

## 1 Чем отличается векторная от растровой графики?

Векторная	Растровая
Рисунок состоит из отдельных сущностей: точек, линий, кривых, многоугольников	Рисунок состоит из сетки пикселей
Масштабирование без потерь качества	Потеря качества при масштабировании
Файлы занимают мало места	Файлы занимают много места
Отсутствие реалистичности	Высокая реалистичность изображения

## 2 Что такое фрактальная графика? Каким образом в ней создаются изображения?

**Фрактальная графика** - вид компьютерной графики. Математическая основа фрактальной графики - фрактальная геометрия. В основу метода построения изображений положен принцип наследования от, так называемых, "родителей" геометрических свойств объектов-наследников. Фрактальная графика является вычисляемой.

Изображение строится по уравнению или системе уравнений. Поэтому в памяти компьютера для выполнения всех вычислений, ничего кроме формул хранить не требуется.

Фрактальная графика незаменима при создании изображений облаков, гор, водных и других поверхностей, очень напоминающих природные неевклидовы поверхности. Достаточно широко фрактальные изображения используются для оформления рекламных листовок, дискотек и веб-сайтов, методами фрактальной графики часто моделируют турбулентные потоки и создают различные узоры.

## 3 Поясните, что такое компьютерная графика (КГ)? Когда и в связи с чем она появилась?

**Компьютерная графика** — раздел информатики, предметом которого является создание и обработка на компьютере графических изображений (рисунков, чертежей, фотографий и пр.)

Компьютерная графика берет свое начало еще с 1961 года, когда С. Рассел, пытался сделать свою первую графику под компьютерную игру и это ему частично удалось.

## 4 Чем отличается двумерная от трёхмерной компьютерной графики?

**Двумерная графика** — это «плоские» изображения, такие как фотографии или рисунки.

**Трёхмерная графика** — это уже работа с объектами в трёхмерном пространстве.

## 5 Что такое синтез, анализ и обработка изображений? Что у них общее и в чём отличия?

**Синтез изображений** — область компьютерной графики, объединяющая методы построения реалистических изображений трехмерного мира, включая математическое моделирование.

**Анализ изображений** — извлечение значимой информации из изображений используя техники обработки цифровых изображений (2D и 3D распознавание объектов, обнаружение движения, сегментация изображения)

**Обработка изображений** — любая форма обработки информации, для которой входные данные представлены изображением, например, фотографиями или видеокадрами.

## 6 Где используются результаты компьютерной графики?

- цифровая графика
- лазерная графика
- цифровая живопись и фотография
- фильмы
- компьютерные игры

## 7 Какие технические средства необходимы для работы приложений компьютерной графики?

Аппаратное обеспечение компьютерной графики:

- Монитор
- Видеокарта
- Принтер
- Плоттер
- Сканер
- Графический планшет
- Мышь

## 8 Поясните, как работают устройства ввода изображений?

**Сканер**

Принцип работы однопроходного планшетного сканера состоит в том, что вдоль сканируемого изображения, расположенного на прозрачном неподвижном стекле, движется сканирующая каретка с источником света. Отражённый свет через оптическую систему сканера (состоящую из объектива и зеркал или призмы) попадает на 3 расположенных параллельно друг другу фоточувствительных полупроводниковых элемента на основе ПЗС, каждый из которых принимает информацию о компонентах изображения.

**Графический планшет**

- Электростатический: регистрируется локальное изменение электрического потенциала сетки под пером
- Электромагнитный: перо излучает волны, а сетка регистрирует
- На электромагнитном резонансе: сетка излучает и принимает сигнал

## **9 Какие устройства можно отнести к группе диалоговых устройств?**

Клавиатуры, сканеры, мониторы, тачпады, принтеры и т.д., и т.п.

## **10 Поясните, каковы принципы работы устройств вывода изображений? Каковы перспективы развития печатающих устройств?**

Принтеры:

- Матричные: используется оттиск печатающей головки (состоит из “иголок”) через красящую ленту
- Струйные: используется выброс микрокапель красителя из печатающей головки (и на основе пьезоэлектрического эффекта)
- Лазерные: используется светочувствительный барабан, на который лазером передается световой импульс. На этих местах остается статический заряд. При прокате барабана через тонер, к заряженным участкам прикрепляется краска. Эту краску переносят на лист бумаги

## **11 Поясните, каким образом появляются изображения на экранах дисплеев?**

Изображение на экране монитора формируется путем считывания содержимого видеопамати и отображения его на экран.

В LCD мониторах молекулы жидких кристаллов под воздействием электрического напряжения могут изменять свою ориентацию и вследствие этого изменять свойства светового луча, проходящего сквозь них.

В ЭЛТ мониторах пучок электрона падает на экран, покрытый люминофором, что вызывает свечение в этом месте. Направлением пучка управляет дисплейный процессор.

## **12 Что такое видеокарта? Как работают современные видеоадаптеры?**

Видеокарта — устройство, преобразующее графический образ, хранящийся как содержимое памяти компьютера (или самого адаптера), в форму, пригодную для дальнейшего вывода на экран монитора.

Центральный процессор, работая совместно с приложениями, передает информацию об изображении на графическую плату. Электроника графической платы принимает решения о том, каким образом использовать экран, чтобы получилось изображение. Затем отправляет эту обработанную информацию по кабелю на монитор.

## **13 Поясните, что в себя включает программное обеспечение компьютерной графики?**

- Графические редакторы (векторные и растровые, 2D и 3D)
- Видеоредакторы
- Форматы файлов
- Конвертеры форматов файлов

## 15 Какую цель ставили перед собой разработчики первых стандартов в КГ?

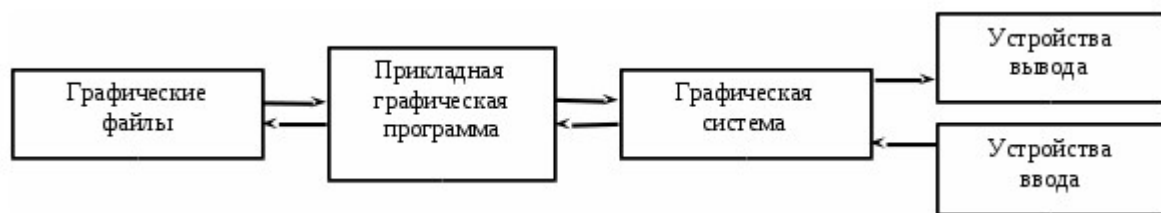
Конференция в Сейлаке, 1976г.

Цель стандартизации: переносимость графических систем. Стандартизация интерфейса между БГС и прикладной программой.

- БГС - Базовая графическая система. Изоляция приложения от графических устройств
- ПВГУ - Протокол виртуального графического устройства. Изоляция БГС от специфики устройств
- ППГИ - Протокол представления графической информации. Изоляция БГС от формата графических данных

1984г. - первый стандарт БГС - GKS (graphics kernel system)

## 16 Нарисуйте, как выглядит схема взаимодействия прикладной программы с графическими устройствами с учётом стандартов на разработку программного обеспечения КГ?



Необходимые исходные данные для визуализации хранятся на компе в виде *графических файлов*. *Прикладная графическая программа* (ППГ) управляет внедрением и извлечением инфы из графических файлов. ППГ передаёт данные и сформированные графические команды в *графическую систему* (ГС). ГС снабжена пакетом подпрограмм вывода (которые управляют конкретными устр-ми вывода (монитор, принтер и т.п.) и обеспечивают визуализацию) и подпрограмм ввода (которые позволяют пользователю динамически управлять содержанием создаваемого на мониторе изображения с помощью уст-в ввода).

## 17-18 Низкоуровневые стандарты

1. Графический интерфейс - его стандартизация (стандартизация методов) доступных программированию к ГУ, она позволяет создавать переносимое программное обеспечение драйверами. Обеспечивает преобразования не зависимо от аппаратуры, сущ. строго регламентированное соглашение взаимодействия м/у CGI и GKS
2. Графический метафайл - его стандартизация позволяет организовать взаимодействие ГС через общие файлы графических данных, физическая среда не имеет значения. Хранит граф инф на ур-не CGI ISO: CGM {Computer Graphics Metafile}; CGI {Computer Graphics Interface}.

Эти стандарты совместимы между собой и учитывают опыт эксплуатации GKS, имеют более широкий набор графических примитивов и некоторые дополнительные возможности в пересылке прямых, блоков пикселей между экраном и памятью, необходимые для мультиплекации, многооконной работе с дисплеями и другие.

Универсальные графические библиотеки Direct3D и OpenGL рассчитаны как на работу с 3х мерными акселераторами, так и без него. В последнем случае соответствующие графические функции эмулируются программно. Фактическими стандартами из них стали Direct3D – ядро поддержки 3х мерной графики, которая используется совместно с DirectDraw ускоренной двумерной графикой и анимацией. Оба эти API – часть API DirectX – набора низкоуровневых библиотек для Windows, которые были созданы фирмой Microsoft для оптимизации работы в Windows различных мультимедийных устройств. Эта библиотека встроена в ядро ОС Windows и является интегрированным сервисом для Win 98, 2000, Internet Explorer. Максимально использует возможности компьютера.

В DirectDraw реализован непосредственный доступ к видеопамяти, чего в самой ОС и API – нет, в отличие от DOS. Поддерживает аппаратную разработку игр и мультимедийных приложений под Windows, где основные графические примитивы реализуются аппаратно. Облегчает создание и анимацию 3х мерных миров, реализуя интерполяторы анимации со смещением цветов, плавными перемещениями объектов и множеством трансформаций, а также последовательное заполнение сеточной модели 3х мерных объектов. Осуществляет z-буферизацию и рендеринг поверхностей, а их непосредственное изображение выполняет DirectDraw, непосредственно обращаясь к видеопамяти и графическим устройствам.

### *Стандарт OpenGL*

Тоже стандарт де факто для всех граф. станции типа SGI, Sun и др. Этот стандарт разработан и утвержден в 1992 (1999) г. ведущими фирмами: IBM, Intel, SGI, Sun, Microsoft. В его основу была положена библиотека IrisGL фирмы SGI, которая насчитывала около 20 функций для задания объектов и операций над ними.

OpenGL – это эффективный аппаратно и платформенно независимый интерфейс. Он позволяет создавать объекты из геометрических примитивов, располагать их в 3х мерном пространстве, выбирая способ и параметры проектирования; вычислять их цвет с учетом источников освещения, параметров освещения и текстур; осуществлять рендеринг создаваемой модели сцены с удаленными невидимыми фрагментами изображения.

Функции OpenGL реализованы по типу клиент-сервер. В данном случае клиент – это приложение, оно вырабатывает команды, а сервер – это библиотека OpenGL, которая интерпретирует и выполняет их, находясь на этом или другом компьютере. OpenGL предоставляет пользователю мощный, но низкоуровневый набор команд и все операции высокого уровня должны выполняться в терминах этих команд. Это является некоторым неудобством для подготовленного пользователя.

Достоинства: стабильность, поскольку библиотека была стандартом профессиональных 3-х мерных приложений долгие годы; переносимость – независимо от ОС все приложения, использующие OpenGL дают одинаковый визуальный эффект или результат на OpenGL совместимом оборудовании; масштабируемость т.е. приложение с OpenGL может выполняться на различных системах от PC до суперкомпьютеров.

## **19 Каким образом можно разложить отрезок в растр?**

В большинстве алгоритмов используется пошаговый метод изображения, т.е. для нахождения координат очередной точки растрового образа наращивается значение одной из координат на единицу растра и вычисляется приращение другой координаты.

Разложить отрезок в растр можно с помощью алгоритмов:

- Алгоритмы Брезенхема
- Простой пошаговый алгоритм

Простейший способ рисования отрезка связан с ур-ем:  $y=kx+b$ , где  $k$ - угловой коэффициент ( $k=(y_k-y_n)/(x_k-x_n)$ ). Если шаг по  $x=1$ , то  $k=dY/dX$  сводится к  $k=dY$ , т.е.изменение  $X$  на единицу приведет к изменению углового коэффициента на  $k$ . А если шаг по  $y=1$ , то  $dX=1/k$ .

## 20 Как повысить эффективность работы базовых алгоритмов разложения отрезка в растр?

### Растр окружности ( $X^2+Y^2=R^2$ )

Процесс можно улучшить, если вычислять одну восьмую часть окружности, а остальные семь частей отображать симметрично (в предыдущем случае  $X$  менять от 0 до  $R/\text{SQRT}(2)$ ). Но необходимый эффект можно получить только при работе с целыми числами.

