столбец в матрице умножают на -1)  Для определения нелицевых граней необходимо провести скалярное умножение точки наблюдения с каждой поверхностью, образующей тело. По знаку скалярного произведения можно понять является ли грань лицевой или же её не видно с точки наблюдения. Точка наблюдения *E*, плоскость тела задана матрицей *V*. Если *EV*<0, то грань *V* видима наблюдателю, в обратном случае грань является нелицевой,  Алгоритм проверяет каждое видимое ребро многогранника на экранирование другими телами, используя сортировку по z для оптимизации. Для каждой точки ребра из точки наблюдения проводится отрезок: если он не пересекает многогранники, точка видима. Ребро задаётся параметрически:  P(t) = P\_1 + (P\_2 - P\_1)t,  где t [0,1], S = P\_1, d = P\_2 - P\_1, так что P(t) = S + dt.  Из точки наблюдения g строится плоскость Q(a,t) = S + dt + ga, где a 0. Видимая часть ребра определяется по условию q\_i = (S \* P\_i) + (d \* P\_i)t + (g \* P\_i)a > 0, где P\_i —столбец матрицы. При q\_i = 0 находятся точки смены видимости, обозначающие границы видимых и невидимых участков. |

| **37** [**Алгоритм, использующий список приоритетов. Алгоритм построчного сканирования**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.41mghml)  Алгоритм, использующий список приоритетов:  Цель: Сортировать многоугольники в сцене по их удаленности от точки наблюдения (глубине) для обеспечения правильной заливки и корректного отображения объектов. Это позволяет избежать эффекта "неверного перекрытия" (например, когда более близкие объекты оказываются закрытыми более удалёнными).  Этапы алгоритма   1. Предварительная сортировка:    * Многоугольники сортируются по глубине (расстоянию от точки наблюдения).    * Сортировка выполняется таким образом, чтобы самый дальний многоугольник обрабатывался первым , а самый ближний — последним.    * Этот этап может быть реализован с использованием различных методов сортировки (например, быстрой сортировки или сортировки слиянием), но важно учитывать, что глубина каждого многоугольника вычисляется как расстояние от его центра (или другого характерного точки) до точки наблюдения. 2. Обработка конфликтов между многоугольниками:    * После предварительной сортировки проверяется, не пересекаются ли многоугольники между собой.    * Если два многоугольника *P* и *Q* пересекаются или один полностью перекрывает другой:      + Для каждого многоугольника *P* проверяется соотношение с последующим многоугольником *Q*:        - Полное перекрытие: Один многоугольник полностью находится внутри другого.        - Пересечение: Части многоугольников пересекаются.        - Разделение: Многоугольники не имеют общих точек.      + Если возникает конфликт (например, *P* и *Q* пересекаются или один перекрывает другой), порядок этих многоугольников в списке приоритетов меняется так, чтобы сохранить корректную видимость ближайших объектов.      + Например, если *P* находится ближе к наблюдателю, чем *Q*, но *Q* частично перекрывает *P*, то *P* должен быть отображен раньше *Q*. 3. Пересчет списка приоритетов при наличии конфликтов:    * Если во время проверки обнаруживаются конфликты между многоугольниками, список приоритетов пересчитывается.    * Пересчет может происходить локально (только для конфликтующих пар) или глобально (для всего списка).    * В сложных сценах, где многоугольники часто пересекаются, пересчет списка приоритетов может потребовать значительных вычислительных ресурсов. 4. Обработка сложных сцен:    * Для динамических сцен, где объекты движутся или изменяют форму, список приоритетов должен быть пересчитан для каждого кадра .    * Это гарантирует, что изменения глубины объектов учитываются правильно, и отображение остается корректным.   Алгоритм построчного сканирования (ПС):  Алгоритмы построчного сканирования (ПС): с использованием z-буфера, интервальный алгоритм ПС, алгоритм ПС для криволинейных поверхностей.  Алгоритмы построчного сканирования сводят трехмерную задачу удаления невидимых линий и поверхностей к двумерной. Сканирующая плоскость определяется точкой наблюдения, расположенной в бесконечности на положительной полуоси z, и сканирующей строкой Общий подход алгоритма построчного сканирования:   * + Создается таблица ребер (ТР) :     - Содержит все негоризонтальные ребра многоугольников.     - Ребра сортируются по *y\_*min​ — минимальной *y*-координате ребра.     - Внутри групп ребра сортируются по тангенсу угла наклона (tan(*α*)).     - Данные ребра включают:       * *x*-координату при *y\_*min​,       * *y\_*max​ — максимальная *y*-координата ребра,       * Δ*x*=1/tan(*α*) — приращение *x* для перехода между строками.   + Создается таблица многоугольников (ТМ) :     - Содержит информацию о каждом многоугольнике:       * Коэффициенты уравнения плоскости,       * Цвет или информация о закраске,       * Булев флаг (внутри/вне), изначально установленный в false.   Тут начинается еще более подробно   * + Далее создается таблица активных ребер (ТАР) :     - Упорядочена по координатам *x*.     - Когда алгоритм доходит до сканирующей строки *y*=*α*, ТАР содержит ребра, которые пересекаются с этой строкой.     - Ребра рассматриваются слева направо.     - При обработке ребра меняется булев флаг внутри/вне многоугольника:       * Если флаг становится true, значит мы попали внутрь многоугольника.       * Если флаг становится false, значит мы вышли из многоугольника.   + Обработка завершается, когда последнее ребро многоугольника покинет ТАР.   – Когерентность по глубине:   * + Если при обработке текущей строки в ТАР находятся те же ребра, что и при обработке предыдущей строки, то отношения глубин остаются прежними, и их не нужно пересчитывать. | 38 **Конические сечения. Параметрическое задание окружности, эллипса, параболы, гиперболы. Построение наилучшего приближения с этим кривым**  Общий вид параметрических уравнений: x=x(t); y=y(t). P(t) = [x(t), y(t)]  Проблема: если строить окружность стандартно, используя формулу , имеем проблему неравномерности построения. В первой четверти при равных приращениях отрезки окружности не будут равны.  Параметрическая формула единичной окружности:  Будем использовать приращения углов при построении:    т.к.можно записать рекурсивную формулу:    Параметрическая формула для эллипса:(каноническая в координатах х и у: . Таким образом  Учитывая, формулы для xi и yi:    Параметрическая формула для параболы:  Тогда Реккурентно:    Формула для гиперболы: Параметрически: Используем    (в конспекте тут +в знаменателе, не понимаю поч) |
| --- | --- |
| 39 **Алгоритм определения видимых поверхностей путем трассировки лучей. Интервальный алгоритм построчного сканирования**  Трассировка лучей - метод грубой силы. Метод грубой силы - метод, не учитывающий специфику обрабатываемого объекта.  Главная идея: наблюдатель видит любой объект посредством отраженного света, испускаемого неким источником света, который падает на этот объект и затем каким-то путем доходит до наблюдателя. Свет может достичь наблюдателя, отразившись от поверхности, преломившись или пройдя через неё. Но немногие лучи дойдут до наблюдателя, следовательно, метод вычислительно неэффективен.  Луч проверяется на пересечения с объектами, чтобы определить, какой из них ближе к наблюдателю и, соответственно, видим. Метод требует проверки всех объектов на пересечение с каждым лучом, что делает его вычислительно затратным. Для ускорения используются объемные оболочки (например, сферы или параллелепипеды), позволяющие исключить ненужные проверки.  Тест сферической оболочкой:  Тест сводится к вычислению расстояния от центра сферы до луча. Если квадрат расстояния больше квадрата радиуса оболочки, пересечения с объектом нет. Этот тест прост и быстр в вычислении.  Тест прямоугольной оболочкой:  Требует проверки пересечения луча с тремя параллельными плоскостями. Для ускорения проверяются знаки координат в преобразованной системе. Хотя точнее, такой тест более сложен и медленнее сферического.  Алгоритм трассировки лучей работает следующим образом:   1. Подготовка сцены: формируется список объектов с описанием геометрии, сферической и (при необходимости) прямоугольной оболочки. 2. Для каждого луча:  * Выполняется тест пересечения с сферической оболочкой. Объекты, для которых луч проходит тест, попадают в список активных. * Если активных объектов нет — пиксель окрашивается фоновым цветом. * Иначе луч преобразуется так, чтобы совпадать с осью z.  1. Обработка активных объектов:  * При наличии флага прямоугольной оболочки выполняется ее тест в преобразованной системе координат. * Если оболочка пересекается, объект также преобразуется, и определяется пересечение луча с ним. * Все пересечения сохраняются.  1. Отображение пикселя:  * Если пересечений нет — фоновый цвет. * Иначе определяется ближайшее пересечение (по максимальному z), пересчитывается в исходную систему координат, и пиксель окрашивается в соответствии с атрибутами объекта и моделью освещения.   Построчное интервальное сканирование (интервальный алгоритм) — метод определения видимых поверхностей на сканирующей строке с уменьшением количества вычислений глубины:   1. Разбиение строки на интервалы происходит по точкам пересечения ребер многоугольников. Для каждого интервала возможны три случая:  * Пустой — рисуется фон. * Один отрезок — отображается соответствующий многоугольник. * Несколько отрезков — сравниваются глубины, отображается самый ближний (максимальное z).  1. Глубина:  * Вычисляется в начале, середине или конце интервала в зависимости от взаимного расположения отрезков. * При возможных пересечениях многоугольников глубина определяется в центре интервала.  1. Оптимизация:  * Используются простые разбиения, например, по средней точке.  1. Структура алгоритма:  * Для каждого многоугольника определяется первая сканирующая строка. * Формируется группа объектов для строки. * Сохраняется информация: число строк, пересекающих многоугольник, список его ребер, уравнение плоскости и визуальные атрибуты. | 40 **Построение реалистических изображений. Простая модель освещения для одного источника. Определение нормали к поверхности и вектора отражения**  Простая модель: Свет, может быть отражен, поглощен, пропущен. Объект виден если свет отражается /пропускается. Если поглощает весь свет, то не видим, если нек. длины волн, то объект цветной.  Диффузное отражение:Свет как бы поглощается, а потом заново испускается.Свет точечного источника отражается от идеального рассеивателя по закону косинусов Ламберта  , где I - интенсивность отраженного света, I\_l ​ – интенсивность точечного источника, I\_a - инт. рассеянного света, k\_d ​ – коэффициент диффузного отражения, ka - тоже но для рассеянного ( 0≤ kd,ka ≤ 1), θ – угол между направлением света и нормалью к поверхности. d - расст. от центра проекции до объекта, K - произвольная константа. Если θ>π/2, источник света находится за объектом, и его вклад в освещение равен нулю. Если точечный источник, без учета отражений от других объектов I\_a и k\_a = 0. K - элемент лин. затухания инт. Нужен для того, чтобы интенсивность падала при удалении от источника света. if точка набл. на беск., то d определяется положением объекта ближайшего к точке набл.  