### Простой пошаговый алгоритм ( $Y=mX+b$ )

Алгоритм корректно работает только для отрезков в первом и восьмом октандах, а в остальных случаях требует модификации. При модификации следует учесть, что при  $m>1$ ,единичный шаг по  $X$  приведет к такому увеличению  $Y$ , при котором две соседние точки на прямой расположатся далеко друг от друга. Поэтому  $X$  и  $Y$  следует поменять, чтобы увеличивать на единицу  $Y$ , а  $X$  - на  $dX = dY/m = 1/m$ .

## 21 Поясните, какова идея Брезенхема и как она реализуется в базовых алгоритмах растрирования отрезков?

**Идея:** алгоритм выбирает оптимальные растровые координаты для представления отрезка.

**Суть алгоритма:** в процессе работы одна из координат либо  $x$ , либо  $y$  (в зависимости от углового коэффициента) изменяется на единицу. Изменение другой координаты (на 0 или 1) зависит от расстояния - ( $e$ ) между действительным положением отрезка и ближайшими координатами растра ( $e$  назовем управляющей переменной). Алгоритм построен так, что на каждом шаге проверяется лишь знак  $e$  и корректируется ее значение после каждого изменения очередной координаты.

Значение исходной управляющей переменной:  $e=2*(y_2-y_1)-(x_2-x_1)$ ,

где  $x_1,y_1,x_2,y_2$  - координаты начальной и конечной точек отрезка. В каждом шаге при  $e \geq 0$  значение  $y$  от предыдущего увеличивается на единицу, а  $e$  уменьшается на  $2*(x_2-x_1)$ , в противном случае -  $y$  не меняется, а значение  $e$  увеличивается на  $2*(y_2-y_1)$ . В обоих случаях координата  $x$  следующего пиксела увеличивается на единицу от предыдущего значения.

## 22 Каким образом можно разложить в растр окружность? Как при этом можно использовать геометрические свойства данного примитива?

Существуют несколько простых способов преобразования окружности в растровую форму. Например, по формуле  $X^2+Y^2=R^2$  для окружности с центром в начале координат. Чтобы изобразить четверть такой окружности, на каждом шаге следует поменять  $X$  от 0 до  $R$  на единицу и вычислить  $Y$  как  $\text{SQRT}(R^2-X^2)$ . Остальные четверти изображают симметрично. Этот метод содержит операции умножения и извлечения корня, потому не эффективен. Кроме того, при  $X$  близких к  $R$ , в окружности появляются заметные промежутки, так как при таких  $X$  тангенс угла наклона касательной к окружности стремится к бесконечности.

## 23 Поясните, какова идея Брезенхема и как она реализуется в базовых алгоритмах растривания окружности и эллипса?

Необходимо сгенерировать только одну восьмую часть окружности. Остальные ее части могут быть получены последовательными отражениями. Пусть мы только что поставили точку  $(x_i, y_i)$  и теперь должны сделать выбор между точками 1  $(x_{i+1}, y_{i-1})$  и 2  $(x_{i+1}, y_i)$ . Алгоритм выбирает пиксел, для которого минимален квадрат расстояния между одним из этих пикселей и окружностью. Алгоритм построения окружности основан на последовательном выборе точек; в зависимости от знака контрольной величины  $D_i$  выбирается следующая точка и нужным образом изменяется сама контрольная величина. Процесс начинается в точке  $(0, r)$ , а первая точка имеет координаты  $(x_c, y_c+r)$ . При  $x = y$  процесс заканчивается.

При  $D^i < 0$   $D^{i+1}$  [при  $y_{i+1} = y_i$ ] =  $D^i + 4x_i + 6$ .

При  $D^i \geq 0$   $D^{i+1}$  [при  $y_{i+1} = y_i - 1$ ] =  $D^i + 4(x_i - y_i) + 10$ .

## 24 Расскажите, как можно закрасить сплошные области? Какие две группы алгоритмов закрашки Вы знаете?

Для закрашивания замкнутых контуров можно использовать несколько методов, которые обычно делятся на две группы:

- заполнение в порядке сканирования строк;
- затравочное заполнение.

При использовании методов первой группы пытаются определить в порядке сканирования строк, лежит ли точка внутри многоугольника или контура. Эти алгоритмы обычно идут от «верха» многоугольника или контура к «низу». Этот метод развертки также применим и к векторным дисплеям, в которых он используется для штриховки или закрашки контуров.

В методах затравочного заполнения предполагается, что известна некоторая точка (затравка) внутри замкнутого контура. В алгоритмах ищут точки, соседние с затравочной и расположенные внутри контура. Если соседняя точка расположена не внутри, значит, обнаружена граница контура. Если же точка оказалась внутри контура, то она становится новой затравочной точкой и поиск продолжается рекурсивно.

## 25 Поясните, в чём идея растровой развёртки полигонов? Как можно эффективно закрасить область внутри треугольника?

Многие замкнутые контуры являются простыми многоугольниками. Если контур состоит из кривых линий, то его можно аппроксимировать подходящим многоугольником или многоугольниками. Простейший метод заполнения многоугольника состоит в проверке на принадлежность внутренности многоугольника каждого пиксела в растре. Этот метод слишком расточителен. Затраты можно уменьшить путем вычисления для многоугольника прямоугольной оболочки - наименьшего многоугольника, содержащего внутри себя многоугольник.

В случае, когда многоугольник, ограничивающий область, задан списком вершин и ребер (ребро определяется как пара вершин), можно предложить еще более экономный метод. Для каждой сканирующей строки определяются точки пересечения с ребрами многогранника, которые затем упорядочиваются по координате  $x$ . Закрашиваются все пикселы между парами точек.

Одним из важнейших алгоритмов растеризации является алгоритм растеризации треугольника (заполнения треугольника), т.к. в большинстве моделей построения трехмерных объектов последние состоят именно из треугольников.



Схематично алгоритм можно описать следующим образом:

1. Треугольник разбивается на две части вертикальной линией, проходящей через среднюю точку:
2. Каждый из двух полученных треугольников растеризуется по отдельности.
3. В процессе растеризации обе стороны растеризуются параллельно, таким образом, чтобы координаты по оси  $y$  текущих точек на обоих отрезках совпадали. При этом связность растеризуемых линий (сторон треугольника) не обеспечивается:
4. При получении координат текущих точек горизонтальный отрезок между ними заполняется.

## **26 Поясните, какова идея заполнения с затравкой? Как можно эффективно залить сплошную область, используя стек?**

Помещаем затравочный пиксель в стек. Далее, пока стек не пуст, будем извлекать из него очередной пиксель и, закрасив его, будем изучать соседние с ним пиксели. Если среди них будут пиксели, не принадлежащие границе и не окрашенные нужным цветом, то поместим их в стек, после этого вернемся к операции извлечения пикселя из стека. По окончании алгоритма все пиксели будут закрашены.

Более эффективный алгоритм:

1. Затравочный пиксел на интервале извлекается из стека, содержащего затравочные пиксели.
2. Интервал с затравочным пикселем заполняется влево и вправо от затравки вдоль сканирующей строки до тех пор, пока не будет найдена граница.
3. В переменных  $X_{\text{лев}}$  и  $X_{\text{прав}}$  запоминаются крайний левый и крайний правый пиксели интервала.
4. В диапазоне  $X_{\text{лев}} \leq x \leq X_{\text{прав}}$  проверяются строки, расположенные непосредственно над и под текущей строкой. Определяется, есть ли на них еще не заполненные пиксели. Если такие пиксели есть (т. е. не все пиксели граничные, или уже заполненные), то в указанном диапазоне крайний правый пиксел в каждом интервале отмечается как затравочный и помещается в стек.
5. При инициализации алгоритма в стек помещается единственный затравочный пиксел, работа завершается при опустошении стека.

## **27 Поясните, для чего необходима процедура отсечения изображения границами окна? Какие идеи реализуются в алгоритме отсечения Сазерленда-Козна?**

Если изображение выходит за пределы экрана, то на части дисплеев увеличивается время построения за счет того, что изображение строится в "уме". В некоторых дисплеях выход за пределы экрана приводит к искажению картины, так как координаты просто ограничиваются при достижении ими граничных значений, а не выполняется точный расчет координат пересечения (эффект "стягивания" изображения). Некоторые, в основном, простые дисплеи просто не допускают выхода за пределы экрана. Все это, особенно в связи с широким использованием технологии просмотра окнами, требует выполнения отсечения сцены по границам окна видимости.

Отсекаемые отрезки могут быть трех классов - целиком видимые, целиком невидимые и пересекающие окно. Очевидно, что целесообразно возможно более рано, без выполнения большого объема вычислений принять решение об видимости целиком или отбрасывании. По способу выбора простого решения об отбрасывании невидимого отрезка целиком или принятия его существует два основных типа алгоритмов отсечения - алгоритмы, использующие



кодирование концов отрезка или всего отрезка и алгоритмы, использующие параметрическое представление отсекаемых отрезков и окна отсеечения. Представители первого типа алгоритмов - алгоритм Козна-Сазерленда(Cohen-Sutherland,CS-алгоритм)иFC-алгоритм(Fast Clipping - алгоритм). Представители алгоритмов второго типа - алгоритмКируса-Бека(Curus-Beck,CB - алгоритм) и более поздний алгоритмЛианга-Барски(Liang-Barsky,LB-алгоритм).

Алгоритмы с кодированием применимы для прямоугольного окна, стороны которого параллельны осям координат, в то время как алгоритмы с параметрическим представлением применимы для произвольного окна.

#### **Алгоритм Козна-Сазерленда.**

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

Этот алгоритм особенно эффективен в двух крайних случаях:

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

Идея алгоритма состоит в следующем:

Окно отсеечения и прилегающие к нему части плоскости вместе образуют 9 областей (рис. 32). Каждой из областей присвоен 4-хразрядный код. Две конечные точки отрезка получают 4-хразрядные коды, соответствующие областям, в которые они попали. Смысл разрядов кода:

- 1 pp = 1 - точка над верхним краем окна;
- 2 pp = 2 - точка под нижним краем окна;
- 3 pp = 4 - точка справа от правого края окна;
- 4 pp = 8 - точка слева от левого края окна.

Определение того лежит ли отрезок целиком внутри окна или целиком вне окна выполняется следующим образом:

- если коды обоих концов отрезка равны 0 то отрезок целиком внутри окна, отсеечение не нужно, отрезок принимается как тривиально видимый;
- если логическое & кодов обоих концов отрезка не равно нулю, то отрезок целиком вне окна, отсеечение не нужно, отрезок отбрасывается как тривиально невидимый;
- если логическое & кодов обоих концов отрезка равно нулю, то отрезок подозрительный, он может быть частично видимым или целиком невидимым; для него нужно определить координаты пересечений со сторонами окна и для каждой полученной части определить тривиальную видимость или невидимость.

## **28-Расскажите, для каких целей следует хранить созданные изображения? Какие проблемы приходится решать при хранении растровых и векторных данных?**

Созданные изображения следует хранить для следующих целей:

- Возможность последующего редактирования изображения
- Передача изображения на другие носители информации

Векторные данные:

содержат описание изображений в виде набора команд построения примитивов. С векторными данными всегда связана информация об атрибутах пикселей или других примитивов и набор правил построения примитивов. Это программно-зависимая информация.

Достоинством данного типа является:

малый объём памяти для хранения, а также возможность редактирования данных, масштабирования объектов и возможность доступа к каждому объекту не зависимо.

Недостатки: время на преобразование файлов для визуализации, не универсальность.

Растровые данные:

содержат набор числовых значений кодов-цветов отдельных пикселей изображения.

Характеризуется битовой глубиной, т.е. кол-ва возможных цветов пикселя.

Достоинством является:

универсальность, простота, распространённость.

Недостатки: огромный размер, который предъявляет специальные требования к процессору и шине передачи данных, потеря качества при масштабировании, сложность редактирования, фиксированное разрешение.

## **29-Для чего нужно сжимать графические данные? Какие алгоритмы сжатия следует использовать? Чем обусловлен их выбор?**

Сжатие графических данных нужно для того, чтобы при сохранении изображение его вес был как можно меньше.