Зеркальное отражение: от поверхности. зависит от угла падения, длины волны и св-в вещества. Главное - угол падения = углу отражения, иначе наблюдатель не видит отраженный свет. Св-ва вещества определяют пространственное распределение света. Гладкие - узкий, фокусированное,шероховатые - широкое. В качестве модели - Буи-Туонга Фонга:  , где w(i,л) кривая отражения, пред.отношение зеркально отраженного света к падающему. л - длина волны, i-угола падения n - степень аппроксимации распр. света.Большие n - узкие, менее - широкие. Коэффициент зерк. отражения зависит от угла падения и св-в вещества и длины волны, объединяя диффузное и зеркально отражение получим:  , функция w - сложно, поэтому заменяем на конст k\_s, которую определяем по вайбам. Данная модель - функция закраски. If имеется несколько источников света, то общая модель - сумма: , m - кол-во источников .Заменяя косинусы по векторам можно получить :, где n,L - единичные векторы нормали к поверхности и направления к источнику, R,S - ед векторы - направления отраженного луча и наблюдения.  Нормаль к поверхности: Для объединения с моделью освещения нам надо знать только прибл. знач. нормали на ребрах и в вершинах. Если заданы уравнения плоскостей полигональных граней тогда нормаль к их общей вершине = среднему значению нормали, ко всем многоугольникам сходящимся в этой вершине для примера с картинки: , где abc- коэф уравнений плоскостей трех многоугольников. if нет уравнений - можно уравнивать произведения всех ребер пересекающихся в вершине:  Определение вектора отражения: вектор отражения всегда в одной плоскости с вектором падения  Первый вариант; свет идет по оси z => if перенести начало координат в точку поверхности, то проекция нормали и вектора отражения на плоскость xy будет лежать на одной прямой, тогда: , где R -составляющие ед. векторов отражения, n - нормали. Пусть - угол между нормалью и осью z, тогда: тогда используя, что сумма векторов по всем координатам как R, так и n = 1, получим итоговое выражение  Если свет не по оси z- можно перенести и повернуть нормаль так, чтобы она была параллельна оси z, а точку P принять за начало. Для того, чтобы получить резы в перв. ориентации надо выполнить обратное преобразование: , где L - вектор падения света.  Третий метод: условие того, то вектора падения, отражения и нормаль э одной плоскости пишем в виде векторного произведения, а равенство углов получаем из скалярного произведения: . Векторные произведения равны, если равны их состав.=> СЛАУ. Одно из уравнений в СЛАУ, всегда ЛНЗ => добавим условие nL = nR, получим то, что на пикче. |

| **41. Закраска методом Гуро и методом Фонга**  **Термины для понимания:**  1.1. Полигональная сетка (polygonal mesh) - представление объекта в виде набора многоугольников (обычно треугольников или четырёхугольников).  1.2. Нормаль (normal) - вектор, перпендикулярный поверхности. В компьютерной графике бывает:  -Нормаль к полигону (плоскости)  -Нормаль к вершине (усреднённое значение)  -Нормаль к поверхности (истинная)  1.3. Модели освещения:  -Диффузное (рассеянное) - свет, равномерно отражаемый поверхностью  -Зеркальное - блики от источника света  -Фоновое (ambient) - постоянная подсветка всех объектов  1.4. Полосы Маха - визуальный артефакт, когда на границах полигонов видны полосы из-за резкого изменения интенсивности.  1.5 Интерполяция — это метод нахождения промежуточных значений величины по некоторым известным её значениям.  **1. ОДНОТОННАЯ ЗАКРАСКА**  **Дано:**  -Полигональная модель объекта  -Параметры освещения (позиция источника, интенсивность)  -Модель наблюдателя (позиция камеры)  **Требуется:**  Присвоить всем пикселям каждого полигона одинаковое значение интенсивности. Обеспечить визуальное разделение разных полигонов  **Ограничения:** Подходит только для идеально плоских граней Не может аппроксимировать криволинейные поверхности Даёт "гранёный" вид объектов  **Сложность:** Вычислительная: O(n) где n - количество полигонов Память: минимальные требования Реализация: простейшая из всех методов  **Решение (пошагово):**  1. Для каждого полигона P: a) Вычислить нормаль к плоскости полигона: n = (v₂-v₁)×(v₃-v₁) b) Нормализовать вектор: n̂ = n/||n||  2. Рассчитать освещённость для центра полигона: I = Iₐ + Iₗ(max(0, n̂·l̂)) + Iₛ(max(0, r̂·v̂))^s где:  l̂ - вектор к источнику света  v̂ - вектор к наблюдателю  r̂ = 2(n̂·l̂)n̂ - l̂ (отражённый вектор)  3. Заполнить все пиксели полигона P рассчитанным значением I   | **2. ЗАКРАСКА ГУРО** **Дано:**  Полигональная модель с вершинами V = {v₁, v₂,...}  Возможность вычисления нормалей в вершинах  Параметры освещения и наблюдателя  **Требуется:**  Обеспечить плавные переходы интенсивности между полигонами  Устранить эффект "гранёности"  Сохранить приемлемую скорость вычислений  **Ограничения:**  Линейная интерполяция может давать полосы Маха  Не учитывает нелинейные эффекты освещения  Проблемы с резкими перегибами поверхности  **Сложность:**  Вычислительная: O(n+m) где n - полигоны, m - вершины  Память: требуется хранение нормалей и цветов вершин  Реализация: средняя сложность  **Решение (детализированное):**  1. Фаза предварительных вычислений: a) Для каждой вершины v: i) Найти все полигоны Pᵢ, содержащие v ii) Вычислить усреднённую нормаль: n̂ᵥ = (Σn̂ᵢ)/||Σn̂ᵢ|| iii) Рассчитать интенсивность Iᵥ по модели освещения  2.