Для хранения изображений уменьшают их объем с помощью сжатия. Методы сжатия основаны на поиске избыточной информации и последующего кодирования этой информации. Алгоритмы сжатия оценивают по - коэффициенту сжатия - скорости упаковки/распаковки.

-Алгоритм Хаффмана: основан на том, что символы заменяются кодовыми последовательностями различной длины, чем чаще встречается символ тем короче его кодовая последовательность. Иначе называют кодированием символами переменной длины. В одних вариантах реализации используются готовые кодовые таблицы (азбука Морзе). В других кодовая таблица строится на основе статистического анализа каждого конкретного файла.

- Алгоритм RLE(Run Lenght Encoding) основан на сжатии последовательностей одинаковых символов. Этот метод прост в реализации, лучше всего работает в составе системы, учитывающей типы данных. Если изображение содержит преимущественно низкочастотный спектр без резкоменяющихся переходов, то сжимается хорошо, иначе плохо.

- Алгоритм JPEG(Joint Photografic Expert Group) – это стандарт для обработки статических изображений. Позволяет достичь высоких коэффициентов сжатия 14 Mb -> 2 Mb. Алгоритм идентифицирует и отбрасывает данные, которые человеческий глаз не в состоянии различить (отдельные пиксели, незначительное изменение цвета и яркости) Алгоритм не подходит для обработки черно-белых и полутоновых изображений, а больше для цветных для вывода на нечувствительное оборудование.

- Алгоритм MPEG определяет метод сжатия, позволяющий свести скорости поступления аудио и видео данных до 1.5 Mbit/сек. Что соответствует скоростям обмена обычных приводов CD-ROM и Data S

## 30. Примеры форматов растровых изображений. Популярные в Internet форматы.

**BMP** — (*Bit Map image*) представляет собой **стандартный растровый формат** и имеет универсальное назначение.

- + поддерживается большинством графических редакторов;
- + почти оптимально подходит для хранения данных и обмена ими;
- занимает слишком много места в памяти;
- не поддерживает анимацию и чередующее отображение.

**TIFF** — (*Taged Image File Format*) – универсальный для издательских систем и топографической графики.

- + обеспечивает высокое качество печати;
- + совместимость со всеми платформами;
- имеет значительные размеры, вследствие чего непопулярен в сети и при создании веб-сайтов;
- непригоден для анимации.

**GIF** — (*Graphic Interchange Format*) служит для хранения **растровых изображений в графике** и для обмена ими.

- + вид картинки не зависит от базовой платформы или типа браузера;
- + сжатие без потерь;
- + имеет небольшой размер, вследствие чего широко применяется при создании HTML-страниц;
- незначительный набор цветов, не поддерживающий плавные переходы цвета.

**JPEG** — (*Joint Photographic Expert Group*) является наиболее распространенным при хранении многоцветных картинок.

- + высокая степень сжатия с низкими потерями данных;
- + небольшой размер;
- + приемлемое качество изображения для публикации в компьютерной сети;
- потери при сжатии;
- возрастание шанса потери данных при каждом повторном сохранении изображения.

**PNG** — (*portable network graphics*) позволяет хранить растровую графику в сжатом виде без потерь, причем файлы получаются меньше по объему, чем в GIF.

- + хранение в сжатом виде без потерь;
- + меньший размер, чем в **GIF**;
- + доступны широкий спектр цветов и прозрачность, дающие широкие возможности в веб-конструировании;
- + кросс-платформенность;
- + поддержка чередующего отображения и анимации.

**ICO** — (*Windows icon*) используется программами для создания картинок малого размера (так называемых «иконок») в браузерах компьютерных систем. Иконками маркируются веб-проекты в строке «Избранное» или URL.

### 31. Примеры форматов векторных изображений. Популярные в Internet форматы.

**AI** - формат программы Adobe Illustrator.

- + очень высокое качество рисунков;
- плохая совместимость с другими программами (например, различные эффекты Illustrator и градиентная заливка могут не передаваться в другие форматы);
- несовместимость с ранними версиями программы Adobe Illustrator.

**CDR** - внутренний формат программы CorelDraw. Имеет преимущества и недостатки, схожие с форматом **AI**.

**DXF** - сложный и многофункциональный формат, обладающий рядом крупных преимуществ над аналогами.

- + поддерживаемость всеми программами автоматизированного проектирования (AutoCAD и др.);
- + возможность хранения трёхмерных объектов;
- + отличный встроенный кодировщик текста;
- из-за сложности формата некоторые приложения не могут сохранять в нём данные.

**EPS** - универсальный векторный формат файлов, поддерживаемый большинством векторных редакторов.

- + совместимость со многими платформами и векторными редакторами;
- + надёжность;
- + большая совокупность настраиваемых параметров;
- + очень высокое качество изображений, популярен в Интернете;
- каждый из редакторов поддерживает формат лишь до определённой версии.

**PDF** - первоначально проектировался как компактный формат электронной документации, но в последнее время все больше используется для передачи по сети графических изображений и смешанных документов, содержащих и текст и графику.

- + независимость внешнего вида документа от платформы;
- + широкие возможности для применения.

**TGA** - ориентирован на аппаратные системы с видеокартами TrueVision, адаптированные к платформам Windows и Macintosh и способные захватывать видеосигналы в стандартах NTSC и PAL, а затем сохранять их в оцифрованном виде. Широко используется в программах редактирования как векторных, так и растровых изображений в тех случаях, когда необходимо сохранить информацию с глубиной 32 разряда на точку.

- + большая популярность в мире анимационной графики и редактирования видеоизображений;
- аппаратная зависимость.

**TIFF** - "теговый изобразительный файл".

- + гибкость записи данных благодаря теговой структуре;
- + возможности для расширения функционала;
- + вся информация хранится в тегах;
- высокая сложность работы с форматом.

**SVG** - формат, представляющий собой открытый стандарт, не являющийся чьей-либо собственностью. Он основан на XML язык разметки и предназначен для описания двумерной векторной графики. SVGZ отличается от SVG сжатием информации при ее сохранении.

- + поддерживается многими браузерами, часто используется для оформления Web-страниц;
- низкое качество в отношении сложных рисунков.

## 32. Поясните, что такое формат метафайла? Какие языки являются типичными носителями данного формата?

**Метафайл** - это общий термин для форматов файлов, которые могут дополнительно хранить в себе скрытые в обычном режиме просмотра дополнительные сведения о хранимых в них данных.

В графике метафайлы содержат дополнительную неграфическую информацию о дате создания, применённых инструментах и их данных, а также водяной знак.

Примеры метафайлов:

- **PICT**;
- **WMF** (Windows Metafile);
- **EPS** (Encapsulated PostScript);
- **CGM** (Computer Graphics Metafile);
- **PDF** (Portable Document Format);
- **CDR** (Corel Draw File);
- **SVG** (Scalable Vector Graphics);
- **WPG** (Word Perfect Graphics File);
- **RTF** (Rich Text Format file).

Язык-носитель формата метафайла - PostScript.

### **33. Поясните, для чего в КГ применяются геометрические преобразования объектов? В каких системах координат они реализуются?**

Геометрические преобразования объектов в КГ используются для:

- операций перемещения, масштабирования и поворота графических объектов и их проекций;
- создания анимации на основе вышеперечисленных действий.

Система координат - совокупность правил, ставящая в соответствие каждому объекту набор координат. Их число называется размерностью пространства.

В КГ применяются следующие системы координат:

- аффинная
- декартова
- полярная
- цилиндрическая
- сферическая

Чаще всего используются аффинная и декартова система координат. Под компьютерной графикой на плоскости, или двумерной компьютерной графикой понимают отображение на экране плоских, то есть двумерных объектов. В двумерной графике нет необходимости в операциях проецирования и наложения теней, так как объект плоский и расположен в одной плоскости – плоскости экрана. К геометрическим преобразованиям плоских объектов относятся сдвиг, поворот, масштабирование и отражение в плоскости экрана. Эти преобразования относятся к так называемым аффинным преобразованиям.

### **34-35 Чем характеризуются аффинные преобразования? Какие аффинные преобразования составляют частные случаи? Как они используются для организации движения объектов изображений?**

В компьютерной графике чаще всего используют аффинную и декартовую систему координат. Это прямоугольная координатная система, в которой выбирается точка  $O$  - начало координат и два приложенных к ней неколлинеарных единичных вектора  $e_x$  и  $e_y$ , которые задают оси координат. Если единичные отрезки на осях не равны, система называется аффинной, если равны и угол между осями координат прямой - прямоугольной декартовой.

Однородным представлением  $n$ -мерного объекта является его представление в  $(n+1)$ -мерном пространстве, полученное добавлением еще одной координаты - скалярного множителя. При решении задач компьютерной графики однородные координаты обычно вводятся так: произвольной точке  $M(x, y)$  на плоскости ставится в соответствие точка  $M(x, y, 1)$  в пространстве. Если нам необходимо преобразовать точку на плоскости с координатами  $(x, y)$  в другую точку то задача сводится к поиску новых координат для этой точки -  $(x', y')$ . В случае аффинных преобразований такой поиск сведется к решению уравнений

$$x' = ax + by + m$$

$$y' = cx + dy + n,$$

где  $a, b, c, d, m, n$  - произвольные числа, причем:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \approx 0$$

В случае, когда  $m$  и  $n$  не равны нулю, для представления преобразования в матричной форме нужно исходные и преобразованные координаты точки записать в однородных координатах  $(x, y, 1)$  и  $(x', y', 1)$ .

Тогда в матричной форме общий вид преобразования будет следующим

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} * \begin{bmatrix} a & c & 0 \\ b & d & 0 \\ m & n & 1 \end{bmatrix} = \begin{bmatrix} a * x + b * y + m & c * x + d * y + n & 1 \end{bmatrix}$$

Наибольшее распространение получили частные случаи аффинных преобразований:

- Единичное преобразование. Единичная матрица оставляет точку неподвижной.
- Сдвиг или плоскопараллельный перенос. Матрица переводит точку на  $m$  единиц вдоль оси  $x$  и на  $n$  –вдоль оси  $y$ .
- Вращение вокруг начала координат. Его матрица осуществляет поворот точки объекта на угол  $g$  против часовой стрелки. Вращение вокруг произвольного центра осуществляет поворот вокруг точки  $(m, n)$  на угол  $g$  против часовой стрелки. Преобразование выполняется как последовательность трех элементарных:
  - -сдвиг центра вращения  $(m, n)$  в начало координат с помощью матрицы сдвига
  - -поворот на угол  $g$  вокруг начала координат с помощью матрицы вращения
  - -сдвиг точки  $(m, n)$  в исходное положение, используя матрицу сдвига
- Симметрия относительно оси, проходящей через начало координат. При этом, если угол между осью симметрии и осью  $Ox = w$ , то угол  $g = 2 * w$ .
- Масштабирование – увеличение (уменьшение размеров изображения) - в общем случае изменяет форму объекта. Назначается точка, относительно которой производится преобразование. где  $k_x, k_y$  - коэффициенты искажения по осям  $Ox, Oy$  соответственно. При  $k_x = k_y = k$  осуществляется преобразование подобия, при  $k_x \neq k_y$  изображение искажается. Изображение увеличивается при  $k > 1$  и уменьшается при  $k < 1$ .
- Масштабирование относительно произвольной точки с координатами  $(m, n)$

Матрица любого аффинного преобразования может быть получена умножением соответствующих рассмотренных здесь простых матриц. Порядок умножения имеет значение, поэтому выполнять его надо в определенной логической последовательности.

Если координаты точек объекта представлены вектором - столбцом, а не вектором - строкой, соответствующая матрица для преобразований должна быть транспонирована. Для возвращения к исходному состоянию следует использовать матрицу обратную той, что использовалась при преобразовании.

Для эффективной работы с преобразованиями следует руководствоваться следующими рекомендациями:

- лучше умножение на результирующую матрицу, чем последовательность умножений
- лучше масштабировать, а потом поворачивать, чем поворачивать, а затем масштабировать
- если комбинированное преобразование содержит поворот, то его следует делать отдельно и последним
- для того, чтобы движение казалось непрерывным и плавным, следует выводить кадры достаточно быстро (30-60 мсек) и координаты каждой точки преобразовывать быстрее.



## 36-37      Аппаратные и программные средства формирования динамических графических изображений.

Спрайт (32x32) *pic*, который может перемещаться по экрану по заданной траектории независимо от остального изображения сцены (фона). Граф. изображение спрайта м. б. растровым (векторным, прямоугольным, другой геометр. формы), перемещение спрайта дискретно. Через некоторый интервал времени  $\Delta t$  объект перемещается на фиксированный шаг  $\Delta L$ , в данном направлении, чем эти дельты меньше, тем более плавно движется объект по экрану. Для имитации реального движения период изменения формы объекта разбивается на несколько частей- фаз и для каждой из них строится своё изображение, свой спрайт, можно использовать зеркальное изображение фаз.

Граф. изображение каждой фазы движения можно формировать различными способами.

Порядок создания анимации с использованием XOR:

Вывод спрайта по XOR два раза подряд за один шаг первый раз появляется искомое изображение, второй раз оно исчезает, фон восстановлен. Достоинства- быстрота, простота, не нужен дополнительной памяти для сохранения фона. Недостатки –искаженное изображение, мерцание, т. к. существует момент, когда объект исчезает с экрана. Используется данный метод для вывода контурных изображений в Windows для изображения рамок, окон.

Для практического применения используются другие способы. Чтобы изображение спрайта не искажалось, нужно хранить фон над спрайтом, а, чтобы не было мерцания, изображения фона с шагом перемещения в буфере, в ОП, или в активной видеостранице на сохранённый фон накладывается спрайт и затем посылается в видеопамять во время обратного хода луча. Фон нужно хранить  $H*(W*S)$ .

Поскольку спрайт обычно не прямоугольный, наложение его на фон не может выполняться простым замещением. Для этого используется прозрачность, т.е. каждому значению атрибута *pix* спрайта добавляется оверлейный (лежащий сверху) бит, в котором указывается выводить или не выводить данный пиксель, переключая этот бит можно наложить два изображения. При наложении объекта на фон существует ряд проблем - границы объекта - не прямоугольник, объект имеет в себе полости (объект-кольцо), которые решаются 2-я способами:

- изображение в 2-х видах: собственно, изображение и битовая маска;
- битовая маска встраивается в цветовую палитру.

При наложении на фон точка объекта накладывается, если в маске установлен бит (данную операцию выполняет Win-драйвер). Данные методы объединяют цветовую гамму.

## 38. Для чего нужны проекции в компьютерной графике? Какие виды проецирования используются в ней чаще всего?

Изображение объектов на экране связано с такой геометрической операцией – *проецированием*.

В компьютерной графике используют в основном *параллельное* и *центральное* проецирование прямыми лучами на плоскость. *Параллельное* проецирование предполагает наличие: центра проецирования в бесконечности, вектора проецирования и проецирующих лучей, параллельных данному вектору, проецирующей (картинной) плоскости. При *центральной* проецировании явно задаются: координаты точки – центра проецирования, картинная плоскость. Центральное проецирование порождает визуальный эффект, аналогичный зрительному – перспективное укорачивание: с увеличением расстояния от центра проекции до объекта его размеры

уменьшаются. Поэтому оно используется для создания реальных картин, но непригодно для представления точной формы и размеров объектов проецирования, необходимое, например, в чертежных задачах конструкторской графики. Параллельное проецирование дает менее реалистичное изображение (нет перспективного укорачивания), но предоставляет пользователю истинные с точностью до скалярного множителя размеры. Для описания преобразований проецирования используются однородные координаты и матрицы четвертого порядка, что упрощает изложение материала и облегчает решение задач геометрического моделирования.

### 39. Каким образом можно получить ортографические проекции объектов в компьютерной графике?

Самой простой из параллельных проекций является ортографическая проекция, используемая обычно в инженерных чертежах. В этом случае точно изображаются правильные или «истинные» размер и форма одной плоской грани объекта. Ортографические проекции - это проекции на одну из координатных плоскостей  $x = 0$ ,  $y = 0$  или  $z = 0$ . Матрица проекции на плоскость  $z = 0$  имеет вид

$$[P_z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \text{ Заметим, что в третьем столбце (столбце } z) \text{ все элементы нулевые.}$$

Следовательно, в результате преобразования  $z$ -координата координатного вектора станет равной нулю. Аналогичным образом матрицы для проекций на плоскости  $x = 0$  и  $y = 0$  равны  $[P_x] =$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ и } [P_y] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

### 40. Какой математический аппарат следует использовать для получения аксонометрических прямоугольных проекций?

Аксонометрические прямоугольные параллельные проекции – проецирующие прямые перпендикулярны картинной плоскости, которая не совпадает (не параллельна) ни с одной из

координатных плоскостей. Общий вид матрицы таких проекций:

$$\begin{bmatrix} \cos p & \sin f * \sin p & 0 & 0 \\ 0 & \cos f & 0 & 0 \\ \sin p & \sin f * \cos p & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

где  $p$  и  $f$  - углы, которые нормаль к картинной плоскости образует с осями координатных осей (соответственно  $OY$  и  $OX$ ). Для построения стандартной изометрии следует взять  $p$  равным  $45^\circ$ ,  $f$  -  $35.264^\circ$  градусам. Для стандартной диметрии:  $p=22.208^\circ$ ,  $f=20.705^\circ$  градусов. При других значениях углов получается триметрию. Значения углов в матрицу подставляются в радианах.

#### 41. Какой математический аппарат следует использовать для получения аксонометрических косоугольных проекций?

Косоугольная параллельная аксонометрия – проекторы не перпендикулярны картинной плоскости, которая совпадает (параллельна) с одной из координатных плоскостей. Самые простые и наглядные из косоугольных – фронтальные проекции (картинная плоскость параллельна XOZ). Из них – косоугольная фронтальная диметрия (кабине) и косоугольная фронтальная изометрия

(кавалье). Матрицы для этих двух проекций выглядят так: 
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ l \cos \frac{\pi}{4} & l \sin \frac{\pi}{4} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ где } l = 1 \text{ для}$$

кавалье и 0.5 для кабине. Для получения координат проекции любой точки изображения необходимо исходные координаты этой точки перемножить с соответствующей матрицей.

Например, для получения проекции куба на экране, необходимо найти новые координаты восьми точек – вершин куба, затем соединить их отрезками в определенной последовательности.

Процедура нахождения новых координат проекции кавалье, например, будет выглядеть так:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ l \cos \frac{\pi}{4} & l \sin \frac{\pi}{4} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} == \begin{bmatrix} x + z * l \cos \frac{\pi}{4} & y + z * l \sin \frac{\pi}{4} & z & 1 \end{bmatrix}$$
$$\begin{cases} x' = x + z * l * 0.707 \\ y' = y + z * l * 0.707 \end{cases}$$

#### 42. Что такое центральная проекция? Каким математическим аппаратом следует воспользоваться для получения центральных проекций объектов?

Если центр проекции находится на конечном расстоянии от проекционной плоскости, то проекция – центральная. Центральная проекция приводит к визуальному эффекту перспективного укорачивания, когда размер проекции объекта изменяется обратно пропорционально расстоянию от центра проекции до объекта. Центральная проекция любой совокупности параллельных прямых, которые не параллельны проекционной плоскости, будет сходиться в точке схода. Точек схода бесконечно много. Если совокупность прямых параллельна одной из главных координатных осей, то их точка схода называется главной точкой схода. В зависимости от того, сколько координатных осей пересекает проекционную плоскость, различают три вида проекций:

1. Одноточечная (имеющие одну точку схода) центральные проекции характеризуются следующим: плоскость проекции совпадает с координатной  $Z = 0$ , центр

проекции имеет координаты  $(0, 0, -d)$ . Матрица проецирования имеет вид 
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Для получения проекции точки в пространстве с координатами

$(x, y, z, 1)$  необходимо найти ее новые однородные, а затем – новые координаты  $(x', y')$  так

$$\begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix} == \begin{bmatrix} x & y & 0 & \frac{z}{d} + 1 \end{bmatrix}$$

$$\begin{cases} x' = x / (\frac{z}{d} + 1) \\ y' = y / (\frac{z}{d} + 1) \end{cases}$$

2. Двухточечная (широко применяется в архитектурном, инженерном и промышленном проектировании).

3. Трехточечная центральная (практически не используется).

## 43 Для чего в компьютерной графике используется цвет? Каким образом он описывается?

Сейчас цвет используется в основном, как средство усиления зрительного впечатления и увеличения информационной насыщенности изображения. Физически цвет – это видимый свет, т.е. электромагнитная энергия с длинами волн от 380 нм (фиолетовый) до 770 нм (красный). Ощущение цвета возникает у человека в результате анализа светового потока, падающего на сетчатку глаза от излучающих или отражающих объектов.

Излучаемый свет идет непосредственно от источника к глазу, сохраняя свои характеристики. Излучаемый свет (от солнца, лампы или монитора) может быть белым (ахроматическим) или любого цвета (с фильтром).

Отраженный свет (от бумаги, дерева, металла) зависит от цвета источника и физических свойств объекта. Объект кажется цветным, если отражает или пропускает свет в узком диапазоне длин волн. Белый цвет (ахроматический) содержит все видимые длины волн в равных количествах. Объект считается белым, если отражает более 80 % белого света и ахроматически черным, если <3%. Единственным атрибутом белого цвета является его интенсивность, т.е. количество отраженного света от 0 до 1.

Хроматический цвет объективно имеет следующие характеристики:

- доминирующую длину волны  $\lambda$
- чистота (степень разбавления белым)
- яркость (кол-во света на ед. площади)

Субъективно данным характеристикам соответствуют:

- доминирующей длине волны  $\lambda$  - цветовой тон (Hue (Value))
- чистоте – насыщенность (Saturation)
- яркости – светлота (lightness, lumination)

Минимальная разница между яркостью различимых по светлоте объектов – это порог, его величина пропорциональна логарифму отношения яркости.  $I_0 = I_0$ ;  $I_1 = I_0 \cdot r$ ;  $I_2 = I_0 \cdot r^2$ ; ...  $I_{225} = r^{225} \cdot I_0$

Характеризующие параметры цвета.

*Цветовой тон* (собственно цвет). Цветовые тона или спектральные цвета располагаются на цветовом круге. Цветовой тон характеризуется положением на цветовом круге и определяется величиной угла в диапазоне от 0 до 360 градусов. Эти цвета обладают максимальной насыщенностью и максимальной яркостью.

*Насыщенность* (процент добавления к цвету белой краски) — это параметр цвета, определяющий его чистоту. Если по краю цветового круга располагаются максимально насыщенные цвета (100%), то остается только уменьшать их насыщенность до минимума (0%). Цвет с уменьшением насыщенности осветляется, как будто к нему прибавляют белую краску. При значении насыщенности 0% любой цвет становится белым.

*Яркость* (процент добавления черной краски) — это параметр цвета, определяющий освещенность или затемненность цвета. Все цвета рассмотренного выше цветового круга имеют максимальную яркость (100%) и ярче уже быть не могут. Яркость можно уменьшить до минимума (0%). Уменьшение яркости цвета означает его зачернение. Работу с яркостью можно характеризовать как добавление в спектральный цвет определенного процента черной краски. В общем случае, любой цвет получается из спектрального цвета добавлением определенного процента белой и черной красок, то есть фактически серой краски.

*Цветовая модель* Пантон, система PMS (Pantone Matching System) — стандартизованная система подбора цвета, разработанная американской фирмой Pantone Inc в середине XX века. Использует цифровую идентификацию цветов изображения для полиграфии печати как смесевыми, так и триадными красками. Эталонные пронумерованные цвета напечатаны в специальной книге, страницы которой веерообразно раскладываются.

Существует множество каталогов образцов цветов Pantone, каждый из которых рассчитан на определённые условия печати. Например, для печати на мелованной, немелованной бумаге, каталог для металлизированных красок (золотая, серебряная) и т. д. Производитель настаивает на том, что «веера» необходимо ежегодно заменять, так как за это время процесс выцветания и истирания изображения делает цвета неточными.

## 44 Почему цвет в компьютерной графике можно описать тремя компонентами? Поясните, в чём разница между аддитивными и субтрактивными цветовыми моделями?

Исследования показали, что чел. глаз восприимчив к цвету. На экране по 1 байту на компоненту (RGB) мы можем различить.