Растеризация полигона: a) Для каждого ребра e(v₁,v₂): i) Вычислить градиент интенсивности: ΔI = (I₂ - I₁)/(y₂ - y₁) ii) Для каждой строки сканирования y: - Найти x-координаты пересечения - Интерполировать интенсивность: I(y) = I₁ + ΔI\*(y - y₁) b) Для каждой сканирующей строки: i) Найти левое (xₗ, Iₗ) и правое (xᵣ, Iᵣ) пересечения ii) Вычислить горизонтальный градиент: ΔIₕ = (Iᵣ - Iₗ)/(xᵣ - xₗ) iii) Для каждого пикселя (x,y): I(x) = Iₗ + ΔIₕ\*(x - xₗ) Установить цвет пикселя = I(x) | **3. ЗАКРАСКА ФОНГА** **Дано:**  Та же модель, что и для Гуро  Требование высокого качества визуализации  Возможность выполнять больше вычислений  **Требуется:**  Максимально точное воспроизведение освещения  Корректное отображение зеркальных бликов  Устранение всех линейных артефактов  **Ограничения:**  Высокие вычислительные затраты  Требуется интерполяция и нормализация векторов  Сложность аппаратной реализации  **Сложность:**  Вычислительная: O(n\*m) где n - полигоны, m - пиксели  Память: требуется хранение промежуточных нормалей  Реализация: наиболее сложная из трёх  **Решение (полное с формулами):**  1.Предварительная фаза (аналогична Гуро): a) Вычисление усреднённых нормалей вершин  2.Растеризация с интерполяцией нормалей: a) Для каждого ребра e(v₁,v₂): i) Вычислить градиент нормали: Δn = (n̂₂ - n̂₁)/(y₂ - y₁) ii) Для каждой строки сканирования y: n(y) = n̂₁ + Δn\*(y - y₁) Нормализовать: n̂(y) = n(y)/||n(y)|| b) Для каждой сканирующей строки: i) Найти левое (xₗ, n̂ₗ) и правое (xᵣ, n̂ᵣ) пересечения ii) Вычислить горизонтальный градиент нормали: Δnₕ = (n̂ᵣ - n̂ₗ)/(xᵣ - xₗ) iii) Для каждого пикселя (x,y): n(x) = n̂ₗ + Δnₕ\*(x - xₗ) Нормализовать: n̂(x) = n(x)/||n(x)|| Вычислить освещение: I = kₐIₐ + Σ[kₗ(Iₗ(n̂·l̂) + Iₛ(r̂·v̂)^s)] где r̂ = 2(n̂·l̂)n̂ - l̂  3.Оптимизации: a) Линейная аппроксимация нормалей b) Табулирование степенных функций c) Кэширование промежуточных результатов | | --- | --- |   **Выводы:**  - Метод Гуро эффективен и быстр, но имеет ограничения по качеству изображения.  - Метод Фонга требует больше вычислений, но даёт более фотореалистичный результат. | **42 Модель отражения от шероховатых поверхностей Торренса-Спэрроу. Прозрачность, закон Снеллиуса, расчет интенсивности света для многоугольников и криволинейных поверхностей**    Модель отражения **Торренса-Спэрроу** описывает взаимодействие света с шероховатыми поверхностями, учитывая зеркальное и диффузное отражение. Поверхность представляется как совокупность микрограней, ориентированных под разными углами. Отражение зависит от их распределения (функция D), затенения между гранями (G) и коэффициента Френеля (F), который определяет долю отраженного света в зависимости от угла падения и материала.    Модель **Кука-Торрэнса** для m источников:, где слагаемое с индексами a – интенсивность отражения рассеянного света, f – незакрытая часть полусферы, I – интенсивность света, n\*L–проекция нормали поверхности на направление источника света j, dw–дифференциальный элемент телесного угла , который описывает, какую часть полусферы источника света "видит" точка поверхности.  Общий коэффициент отражения вычисляется как сумма зеркальной и диффузной составляющих: r = kd \* rd + ks \* rs, где kd и ks — параметры материала. Интенсивность отраженного света складывается из фонового освещения и вклада каждого источника света, с учётом направления на источник и наблюдателя. rd и rs – коэф. интенсивности.  Для **прозрачных** объектов учитывается преломление света по закону **Снеллиуса**: η1 \* sinθ = η2 \* sinθ', где η — показатель преломления сред, θ — угол падения, θ’ – преломления. Различают зеркальное и диффузное пропускание света. При моделировании криволинейных поверхностей используется нелинейная аппроксимация прозрачности, зависящая от ориентации нормали.  Расчёт интенсивности света для многоугольников и криволинейных поверхностей включает рассеянный, диффузный, зеркальный и пропущенный свет: I = ka \* Ia + kd \* Id + ks \* Is + kt \* It. Для реалистичного отображения необходимы дополнительные данные о прозрачности и глубине объектов. В большинстве моделей предполагается, что k\_i– постоянная, I\_i – интенсивность, определяется по закону Снелиуса  a - рассеянный, d - диффузный, s - зеркальный, t - пропущенный свет. |
| --- | --- | --- | --- |
| 43. Тени, методы построения, алгоритмы Вейлера-Азертона и трассировки лучей. Фактура, нанесение рисунка на поверхность, формирование фрактальной поверхности.  **Дано:**  3D-сцена:  • Параллелепипед P, заданный координатами вершин {vi ∈ R3}^8\_i=1.  • Плоскость H, заданная точкой p\_0 ∈ R3 и нормалью nH ∈ R3, ||nH|| = 1.  • Источник света L, находящийся на бесконечности положительной части оси Z.  • Наблюдатель O, заданный позицией o ∈ R3 и ориентацией (широта ϕ и долгота θ).  Тень — явление, при котором глазу виден наименее освещенный участок. Тень состоит из двух частей – полутени и полной тени. Полная тень – центральная, черная, резко очерченная часть. Полутень – окружающая полную тень более светлая часть.  Метод построения: В трёхмерной графике обычно различают два типа теней:  Собственная тень — та часть объекта, которая сама себя закрывает от света. Проекционная тень — тень, которую один объект отбрасывает на другой.    Для того чтобы построить тени, нужно по существу дважды удалить невидимые поверхности: для положения каждого источника и для положения наблюдателя или точки наблюдения. т. е. это двухшаговый процесс. Алгоритм построения собственных теней аналогичен алгоритму удаления нелицевых граней: грани, затененные собственной тенью являются нелицевыми, если точку наблюдения совместить с источником света. Чтобы найти проекционные тени, нужно построить проекции всех нелицевых граней на сцену. Точки пересечения проецируемой грани со всеми другими плоскостями образуют многоугольники, которые помечаются как теневые многоугольники и заносятся в структуру данных. Для создания разных видов не нужно вычислять тени заново, так как они зависят только от положения источника и не зависят от положения наблюдателя.  **Алгоритм Вейлера-Азертона**  **1.** На первом шаге с помощью алгоритма удаления невидимых поверхностей выделяются освещенные, т. е. видимые из положения источника грани. Для повышения эффективности алгоритма в памяти хранятся именно они, а не грани, лежащие в тени. Освещенные многоугольники помечаются и преобразуются к исходной ориентации, где они приписываются к своим прототипам в качестве многоугольников детализации поверхности. Эта операция выполняется путем присвоения своего номера каждому многоугольнику сцены. В процессе удаления невидимых граней многоугольник может быть разбит на части, которые сохраняют тот же номер. **2.** На втором шаге объединенные данные о многоугольниках обрабатываются из положения наблюдателя. Если какая-либо область не освещена, применяется соответствующее правило расчета интенсивности с учетом затенения. (см. рисунок)  **Алгоритм трассировки лучей**  **1**.На первом, как и в предыдущем случае, трассировкой луча от точки наблюдения через плоскость проекции определяются видимые точки сцены (если таковые есть). **2**. На втором этапе вектор (луч) трассируется от видимой точки до источника света. Если между ними в сцене есть какой-нибудь объект, то свет от источника не попадает в данную точку, т.е. оно оказывается в тени.  **Фактура** – детализация строения поверхности. Виды детализации: 1.На гладкую поверхность наносится заранее заданный узор . После этого поверхность все равно остается гладкой. Наложение узора на гладкую поверхность выполняется с помощью функции отображения. 2. Создание неровностей на поверхности. Такие шероховатые поверхности реализуются путем внесения возмущений в параметры, задающие поверхность.  **Метод для нанесения узора на поверхность.** Главным при нанесении рисунка на поверхность является отображение, поэтому в данном случае задача сводится к преобразованию систем координат. Если рисунок задан в фактурном пространстве в прямоугольной системе координат (u,w), а поверхность — в другой прямоугольной системе координат (θ,φ), то для нанесения рисунка на поверхность нужно найти или задать функцию отображения одного пространства на другое, т.е. θ=f(u,w)φ=g(u,w) или u=r(θ,φ)w=s(θ,φ). Обычно, хотя не обязательно, предполагается, что функция отображения линейна: θ=Au+B, φ=Cw+D, где коэффициенты A, B, C, D выводятся из соотношений между двумя известными точками в системах координат. Короче UV разметк в блендере помните? Вот она..  **Фрактальная поверхность** – это поверхность, которая состоит из случайно заданных полигональных или биполиномиальных поверхностей. Для того чтобы получить полигональную фрактальную поверхность, исходный многоугольник рекурсивно разбивается на фрагменты, как показано на рис. 5.37. Для этого можно, например, случайным образом сместить центр и середины сторон многоугольника, причем и исходный,и полученный многоугольники не обязательно должны быть плоскими. Одно из преимуществ фрактальных поверхностей в том, что их можно разбивать «бесконечно» и получить любой уровень детализации. Он может зависеть от положения наблюдателя: чем ближе точка наблюдения, тем с большей степенью детализации изображается поверхность. Если наблюдатель находится далеко, объем вычислений значительно сокращается.Фрактальная поверхность изображается с помощью любого подходящего алгоритма удаления невидимых поверхностей любой модели освещения. Однако число разбиений возрастает со скоростью выше линейной. | 44[**Глобальная модель освещения с трассировкой лучей, формула вычисления значения наблюдаемой интенсивности. Алгоритм трассировки лучей**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.19c6y18)  Дана: сцена, положения наблюдателя и источников света, тела, модели интенсивности света  Найти: цвет каждого пикселя  Существуют алгоритмы прямой и обратной трассировки лучей.  Алгоритм прямой трассировки лучей.  Есть источник света, из него выпускаются все возможные лучи. Какие то из них отражаются, какие то преломляются, какие то поглощаются. Большинство таких лучей в экран не попадают (не влияют на обзор наблюдателя), при этом отслеживать надо много лучей (особенно если поверхности шероховатые. В этом случае нужны большие вычислительные мощности и это неэффективно.  Алгоритм обратной трассировки лучей  Идея состоит в том, что лучи испускаются не от источника, а от наблюдателя. Будем считать первичным луч V, запущенный от наблюдателя в какую-то точку сцены. Далее рассчитываются лучи, отраженные или преломленные после того как луч V столкнулся с каким-то объектом. Отраженный от объекта А луч попал на объект В. Часть луча прошла через В, а часть отразилась. Можно отобразить структуру таких изменений луча деревом. Ветвь дерева обрывается, если :   * луч выходит за пределы сцены * луч встречается с объектом, от которого не отражается, а поглощается полностью * луч попадает в источник света * интенсивность луча становится настолько мала, что с учетом порога чувствительности невидим   Результирующая световая энергия, попавшая в приемник рассчитывается с учетом всех терминальных вершин дерева с учетом их потерь в оптических средах. То есть начинаем от листьев двигаться вверх по дереву, рассчитывая общую освещенность точки. Освещенность точки складывается из освещенности от всех источников света и из освещенности от всех потомков дерева.  Формула вычисления освещенности при нескольких источниках света: (основная модель, используемая при диффузном отражении) |