Аддитивные (addition-сложение)

Получаются сложением разных цветов (в природе: естественный белый цвет – сложение 7 цветов радуги). Однако ощущение белого цвета дает смешение любых 3-х цветов, если и 1 из них не является линейной комбинацией 2-х остальных. Такие 3 называют основными и обычно это красный, синий, зеленый (RGB). Присутствие 3-х основных цветов в одинаковых количествах дает белый, отсутствие – черный, а соединение в различных пропорциях – любой другой из

возможного множества. Система аддитивных цветов работает с излучаемым светом (например, от монитора). На мониторе применяется эта модель.

Субтрактивные (subtraction - вычитание).

Цвета получают вычитанием цветов из общего луча (т.е. белого). Наоборот, белый цвет – отсутствие всех цветов, черный – присутствие всех цветов в одинаковых количествах. Работает с отраженным светом (например, от листа бумаги). Чаще всего в качестве основных берут голубой, пурпурный, желтый (СМУ), которые являются противоположными (дополняющими) к RGB. Каждый из основных цветов субтрактивного цвета поглощает (или вычитает) определенный цвет из белого, падающего на страницу (лист бумаги).

## 45-48 Цветовые модели в КГ

Их назначение – дать возможность удобным образом описывать цвета в пределах некоторого цветового охвата.

### RGB

Используется в телевизионных мониторах, растровых дисплеях, цветных струйных принтерах. Такая модель отлично подходит для моделирования цвета от направленных источников (т.е. каждый из основных цветов можно обрабатывать отдельно). Идеально подходит к адаптерам с 2 и 3 байтами на пиксель, т.е. именно она реализована в этих режимах аппаратно. Например, самый яркий синий в данной модели получится  $0,0R+0,0G+1,0B$ ; салатовый:  $0,0R+0,5G+0,5B$ ; оттенок серого в 4 раза меньшей чем самый белый:  $0,25R+0,25G+0,25B$ . Самая распространенная система, т.к. аппаратно реализована в средствах, а они явно или неявно влияют на процесс и характер работы с цветом. В этом и недостаток данной системы: она не подходит для печати, она не очень удобна при практическом получении цвета.

### СМУК – субтрактивная

Моделирует отраженный цвет. В качестве основных использует: голубой (Cyan, синезеленый, бирюзовый)- дополнительный к R, пурпурный (Magenta)- к G, желтый (Yellow)- к B, черный (black). В модели СМУ при освещении каждым из 3-х основных цветов поглощается дополняющий его цвет. Например, если увеличить количество зеленой краски, то интенсивность синего цвета в изображении уменьшится. Новые цвета в модели получают вычитанием цветовых составляющих из белого цвета. Они имеют длину волн отраженного света, не поглощенного основными цветами СМУ. (Составляющие задаются в процентном соотношении – 0-100%) Черный должен получаться смешением C+M+Y, но на не очень белой бумаге и при наличии красок, не полностью поглощающих цвет (не идеальных), получается не черный цвет, а темно-коричневый или грязно-серый. Поэтому вынужденно добавляется краска черного цвета. Система СМУК была известна еще давно. Ее использовали для разделения цветов при печати изображения и еще когда применялись фотомеханические способы. Сейчас большую часть по цветоделению выполняют компьютеры. Процесс 4х цветной печати состоит из 2х этапов: цветоделение исходного рисунка на 4 составляющих (голубой, пурпурный, желтый, черный) и печать каждого из полученных изображений последовательно на одном листе бумаги.

Связь моделей RGB и СМУ простая (сумма смежных = цвет в центре). Перевод:  $(СМУ) = (111)-(RGB)$ ;  $(RGB) = (111) - (СМУ)$ . При использовании еще одной модели или СМУК процесс цветоделения усложняется. Вместо сплошных цветных областей программа цветоделения создает растры отдельных точек, слегка повернутых друг относительно друга. Повороты нужны, чтобы точки разных цветов не накладывались одна на другую, а располагались рядом. Поскольку в системах RGB и СМУК разная природа получения цветов, цвет на мониторе трудно, а иногда и невозможно повторить на печати. Это происходит потому, что печатная машина физически не может отобразить цвет, в котором менее 7% краски. Кроме того, преобразование RGB – СМУК



неизбежно вносит искажения. На передачу цвета влияют состав и качество бумаги, красок. В связи с этим цветовой диапазон модели CMYK меньше чем RGB.

HSV (HSB) Система RGB и CMYK базируется на ограничениях связанных с аппаратным обеспечением. В отличие от них система HSB (тон, яркость, насыщенность) является более интуитивной и приближенной к человеку. Здесь тон или оттенок – это цвет в общем употребительном смысле, насыщенность – цветность или количество белого в тоне т.е. дает частоту цвета; яркость – интенсивность свечения зависит от количества черной краски: чем ее меньше, тем цвет ярче. Данная модель похожа на те, что применяют художники. Измерять цвет можно процентным содержанием чистого тона и добавок, их смешение дает результирующий цвет H – чистый пигмент, S – количество белого, V – количество черного. Сущ. несколько цветовых моделей подобных этой, в которых цвет тона моделируется по средствам изменения оттенка двумя другими составляющими HSI (тон, насыщенность, интенсивность) HSL (тон, насыщенность, освещение) HSB (тон, яркость, освещение) Эта цветовая модель является наиболее простой для понимания. Кроме того, она равно применима и для аддитивных, и для субстративных цветов.

HSB — это трехканальная модель цвета. Она получила название по первым буквам английских слов: цветовой тон (hue), насыщенность (saturation), яркость (brightness).

## 49-На что ориентируются цветовые модели типа LAB?

Lab — аббревиатура названия двух разных (хотя и похожих) цветовых пространств. При разработке Lab преследовалась цель создания цветового пространства, изменение цвета в котором будет более линейным с точки зрения человеческого восприятия (по сравнению с XYZ), то есть с тем, чтобы одинаковое изменение значений координат цвета в разных областях цветового пространства производило одинаковое ощущение изменения цвета. Таким образом математически корректировалась бы нелинейность восприятия цвета человеком. Оба цветовых пространства рассчитываются относительно определенного значения точки белого. Если значение точки белого дополнительно не указывается, подразумевается, что значения Lab рассчитаны для стандартного осветителя D50. В цветовом пространстве Lab значение светлоты отделено от значения хроматической составляющей цвета (тон, насыщенность). Светлота задана координатой L (изменяется от 0 до 100, то есть от самого темного до самого светлого), хроматическая составляющая — двумя декартовыми координатами a и b. Первая обозначает положение цвета в диапазоне от зеленого до красного, вторая — от синего до желтого. В отличие от цветовых пространств RGB или CMYK, которые являются, по сути, набором аппаратных данных для воспроизведения цвета на бумаге или на экране монитора (цвет может зависеть от типа печатной машины, марки красок, влажности воздуха в цеху или производителя монитора и его настроек), Lab однозначно определяет цвет. Поэтому Lab нашел широкое применение в программном обеспечении для обработки изображений в качестве промежуточного цветового пространства, через которое происходит конвертирование данных между другими цветовыми пространствами (например, из RGB сканера в CMYK печатного процесса). При этом особые свойства Lab сделали редактирование в этом пространстве мощным инструментом цвета коррекции. Благодаря характеру определения цвета в Lab появляется возможность отдельно воздействовать на яркость, контраст изображения и на его цвет. Во многих случаях это позволяет ускорить обработку изображений, например, при допечатной подготовке. Lab предоставляет возможность избирательного воздействия на отдельные цвета в изображении, усиления цветового контраста, незаменимыми являются и возможности, которые это цветовое пространство предоставляет для борьбы с шумом на цифровых фотографиях

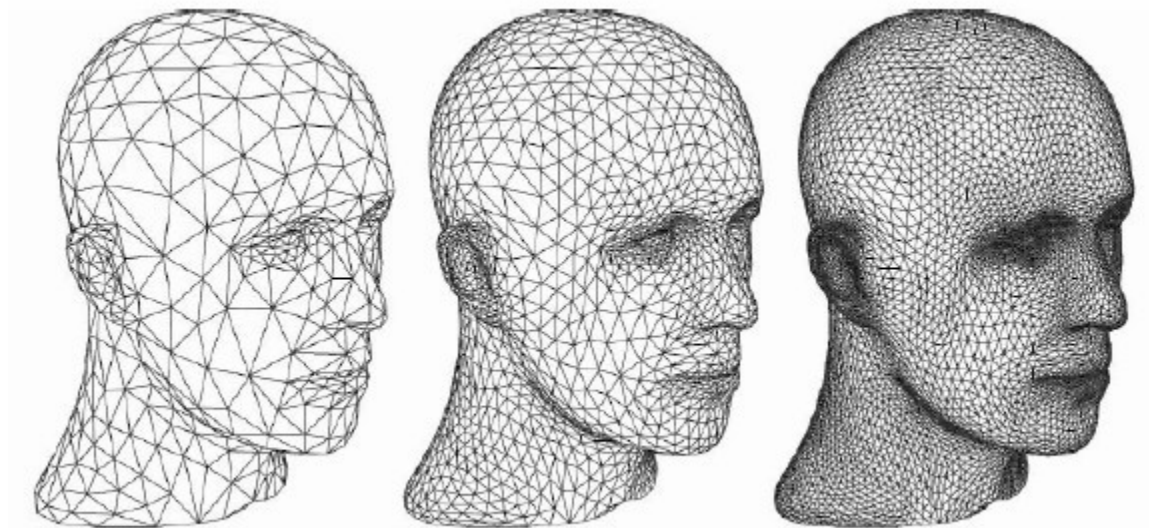
## 50-Поясните, что такое полигональная сетка?

Полигональная сетка — это совокупность вершин, ребер и граней, которые определяют форму многогранного объекта в трехмерной компьютерной графике и объемном моделировании.



Гранями обычно являются треугольники, четырехугольники или другие простые выпуклые многоугольники (полигоны), но сетки могут также состоять и из наиболее общих вогнутых многоугольников, или многоугольников с дырками.

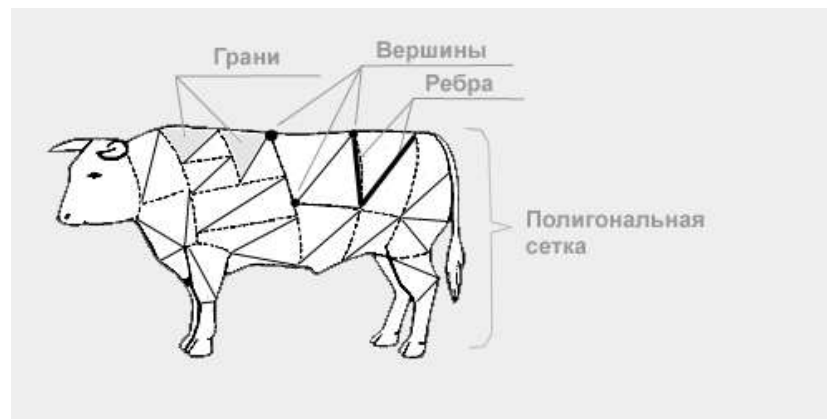
Учение о полигональных сетках — это большой подраздел компьютерной графики и геометрического моделирования. Множество операций, проводимых над сетками, может включать булеву алгебру, сглаживание, упрощение и многие другие. Разные представления полигональных сеток используются для разных целей и приложений. Полигональные сетки явно представляют лишь поверхность, а не объём.



## 51-Какие существуют способы задания полигональной сетки?

Полигональные сетки могут быть представлены множеством способов, используя разные способы хранения вершин, ребер и граней. В них входят:

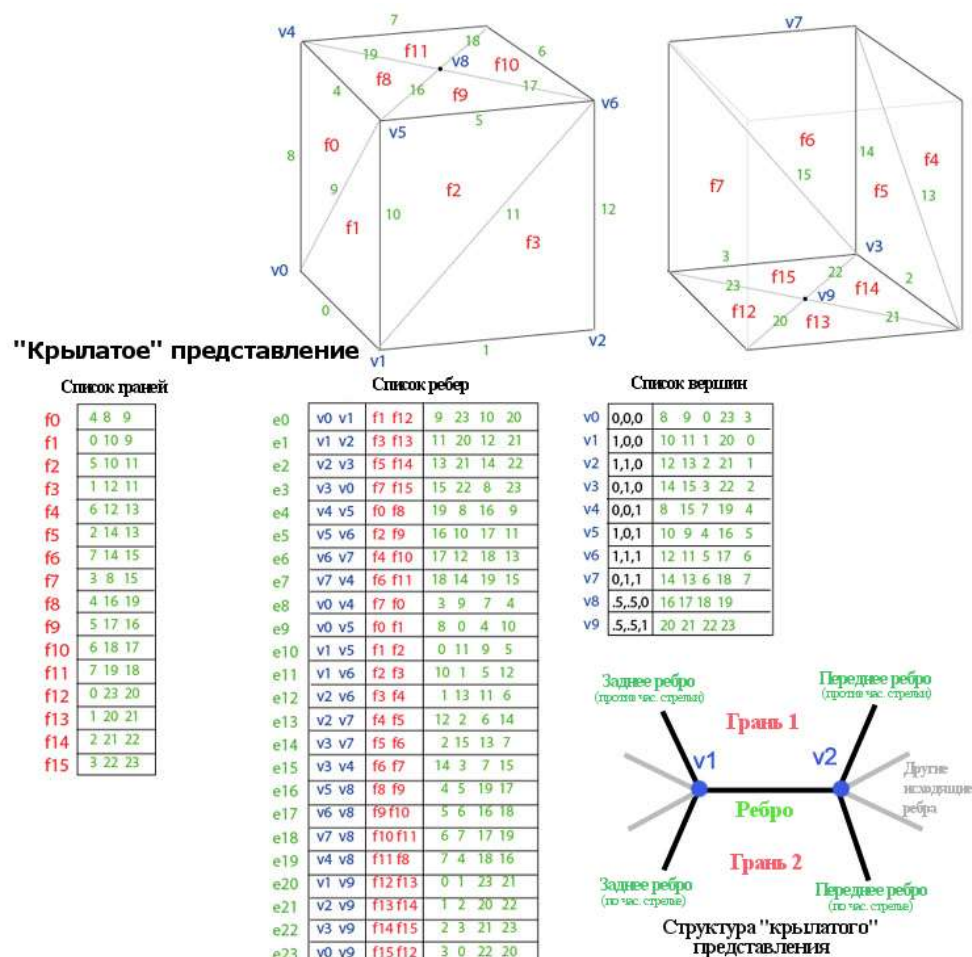
-Список граней: описание граней происходит с помощью указателей в список вершин. Сетка с использованием списка граней представляет объект как множество граней и множество вершин. Это самое широко используемое представление, будучи входными данными, типично принимаемыми современным графическим оборудованием. Список граней лучше для моделирования, чем вершинное представление тем, что он позволяет явный поиск вершин грани, и граней окружающих вершину.



-Крылатое представление: в нём каждая точка ребра указывает на две вершины, две грани и четыре (по часовой стрелке и против часовой) ребра, которые её касаются. Крылатое представление позволяет обойти поверхность за постоянное время, но у него большие требования по памяти хранения. Это представление широко используется в программах для моделирования для предоставления высочайшей гибкости в динамическом изменении геометрии сетки, потому что могут быть быстро выполнены операции разрыва и объединения.

Их основной недостаток: высокие требования памяти и увеличенная сложность из-за содержания множества индексов.

Вершинное



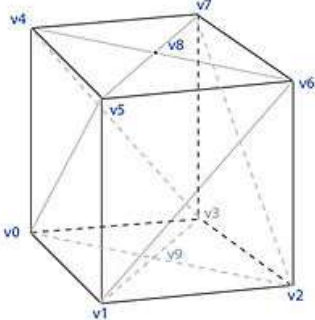
представление: представлены лишь вершины, указывающие на другие вершины. Информация о гранях и ребрах выражена неявно в этом представлении. Однако, простота представления позволяет проводить над сеткой множество эффективных операций. Вершинное представление описывает объект как множество вершин, соединенных с другими вершинами. Это простейшее представление, но оно не широко используется, так как информация о гранях и ребрах не выражена явно. Поэтому нужно обойти все данные, чтобы сгенерировать список граней для

рендеринга. Кроме того, нелегко выполняются операции на ребрах и гранях. Однако, сетки ВП извлекают выгоду из малого использования памяти и эффективной трансформации.

### Вершинное представление

Список вершин

v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,0	v5 v6 v7 v8
v9	.5,.5,1	v0 v1 v2 v3



-Четырёх рёберные сетки, которые хранят ребра, полуребра и вершины без какого-либо указания полигонов. Полигоны прямо не выражены в представлении, и могут быть найдены обходом структуры. Требования по памяти аналогичны полурёберным сеткам

-Таблица углов, которые хранят вершины в предопределенной таблице, такой, что обход таблицы неявно задает полигоны. В сущности, это "веер треугольников", используемый в аппаратном рендеринге. Представление более компактное и более производительное для нахождения полигонов, но операции по их изменению медленны. Более того, таблицы углов не представляют сетки полностью. Для представления большинства сеток нужно несколько таблиц углов (вееров треугольников).

## 52. В чём суть проблемы загораживания? Какие четыре характеристики лежат в основе различий известных методов решения задачи загораживания?

Задача загораживания (удаление невидимых частей сцены) возникла в процессе визуализации каркасных изображений. Чтобы создать иллюзию непрозрачности отображаемых объектов, нужно найти и отобразить только части объекта, видимые для наблюдателя.

В алгоритмах загораживания обычно предполагается, что экран расположен в проекционной плоскости  $Z = 0$ , сцена находится позади него, а наблюдатель – перед экраном. При проектировании проекторы проводятся через каждую точку объекта, тогда видимыми будут те точки, которые вдоль направления проектирования располагаются ближе всего к экрану.

Нет единственного универсального решения задачи. Известные методы решения различаются между собой по следующим характеристикам:

1. Выбор структуры данных для представления поверхностей:
  - аналитическое представление;
  - уравнения поверхностей: сфера, цилиндр, конус;
  - совокупность полигонов;
  - бикубические сегменты.
2. Способ визуализации поверхности:
  - получение сеточных изображений;
  - получение полутоновых или закрашенных изображений.
3. Использование специфических геометрических свойств изображаемых объектов.

#### 4. Пространство работы алгоритма:

- пространство *объектов* (ПО)
- пространство *изображений* (ПИ)

Алгоритмы, работающие в ПО, имеют дело с системой координат объекта, в декартовом пространстве каждый объект сцены сравнивается с оставшимися  $n - 1$  объектами. Объем вычислений пропорционален квадрату объектов. Достоинство: качество, возможность масштабировать без потери качества.

Алгоритмы, работающие в ПИ, отталкиваются от приборной системы координат экрана, т.е. определяют видимость каждого пикселя картины. Объем вычислений пропорционален  $n * N$ ,  
 $n$  – кол-во объектов в сцене,  
 $N$  – число пикселей экрана.

В общем случае алгоритмы загораживания нужно строить так, чтобы каждый шаг был как можно более эффективным, т.к от этого зависит качество и быстрота построения изображения в целом.

### 53. 54. 55.

Алгоритмы переборного типа относятся к первым алгоритмам. В них пытались точно решить задачу загораживания (**33**), для чего использовали геометрический подход: рассматривалось каждое ребро сцены и его положение сопоставлялось с каждой из граней, при этом можно выделить 3 случая:

- 1) ребро видимое, расположено между гранью и наблюдателем.
- 2) ребро находится за гранью
- 3) ребро пересекает грань.

Третий случай приводили к первым двум делением ребра на части. Алгоритм завершался в случае, если были просмотрены все грани, и выводилась видимая часть, либо, когда всё ребро становилось невидимым.

Алгоритмы данного типа работают в пространстве объекта. Типичным представителем данного семейства является алгоритм Робертса: прежде всего в нем удаляются невидимые части, экранируемые самим телом, а затем те, которые загораживает другой объект, после чего каждая из видимых частей, каждого тела, сравнивается с каждым из оставшихся. Достоинство: использование простых, мощных и точных математических методов, которые используются до сих пор.

Метод удаления **не лицевых** граней: внешняя сторона объекта (лицевая грань) будет видима наблюдателю тогда, когда её нормаль будет направлена к наблюдателю, т.е. угол  $\theta$  между вектором нормали и наблюдателем лежит в пределах  $-90^\circ \leq \theta \leq 90^\circ$ ,  $\cos \theta \geq 0$ . Знак косинуса легко определяется по скалярному произведению. Если уравнение плоскости грани имеет вид:  $ax + by + cz + d = 0$  то требуется проверить только знак коэффициента  $c$ .

Для нахождения нормали также можно воспользоваться векторным произведением любых 2х смежных векторов грани. Данный метод решает **33** полностью для визуализации выпуклых многогранников, но может быть применен и для решения частных задач отображения сцены.

Алгоритмы, задающие приближенное решение. **Алгоритм Варнока**. Его основные принципы исходят из того, что большая часть вычислений затрачивается на высоко информационные

области, считается что в ограниченном видимом объеме пространства можно распознать конечное число возможных ситуаций, для визуализации:

а) Проекция видимого объема пуста. Она не содержит проекцию ни одного видимого объекта сцены. Проекция объема- окно, которое закрашивается цветом фона.

б) Окно охватывается полностью ближайшей к нему по глубине грани проекций. Оно закрашивается цветом этой грани.

в) Размеры окна равны 1 пикселю. Этот пиксель закрашивается цветом ближайшей охватывающей грани.

г) Не распознана не одна из предыдущих ситуаций. Окно разбивается на 4 части, алгоритм повторяется для каждой части.

Алгоритм позволяет распараллелить вычисления, применим к удалению ребер и граней. Возможны дефекты из-за относительно произвольной разбивки для распознавания.

К данной группе можно отнести один из самых универсальных и используемых алгоритмов загораживания – алгоритм **Z-буфера**, или **буфера глубины** (был предложен Кеттелом), алгоритм отталкивается от пространства изображения, но включает цикл просмотра полигонов, а не пикселей и может быть реализован в процессе растривания объектов: информация о глубине размещения каждого полигона записывается в z-буфер. Его размеры:  $K_a \times K_b \times \Gamma$ , где  $\Gamma$  – количество битов для хранения информации о глубине сцены с приемлемой для количества изображения точностью,  $K_a$  – кол-во пикселей по горизонтали,  $K_b$  – по вертикали. Инициализируется буфер максимальной глубиной (аналогия с видеопамью). В процессе преобразования объектов каждая ячейка z-буфера содержит значение расстояния до ближайшего из обработанных полигонов, вдоль проецирующего луча, проходящего через соответствующий пиксель экрана. При растривании полигона для каждой его точки вычисляется расстояние от центра проецирования до экрана. Оно сравнивается с соответствующим для этой точки значением в z-буфере.

Когда значение сравниваемого пикселя больше хранящегося точку в буфер кадра не помещают. В противном случае данный пиксель заменяет точку экрана, а его z-глубина – соответствующую ячейку Z-буфера. Объем дополнительных вычислений невелик. Единственный недостаток – большой объем памяти.

Алгоритмы построчного сканирования работают в пространстве изображения, обрабатывают сцену, рассекая ее параллельными плоскостями, проходящими по сторонам развертки. Для каждой такой плоскости определяется совокупность отрезков – следов пересечения с ней граней объектов, т.е. задача 3D сводится к задаче 2D. Алг данного типа одни из первых были реализованы аппаратно и до недавнего времени считались наиболее эффективными. Отличие данных алгоритмов от алгоритмов с применением z-буфера состоит в том, что алгоритмы построчного сканирования должны просматривать при обработке все полигоны, претендующие на отображение в данной строке, а алгоритм z-буфера в текущий момент времени имеет дело с одним. Подобные алгоритмы требуют тщательно продуманной организации данных для полигонов, используются для создания компьютерных игр и являются универсальными.

Общая идея алгоритмов, использующих **список приоритетов** – получить преимущество за счет некоторого приоритета, например, сортировки по глубине. После такой сортировки любые два объекта не будут взаимно перекрывать друг друга и их можно вывести поочередно, начиная с дальнего. Тогда более близкие объекты затрут более дальние и задача удаления решится сама собой. Для простых объектов сцены – **алгоритм художника** (реализуется аппаратно).