| 45 [**Цвет в машинной графике. Основные характеристики света и человеческого восприятия цвета. Аддитивная и субтрактивная системы смешения цветов, законы Грассмана. трехмерное цветовое пространство**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.3tbugp1)  Зрительная система человека воспринимает электромагнитную энергию с длинам волн от 400 до 700 нм как видимый свет. Свет принимается либо непосредственно от источника, либо косвенно при отражении от поверхности объекта или преломлении в нем. Источник или объект является ахроматическим, если наблюдаемый свет содержит все видимые длины волн в приблизительно равных количествах. Ахроматический источник кажется белым, а отраженный или преломленный ахроматический свет — белым, черным или серым. Интенсивность отраженного света удобно рассматривать в диапазоне от 0 до 1, где 0 соответствует черному, 1 — белому, а промежуточные значения — серому цвету. Если воспринимаемый свет содержит длины волн в произвольных неравных количествах, то он называется хроматическим. Цвет хроматического света зависит от доминирующей длины волны в спектре.  **В машинной графике применяются две системы смешения основных цветов:** *аддитивная* — красный, зеленый, синий (RGB) и *субтрактивная* — голубой, пурпурный, желтый (CMY). Цвета одной системы являются дополнительными к другой: голубой – к красному, пурпурный – к зеленому, желтый – к синему. Дополнительный цвет – это разность белого и данного цвета: голубой это белый минус красный, пурпурный – белый минус зеленый, желтый – белый минус синий. (рисунки бесполезны).  Схема rgb используется для светящихся поверхностей(мониторы). Схема cmy используется для печатающих устройств. Смешивать 3 краски для печати, чтобы получить черный, который чаще всего используется для печати, невыгодно-> к этой системе часто добавляют черный цвет. Получается cmyk.  Имея три источника основных цветов системы смешения можно получить все возможные цвета, воспринимаемые человеком. Интенсивность каждой из компонент отображается на некоторый диапазон [0, 1] и кодируется. Таким образом можно представить цвет в памяти ЭВМ и использовать в нуждах компьютерной графики. При этом соотношение между системами RGB и CMY следующее:  **[R, G, B] = [1, 1, 1] − [C, M, Y ]**  **Законы Грассмана**  Когда вы смешиваете два чистых (монохроматических) цвета, то числовые значения Красной (R), Зеленой (G) и Синей (B) составляющих в получившемся смешанном цвете будут просто равны сумме соответствующих значений этих составляющих от каждого из двух исходных чистых цветов.  Законы Грассмана стоят в основе аддитивной модели смешения цветов.  **Трёхмерное цветовое пространство**  Трехмерная природа света позволяет отобразить значение каждого из стимулов на оси ортогональной системы. При этом получается трехкомпонентное цветовое пространство. Любой цвет C можно представить как вектор с составляющими [rR, gG, bB]. Пересечение вектора C с единичной плоскостью R + G + B = 1 даёт относительные веса его красной, зелёной и синей компонент. Они называются координатами цветности. | 46 [**Цветовой график МКО, цветовое пространство XYZ, стандартные источники МКО, цветовые охваты, вычисление координат цветности МКО для смеси двух цветов.**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.28h4qwu)  Трехмерная природа восприятия цвета позволяет отображать его в прямоугольной системе координат. Любой цвет можно изобразить в виде вектора, компонентами которого являются относительные веса красного, зеленого и синего цветов, вычисленные по формулам. r, g, b - трехмерные относительные компоненты, а R, G, B — это абсолютные значения интенсивности (яркости) основных цветов.  r+g+b =1. Ясно, что при таком представлении все множество точек этого треугольника можно описать с помощью двух координат, так как третья выражается через них посредством соотношения b = 1- g- r. Таким образом, мы переходим к двумерному представлению области, т.е. к проекции области на плоскость. С использованием такого преобразования в 1931 г. были выработаны международные стандарты определения и измерения цветов. Основой стандарта стал так называемый двумерный цветовой график МКО. Поскольку, как показали физические эксперименты, сложением трех основных цветов можно получить не все возможные цветовые оттенки, то в качестве базисных были выбраны другие параметры, полученные на основе исследования стандартных реакций глаза на свет. Эти параметры - X,Y Z - являются чисто теоретическими, поскольку построены с использованием отрицательных значений основных составляющих цвета. Треугольник основных цветов был построен так, чтобы охватывать весь спектр видимого света. Кроме того, равное количество всех трех гипотетических цветов в сумме дает белый цвет. Координаты цветности строятся так же, как и в приведенной выше формуле (вместо R, G, B - X, Y,Z). При проекции этого треугольника на плоскость получается цветовой график МКО. Но координаты цветности определяют только относительные количества основных цветов, не задавая яркости результирующего цвета. Яркость можно задать координатой Y, а X, Z определить исходя из величин (x, y, Z), по формулам    (На графике буквы - названия цветов, красный, оранжевый… радуга). На контуре указаны длины волн в нанометрах  Область, ограниченная кривой, охватывает весь видимый спектр, а сама кривая называется линией спектральных цветностей. Числа, проставленные на рисунке, означают длину волны в соответствующей точке.  **Стандартные источники света MKO** Система MKO определяет несколько стандартных источников света с разными цветовыми характеристиками:   * ( A = (0.448, 0.408) ) — тёплый свет, аналогичный излучению лампы накаливания. * ( B = (0.349, 0.352) ) — естественный солнечный свет в полдень. * ( C = (0.310, 0.316) ) — освещение в пасмурный день. * ( D = (0.313, 0.329) ) — соответствует излучению абсолютно чёрного тела с температурой около 6500 K (например, дневной свет с голубоватым оттенком).   **Цветовой охват** Цветовой охват — это область на цветовом графике MKO, которую может воспроизвести устройство (дисплей, принтер, проектор). Он имеет треугольную форму, ограниченную координатами основных цветов устройства. Чем больше эта область, тем шире цветовой охват, что означает способность отображать более насыщенные и разнообразные оттенки. Например, профессиональные мониторы и печатные системы обладают расширенным охватом по сравнению со стандартными устройствами.  Для смешения двух цветов используются законы Грассмана. Пусть два цвета заданы на графике МКО координатами D_1=(x_1,y_1,Y_1) и D_2=(x_2,y_2,Y_2). Тогда смешение их дает цвет D_{12}=(x_1+x_2,y_1+y_2,z_1+z_2). Если ввести обозначения t_1=\frac{Y_1}{y_1}, t_2=\frac{Y_2}{y_2}, то получим координаты цветностиx_{12}=\frac{x_1 t_1 + x_2 t_2}{t_1+t_2}, \quad y_{12}=\frac{y_1 t_1 + y_2 t_2}{t_1+t_2}, \quad Y_{12}=Y_1+Y_2. |
| --- | --- |
| 47 [**Переход от цветового пространства RGB к пространству МКО и обратно. Цветовые системы YIQ, HSV, HLS, цветовой шестигранный конус, алгоритмы преобразования цветового пространства (HSV в RGB, RGB в HSV, HLS в RGB, RGB в HLS)**](https://docs.google.com/document/d/1QAkJbY-iNg2OZxOsYM67m0Svwhly6_7G/edit#heading=h.nmf14n)  Преобразование МКО в RGB  При Cg =Xg +Yg +Zg и Cb =Хb +Yb +Zb уравнение принимает вид:  page152image129288224      Цветовая система YIQ  Цвет представляется как 3 компоненты — яркость (Y) и две искусственные цветоразностные (I и Q). Сигнал I называется синфазным, Q - квадратурным.  **Цветовая система HSV**  HSV (англ. Hue, Saturation, Value — тон, насыщенность, значение) — цветовая модель, в которой координатами цвета являются:  Шкала оттенков — Hue   * Hue — цветовой тон, (например, красный, зелёный или сине-голубой). Варьируется в пределах 0—360°, однако иногда приводится к диапазону 0—100 или 0—1. page154image129354384 * Saturation — насыщенность. Варьируется в пределах 0—100 или 0—1. Чем больше этот параметр, тем «чище» цвет, поэтому этот параметр иногда называют чистотой цвета. А чем ближе этот параметр к нулю, тем ближе цвет к нейтральному серому. * Value (значение цвета). Также задаётся в пределах 0—100 или 0—1.   **Цветовая система HLS**  HSL, HLS (от англ. hue, saturation, lightness)— цветовая модель, в которой цветовыми координатами являются тон, насыщенность и светлота. page154image129354800   | Rgb ->Hsv  page157image129018640 | Hsv->Rgb (0<=(V<=100)  page156image129230800 | | --- | --- | | Rgb -> HLS  page158image129204640   * R, G, B — значения цвета в цветовой модели RGB, значения в диапазоне [0; 1] (R - красный, G - зелёный, B - синий). * ∙MAX — максимум из трёх значений (R, G, B) | HLS-> Rgb  page158image129204432page158image129204432MIN — минимум из трёх значений (R, G, B)   * H—тон[0;360] * S — насыщенность [0; 1] * L — светлота [0; 1] | | 48 **Фракталы. Множество Кантора; фракталы Серпинского; кривая Коха; кривые, заполняющие плоскость; фрактальные кривые и рекурсии.**  **Фрактал - геометрическая фигура, в которой один и тот же фрагмент повторяется при каждом уменьшении масштаба.**  **Множество Кантора** (также известное как **канторово множество** или **канторов дисконтинуум**) — это классический пример множества в математике, которое обладает рядом удивительных свойств.  Множество Кантора строится последовательным удалением частей из единичного отрезка ([0, 1]). Процесс выглядит следующим образом:   1. **Шаг 0**: Начинаем с единичного отрезка (C\_0 = [0, 1]). 2. **Шаг 1**: Удаляем среднюю треть этого отрезка . Остаются два отрезка: 3. **Шаг 2**: Удаляем среднюю треть из каждого из оставшихся отрезков. То есть удаляем интервалы и . Остаются четыре отрезка: 4. **Шаг n**: Продолжаем процесс удаления средних третей из всех оставшихся отрезков на каждом шаге. После n шагов получаем множество C\_n, состоящее из 2^n отрезков длины  каждый.   Множество Кантора: Определяется как пересечение всех множеств (C\_n):  **Треугольник Серпинского** является двумерным аналогом множества Кантора и строится путём рекурсивного удаления подобластей из исходного треугольника.  Построение треугольника Серпинского   1. Берём равносторонний треугольник ( S\_0 ) (можно также использовать другие треугольники). 2. Разделяем его на 4 меньших равносторонних треугольника, соединяя середины сторон, и удаляем центральный.