К данной группе также относят алгоритм плавающего горизонта, который используется для визуализации сетчатых поверхностей, описываемых неявно  $f(x, y, z) = 0$ . Работает в пространстве изображения, обладает простотой и высокой скоростью реализации. Использует упорядочение по расстоянию до наблюдателя. Идея алгоритма состоит в том, что трехмерную задачу сводят к двумерной путем сечения поверхности последовательностью плоскостей имеющей постоянное значение одной из координат:  $x$ ,  $y$  или  $z$ , например,  $y = f(x, z, const)$  или  $x = f(y, z, const)$ . Получается, что поверхность представляется семейством кривых. Алгоритм хранит в памяти координаты 2-горизонтов: верхнего и нижнего т.е. хранится координата  $Z$ . При отображении каждой новой точки ее координату  $Z$  сравнивают с соответствующим значением верхнего и нижнего горизонта. Выводятся только пиксели с  $Z$  координатой больше верхнего и меньше нижнего горизонта. Метод просто совмещать с растриванием отрезков. Недостаток: появление дефектов на границах поверхности, в точках быстрого изменения функции и не прямоугольной области определения аргумента функции.

## Билет № 56.

### «Факторы создания реалистичных изображений»

1 Формулировка вопроса.

Какие факторы приходится учитывать при создании реалистичных изображений?

2 Факторы создания реалистичных изображений.

Для создания реалистичного изображения следует учитывать несколько факторов:

- Передача глубины (позволяет учесть расстояние до объектов).
- Освещение объектов (позволяет учесть форму, цвет и материал объекта).
- Наложение теней (позволяет уточнить геометрию сцены).
- Добавление оптических эффектов (позволяет получить дополнительную информацию о некоторых объектах, например: жидкостях, газах и т.п.).

3 Подробный разбор факторов.

3.1 Передача глубины.

Для передачи глубины принято использовать перспективу, яркость или бинокулярный эффект.



Геометрическая перспектива предполагает наличие бесконечно удалённой точки схода, к которой сходятся все прямые изображения. В зависимости расположения точки схода - за или перед рисунком – различают соответственно прямую и обратную перспективу. В добавок можно использовать отсечение по глубине, т.е. чем дальше находится объект, тем позже он будет изображён. Также следует скрывать невидимые линии для большего реализма. Использование перспективы наиболее эффективно для простых угловатых объектов. Для более сложных и нелинейных структур использование перспективы не даст ощутимого эффекта.

Представление глубины с помощью яркости – чем дальше объект от зрителя, тем меньше его яркость. Для реализации данного метода необходимо знать  $z$  координату объекта. Данный метод не позволяет показать значительные изменения глубины вследствие особенностей работы человеческого глаза.

При использовании бинокулярного эффекта создают два плоских изображения на основе разности которых составляется представление о глубине и, как следствие, трёхмерное изображение.

### 3.2 Освещение объектов.

При падении света на тело возможны следующие случаи :

- Свет полностью отражается (глянцевый объект).
- Свет частично отражается, частично поглощается.
- Свет полностью поглощается (матовый объект).
- Свет частично проходит (частично или полностью прозрачный объект).

Для придания большего реализма изображению следует правильно поставить отражённый свет. Отражённый свет зависит от строения, направления и формы источника света, а также от свойств поверхности, на которую падает свет.

Отражённый свет можно разделить на диффузную (рассеянную) составляющую и зеркальную (отражаемую в определённом направлении). К тому же следует учитывать фоновое освещение (ambient). При наложении света следует также учитывать световую составляющую объекта для получения наиболее реалистичных изображений.

### 3.3 Наложение теней и добавление оптических эффектов.

Полутень принято разделять на две части – полная тень (umbra) – наиболее тёмная часть тени и полутень (penumbra) – граница полной тени. Ввиду большой вычислительной сложности при наложении полутени эффективнее всего учитывать только полную тень. Если объект закрывает доступ света на другой объект, возникает проекционная тень. Наложение тени позволяет добавить реалистичности изображению и, возможно, оптимизировать вывод сцены, например, объект находится в тени можно не выводить его или выводить менее детализировано.



## Билет № 57.

### «Простая модель освещения. Моделирование источников света»

#### 1 Формулировка вопроса.

Поясните, какие источники света и каким образом моделируются в простой модели освещения?

#### 2 Виды источников света.

Выделяют 4 основных типа источников света:

- фоновое освещение (ambient lighting, прим. лампа дневного света)
- точечный источник (point sources, прим. лампа накаливания)
- прожектор (spotlights)
- удаленный источник (distant light, прим. Солнце)

Любой из них рассматривается как состоящий из 3х независимых источников, цветов RGB и описывается соответственно трехкомпонентной функцией излучения  $I = [I_R, I_G, I_B]$ . Вычисления для всех цветов осуществляется по одной схеме.

##### 2.1 Фоновое освещение.

Фоновое освещение дает равномерный цвет по всему пространству, т. е. считается, что точки света нет, направление различно. В простых моделях освещения считается, что каждая точка на поверхности объекта сцены освещается светом с одинаковой интенсивностью  $I_A$ .

##### 2.2 Точечный источник

При использовании точечного источника считается, что освещенность поверхности зависит от ориентации относительно источника. Максимальная освещенность при перпендикулярно падающих лучах от источника и уменьшается при уменьшении угла падения. Считается, что идеальный точечный источник излучает свет одинаково во всех направлениях. Источник характеризуется координатами точки, в которой находится источник. Освещенность поверхности им обратно пропорциональна квадрату расстояния между освещаемой точкой  $P$  и точкой источника  $P_0$  и рассчитывается по формуле (2).

$$L(P, P_0) = \frac{1}{(P - P_0)^2} L(P_0) \quad (2)$$

Такая модель проста, часто используется, но в комбинации с другими источниками. Учет расстояния от точечного источника вносит свою долю в изменение контрастности освещения. Вместо обратно пропорциональной зависимости в компьютерной графике используется модификация (3). Формула = корень квадратный от  $(a + bd + cd^2)$   $d = |P - P_0|$  – расстояние  $a, b, c$  – константы, которые подбираются из условия сглаживания контрастности.

$$L(P, P_0) = \frac{1}{\sqrt{a + bd + cd^2}}, \text{ где } d = |P - P_0| \quad (3)$$

## 2.3 Прожектор

Прожектор (направленный источник или точечный ближний) испускает свет пучком, т. е. в каком-то направлении. Проще всего прожектор моделируется точечным источником света, ограничив для него направление распространения световых лучей. Более близкой к реальному является модель прожектора, с функцией распределения интенсивности в конусе излучения. Эту функцию аппроксимируют как  $\cos \varphi$ , где  $\varphi$  – угол между осью косинуса и вектором, направленным на определенную точку поверхности, если он меньше  $\theta$ .  $|\varphi| \leq \theta$ . А  $\epsilon$  определяет быстроту убывания интенсивности по мере увеличения угла  $\varphi$ .

Удаленный источник света. Считается при моделировании, что все испускаемые им лучи параллельны. Моделируется заменой точечного источника параллельным пучком света. Обработка таких лучей аналогична параллельному проецированию.  $P_0=[x,y,z,1]$   $P_{0i}=[x,y,z,0]$ . При этом считается, что вектор  $(0;0;0;0)$  соответствует текущему источнику, тогда  $(0;0;-1;0)$  направление вглубь сцены, а  $(0;-1;0;0)$  – сверху вниз. Прожектор моделирует пространственную неоднородность освещенности и применяется в основном для видеоэффектов, т. к. приводит к уменьшению наглядности изображения.

## Билет № 58.

### «Простая модель освещения. Моделирование отражения света»

#### 1 Формулировка вопроса.

Поясните, какие источники света и каким образом моделируются в простой модели освещения?

#### 2. Составные части отражения света.

Отражённый свет можно разложить на несколько составляющих :

- фоновое освещение (ambient)
- рассеянное освещение (diffuse)
- зеркальное освещение (specular)

Фоновое освещение постоянно в каждой точке и не зависит от формы объекта. Интенсивность фонового освещения рассчитывается по формуле (1).

$$I_a = k_a I_a, \quad (1)$$

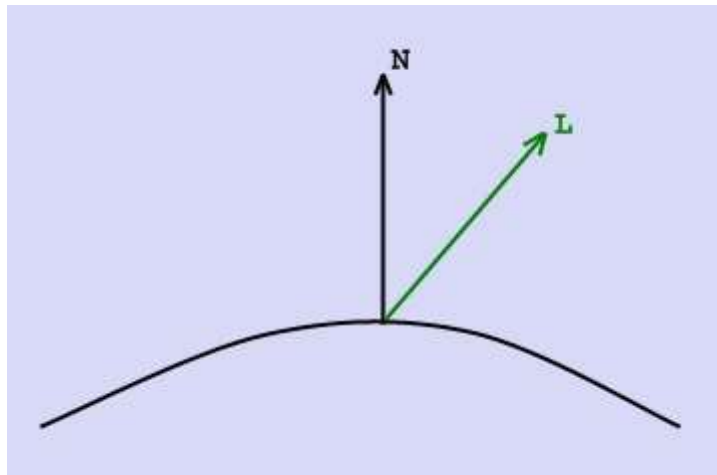
Где  $k_a$  – свойство материала воспринимать фоновое освещение,

$I_a$  – мощность фонового освещения,  $I_a$  – итоговая интенсивность фонового освещения.

## 2.1 Рассеянное освещение. Модель Ламберта.

Рассеянное освещение моделируется по эмпирической модели Ламберта.

Модель Ламберта моделирует идеальное диффузное освещение. Считается, что свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль  $N$ ) и направление на источник света (вектор  $L$ ). Рассеянная составляющая рассчитывается по закону косинусов (закон Ламберта):



Для удобства все векторы, описанные ниже, берутся единичными. В этом случае косинус угла между ними совпадает со скалярным произведением.

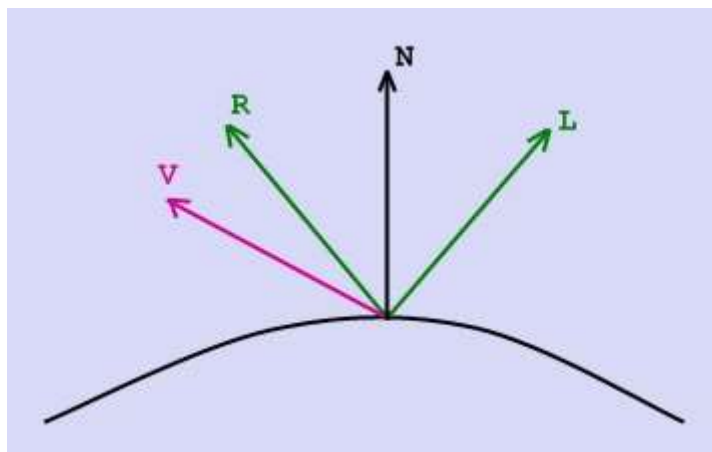
$$I_d = k_d \cos(\vec{L}, \vec{N}) i_d = k_d (\vec{L} \cdot \vec{N}) i_d$$

## 2.2 Зеркальное освещение. Модель Фонга.

Для учёта зеркального освещения применяется эмпирическая модель Фонга.

Модель Фонга – классическая модель освещения. Модель представляет собой комбинацию диффузной составляющей (модели Ламберта) и зеркальной составляющей и работает таким образом, что кроме равномерного освещения на материале может еще появляться блик. Местонахождение блика на объекте, освещенном по модели Фонга, определяется из закона равенства углов падения и отражения. Если наблюдатель находится вблизи углов отражения, яркость соответствующей точки повышается.

Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части. Т.о. отраженная составляющая освещенности в точке зависит от того, насколько близки направления на наблюдателя и отраженного луча. Это можно выразить следующей формулой:



В общем случае вектора  $V$ ,  $L$  и  $N$  не лежат в одной плоскости.

$$I_s = k_s \cos^\alpha (\vec{R}, \vec{V}) i_s = k_s (\vec{R} \cdot \vec{V})^\alpha i_s$$

где

$I_s$  – зеркальная составляющая освещенности в точке,

$k_s$  – коэффициент зеркального отражения,

$i_s$  – мощность зеркального освещения,

$R$  – направление отраженного луча,

$V$  - направление на наблюдателя,

$\alpha$  - коэффициент блеска, свойство материала.

Итоговое освещение рассматривается как линейная комбинация представленных выше трёх видов освещения.

## Билет № 59.

### «Методы закрашки полигональной сетки»

#### 1 Формулировка вопроса

Какие существуют методы закрашки полигональной сетки в компьютерной графике? В чём их отличие?