Остаются 3 треугольника: 3. Повторяем процесс для каждого из оставшихся треугольников: 4. На каждом шаге каждый из существующих треугольников делится на 4 части, и центральная удаляется. Количество треугольников растёт как ( 3^n ), а их размер уменьшается как . 5. Треугольник Серпинского: Предельное множество после бесконечного числа итераций:   Ковер Серпинского аналогично, только берется квадрат, делится на 9 частей и вырезается всегда квадрат посередине.  Кривая Коха (последний рисунок) — классический фрактал, описанный шведским математиком Хельге фон Кохом в 1904 году. Это непрерывная, но нигде не дифференцируемая кривая бесконечной длины. Она является одним из первых примеров фракталов, изученных в математике.  Кривая Коха строится итеративно, начиная с простого отрезка: Берём прямолинейный отрезок единичной длины. Делим отрезок на 3 равные части. Среднюю часть заменяем на два отрезка такой же длины, образующих равносторонний треугольник (без основания). Теперь кривая состоит из 4 отрезков длиной 1/3 каждый. K каждому из 4 отрезков применяем ту же процедуру. Получаем 16 отрезков длиной 1/9. На каждом шаге каждый отрезок заменяется на 4 новых, в 3 раза меньших. Число отрезков: ( 4^n ), длина каждого: ( 3^{-n} ). Предельная кривая Коха: Получается при ( n -> infty ).  Крива́я Пеа́но — общее название для параметрических кривых, образ которых содержит квадрат (или, в более общем смысле, открытые области пространства). Другое название — заполняющая пространство кривая.    **Рекурсивные алгоритмы легко позволяют описывать структуры фракталов** |

| 49 **Множества Жюлиа и Мандельброта и их компьютерное построение. Динамические процессы. Бифуркации.**  **Мноожество Мандельброта** — множество точек **c** на комплексной плоскости, для которых рекуррентное соотношение Z\_n+1 = (Z\_n)^2 + **c** при Z\_0 = 0 (или при Z\_0=**c**, одно и то же, ведь Z\_1=**с** при нулевом старте) задаёт ограниченную последовательность. Иными словами, это множество таких c, для которых существует такое действительное R, что неравенство |Z\_n| < R выполняется при всех натуральных n. (ещё раз: каждому комплексному **c**, сопостовляется последовательность комплексных Z\_i, которая может либо расходиться, тогда **c** не будет входить во множество Мандельброта, либо быть ограниченной.)  Вместо арифметических операций с комплексными числами можно моделировать на плоскости ху движение точек Z\_k = [X\_k, Y\_k], задаваемое двумя рекуррентными уравнениями координат: X\_k+1=X\_k^2 - Y\_k^2+**с**\_x, Yk+1=2 \* X\_k \* Y\_k + **с**\_y, где X\_0 = **с**\_x , Y\_0 = **c**\_y. Последовательность Z\_k с ростом числа итераций демонстрирует поведение двух типов: элементы, либо постепенно уходят в бесконечность , либо  всегда остаются в определенной замкнутой области, совершая циклическое движение или сходясь в точку. Строго доказано, что если при некотором K модуль,  | Zk | > rmin , где rmin = 2 - минимальный радиус расходимости множества Мандельброта, то далее последовательность расходится и lim | Zk | = ∞ при к → ∞.  Множество точек **с**, для которых последовательность не расходится, называется множеством Мандельброта. На красивой картинке рисуют именно сами точки **c** (точки в закрашенных областях соответствуют ограниченным последовательностям Z\_k).  **Построение изображения мн-ва Мандельброта и Жюлиа**  1) Каждому пикселю (x, y) сопоставляем комплексное число z\_p (от pixel, это буква **c** в предыдущих обозначениях): например, z\_p = x\_0 + L\_x \* x + i (y\_0 + L\_y \* y), где L\_x и L\_y - коеффициенты для масштабирования (ну картинка 800 на 800px, а нарисовать хотим область от -2 до 1 по x и -1.5 до 1.5 по y например).  2) Для каждого пискеля вычисляем последовательность {z\_i}:  Для Мандельброта: z\_i+1 = z\_i^2 + z\_p, где z\_0 = 0.  Для Жюлиа: z\_i+1 = z\_i^2 + c, где z\_0 = z\_p (тут видим ещё одну ‘c’ (это не **c** из предыдущего раздела), то есть множество Жюлия получается разным для разных комплексных констант c, т.е. мн-в Жулиа бесконечно много, а вот множество Мандельброта лишь одно).  3) Считаем n итераций последовательности (больше n - точнее, но медленнее очевидно)  4) Сохраняем k - номер последней итерации, когда |z\_k| <= 2.  5) Красим пиксель в зависимости от k (например, при k = n, красим в чёрный, т.е. для нашей точности n, компл число соответствующее этому пикселю принадлежит мн-ву, при k << n красим в белый - совсем быстро расходится и точно не принадлежит).  **Динамические процессы** - это просто процессы, которые изменяются во времени по каким-то правилам.  **Бифуркация** - это момент, когда маленькое изменение параметра системы приводит к резкому изменению её поведения.  Пример: логистическое уравнение описывает изменение популяции по формуле:  x\_{n+1} = r \* x\_n \* (1 - x\_n)  где x\_n — размер популяции в текущий момент времени, r — коэффициент рождаемости.  Если параметр r менять от 2 до 4, то поведение системы меняется следующим образом:  - при r от 2 до 3 популяция со временем выходит на устойчивое значение  - при r от 3 до примерно 3.45 система начинает колебаться между двумя значениями  -при дальнейшем увеличении r система колеблется между четырьмя значениями  - после определённого значения r поведение становится хаотичным, возникает хаос  Эти моменты, когда система резко переходит от одного типа поведения к другому, называются бифуркациями. Такое поведение можно наблюдать в биологических, физических, экономических и других реальных системах.  удачи!) |  |
| --- | --- |
|  |  |