#### 2 Основные методы закрашки полигональной сетки.

Можно выделить три основных способа закрашки объектов, заданных полигональными сетками. В порядке возрастания сложности ими являются:

- однотонная закрашка
- метод Гуро (основан на интерполяции значений интенсивности)
- метод Фонга (основан на интерполяции векторов нормали).

В каждом из этих случаев может быть использована любая из моделей закрашки, описанная выше.

#### 2.1 Однотонная закрашка

При однотонной закрашке вычисляют один уровень интенсивности, который используется для закрашки всего многоугольника. При этом предполагается, что, во-первых, источник света расположен в бесконечности, поэтому произведение вектора к источнику освещения ( $\vec{L}$ ) и нормали к поверхности ( $\vec{N}$ ) постоянно на всей полигональной грани. Во-вторых, наблюдатель находится в бесконечности, поэтому произведение вектора к точке зрения ( $\vec{V}$ ) и вектора нормали ( $\vec{N}$ ) постоянно на всей полигональной грани и. В-третьих, полигон представляет реальную моделируемую поверхность, а не является аппроксимацией криволинейной поверхности.



Рисунок 1 – обозначения векторов

Если какое-либо из первых двух предположений оказывается неприемлемым, можно воспользоваться усредненными значениями  $\vec{L}$  и  $\vec{V}$ , вычисленными, например, в центре многоугольника. Последнее предположение в большинстве случаев не выполняется, но оказывает существенно большее влияние на получаемое изображение, чем два других. Влияние состоит в том, что каждая из видимых полигональных граней аппроксимированной поверхности хорошо

отличима от других, поскольку интенсивность каждой из этих граней отличается от интенсивности соседних граней. Различие в окраске соседних граней хорошо заметно вследствие эффекта полос Маха.

## 2.2 Метод Гуро

Метод тонирования Гуро — метод закрашивания в трёхмерной компьютерной графике (затенения), предназначенный для создания иллюзии гладкой криволинейной поверхности, описанной в виде полигональной сетки с плоскими гранями, путём интерполяции цветов примыкающих граней. Принцип метода состоит в последовательном вычислении нормалей к каждой из граней трёхмерной модели, дальнейшего определения нормалей вершин путём усреднения нормалей всех примыкающих к вершине граней. Далее на основании значений нормалей по выбранной модели отражения вычисляется освещённость каждой вершины, которая представляется интенсивностью цвета в вершине.

## 2.3 Метод Фонга

Закраска по методу Фонга является усовершенствованием метода Гуро. Подобно методу Гуро, в методе Фонга используются нормали вершин, однако, вместо интерполяции вычисленных для вершин значений, производится вычисление нормалей для каждой точки грани. Эта дополнительная работа обеспечивает более точный результат. Закраска по методу Фонга медленнее, чем закрашка по методу Гуро.

## Билет № 60.

### «Примитивы в компьютерной графике»

#### 1 Формулировка вопроса

Что называется примитивом в компьютерной графике? Каким образом примитивы используются для создания изображений?

#### 2 Определение примитива.

Примитив - это наименьшее неделимое с точки зрения прикладной программы элемент, используемый как базовый.

#### 3 Классификация примитивов

Примитивы подразделяют на:

- геометрические (точки, отрезки, ломаные, дуги)
- текстовые
- символьные (маркеры).

Различают физический и логический уровни формирования примитивов. На физическом уровне объект генерируется с помощью аппаратного блока, в то время как на логическом графический элемент генерируется программным образом.

#### 4 Свойства примитивов

Примитивы определяются параметрами и атрибутами. Среди параметров выделяют

- форму
- размер
- местоположение
- видимость
- яркость
- цвет
- режим мерцания
- толщина линии

Среди атрибутов можно выделить:

- визуальные свойства и статус по отношению к различным операциям (например, возможность или невозможность указания примитива)
- удаление
- преобразования.

## 5 Применение примитивов

Применение примитивов значительно облегчает процесс формирования объектов и изображения. Графический объект можно представить как множество примитивов под одним именем, которые обладают одинаковыми атрибутами. Графическая библиотека BGI (Borland Graphics Interface) состоит из постоянного ядра графической системы и набора графических драйверов, включаемых в систему по мере надобности (обеспечивают независимость программы от аппаратуры).

В растровых шрифтах каждый символ описан в виде набора точек (пикселей), расположенных в узлах сетки растра, т.е. по сути является обычным точечным рисунком. Растровые шрифты непригодны для высококачественной печати и используются в основном в программах с текстовым интерфейсом и в консоли. Они широко использовались в эпоху матричных принтеров и мониторов низкого разрешения.

В векторных (или контурных) шрифтах символы представляют собой криволинейные контуры, описываемые математическими формулами. Каждый знак описан с помощью векторов, определяющих координаты опорных точек, которые соединены прямыми или кривыми и образуют контур знака без привязки к абсолютному размеру или разрешению. Такое описание позволяет легко изменять масштаб изображения без потери качества, что невозможно в случае с растровыми шрифтами. Векторные шрифты одинаково выглядят как на экране, так и на бумаге.



## Билет № 61.

### «Основные особенности архитектуры OpenGL»

#### 1 Формулировка вопроса

Какие основные особенности архитектуры OpenGL?

#### 2 Определение OpenGL.

OpenGL – спецификация, разработанная Silicon Graphics и поддерживаемая Kronos Group, определяющая платформонезависимый программный интерфейс (API) для разработки графических приложений.

#### 3 Архитектура OpenGL

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

- аппроксимация кривых и поверхностей
- обработка вершин и сборка примитивов
- растеризация и обработка фрагментов
- операции над пикселями
- подготовка текстуры
- передача данных в буфер кадра

OpenGL можно сравнить с конечным автоматом, состояние которого определяется множеством значений специальных переменных (их имена обычно начинаются с символов GL\_) и значениями текущей нормали, цвета и координат текстуры. Все эта информация будет использована при поступлении в систему координат вершины для построения фигуры, в которую она входит. Смена состояний происходит с помощью команд, которые оформляются как вызовы функций.

Для обеспечения интуитивно понятных названий в OpenGL полное имя команды имеет вид:

`type glCommand_name[1 2 3 4][b s i f d ub us ui][v](type1 arg1,...,typeN argN)`

Таким образом, имя состоит из нескольких частей:

gl это имя библиотеки, в которой описана эта функция: для базовых функций OpenGL, функций из библиотек GLU, GLUT, GLAUX это gl, glu, glut, aux соответственно.

Command\_name - имя команды.

[1 2 3 4] - число аргументов команды

[b s i f d ub us ui] - тип аргумента:

символ b означает тип GLbyte (аналог char в C/C++), символ f тип GLfloat (аналог float), символ i – тип GLint (аналог int) и так далее.

Символы в квадратных скобках в некоторых названиях не используются. Например, команда `glVertex2i()` описана как базовая в библиотеке OpenGL, и использует в качестве параметров два целых числа, а команда `glColor3fv()` использует в качестве параметра указатель на массив из трех вещественных чисел.

### 3 Основные библиотеки OpenGL

Среди основных библиотек OpenGL можно выделить

- GLU (Utility library) – надстройка над OpenGL, позволяющая создавать сложные объекты, текстуры и т.д. Методы данной библиотеки имеют префикс `glu`.

- GLUT (Utility toolkit library, или аналог `freeglut`) – позволяет: создание окна, управление окном, мониторинг за вводом с клавиатуры и событий мыши; включает функции для рисования ряда геометрических примитивов. Методы данной библиотеки имеют префикс `glut`.

- GLM (Mathematics) – предоставляет доступ к структурам и функциям, позволяющим использовать данные для OpenGL

### 4 GLSL

GLSL (OpenGL Shading Language) – язык высокого уровня для программирования шейдеров, синтаксис языка базируется на языке программирования ANSI C с добавлением некоторых функций и типов данных для работы с матрицами. Является кроссплатформенным.

Различают вертексные (вершинные, геометрические) шейдеры, проводящие операции над вершинами геометрии и фрагментные (пиксельные) шейдеры, выполняющие преобразования пикселей геометрии и возвращающих цвет пикселя.

## Билет № 62.

### «Общая структура OpenGL программы»

#### 1 Формулировка вопроса

Какая общая структура OpenGL программы?

#### 2 Общая структура программы.

Простейшее консольное приложение, использующее OpenGL имеет следующую структуру:

- Инициализация GLUT

- Установка параметров окна.

- Создание окна.

- Установка функций, отвечающих за рисование в окне и изменении формы окна.

- Вход в главный цикл GLUT.

#### 3 Подробный разбор

Рассмотрим создание консольного приложения.

### 3.1 Инициализация GLUT

Инициализация GLUT производится командой:

```
void glutInit(int *argc, char **argv);
```

Первый параметр представляет собой указатель на количество аргументов в командной строке, а второй - указатель на массив аргументов. Обычно эти значения берутся из главной функции программы: `int main(int argc, char *argv[])`.

### 3.2 Установка параметров окна

Установка параметров окна содержит в себе несколько этапов. Прежде всего, необходимо указать размеры окна:

```
void glutInitWindowSize(int width, int height);
```

Первый параметр `width` - ширина окна в пикселях, второй `height` - высота окна в пикселях. Далее можно задать положение создаваемого окна относительно верхнего левого угла экрана. Делается это командой:

```
void glutInitWindowPosition(int x, int y);
```

Необходимо также установить для окна режим отображения информации. Т.е. установить для окна такие параметры как: используемая цветовая модель, количество различных буферов, и т.д. Для этого в GLUT существует команда:

```
void glutInitDisplayMode(unsigned int mode);
```

У команды имеется единственный параметр, который может быть представлен одной из следующих констант или комбинацией этих констант с помощью побитового ИЛИ.

Константа	Значение
GLUT_RGB	Для отображения графической информации используются 3 компоненты цвета RGB.
GLUT_RGBA	То же что и RGB, но используется также 4 компонента ALPHA (прозрачность).
GLUT_INDEX	Цвет задается не с помощью RGB компонентов, а с помощью палитры. Используется для старых дисплеев, где количество цветов например 256.
GLUT_SINGLE	Вывод в окно осуществляется с использованием 1 буфера. Обычно используется для статического вывода информации.
GLUT_DOUBLE	Вывод в окно осуществляется с использованием 2 буферов. Применяется для анимации, чтобы исключить эффект мерцания.
GLUT_ACCUM	Использовать также буфер накопления (Accumulation Buffer). Этот буфер применяется для создания специальных эффектов, например отражения и тени.
GLUT_ALPHA	Использовать буфер ALPHA. Этот буфер, как уже говорилось используется для задания 4-го компонента цвета - ALPHA. Обычно применяется для таких эффектов как прозрачность объектов и антиалиасинг.

GLUT_DEPTH	Создать буфер глубины. Этот буфер используется для отсечения невидимых линий в 3D пространстве при выводе на плоский экран монитора.
GLUT_STENCIL	Буфер трафарета используется для таких эффектов как вырезание части фигуры, делая этот кусок прозрачным. Например, наложив прямоугольный трафарет на стену дома, вы получите окно, через которое можно увидеть что находится внутри дома.
GLUT_STEREO	Этот флаг используется для создания стереоизображений. Используется редко, так как для просмотра такого изображения нужна специальная аппаратура.

Вот пример использования этой команды: `void glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);`

### 3.3 Создание окна.

После того как окно установлено необходимо его создать.

```
int glutCreateWindow(const char *title);
```

Эта команда создаёт окно с заголовком, который вы укажете в качестве параметра и возвращает HANDLER окна в виде числа int. Этот HANDLER обычно используется для последующих операций над этим окном, таких как изменение параметров окна и закрытие окна.

### 3.4 Установка функций, отвечающих за рисование в окне и изменении формы окна.

После того, как окно, в которое будет выводиться (рендерится) графическая информация, подготовлено и создано, необходимо связать с ним процедуры, которые будут отвечать за вывод графической информации, следить за размерами окна, следить за нажатиями на клавиши и т.д. Самая первая и самая необходимая функция отвечает за рисование. Именно она всегда будет вызываться операционной системой, чтобы нарисовать (перерисовать) содержимое окна. Итак, задаётся эта функция командой:

```
void glutDisplayFunc(void (*func)(void));
```

Единственный параметр этой функции - это указатель на функцию, которая будет отвечать за рисование в окне. Например, чтобы функция `void Draw(void)`, определенная в вашей программе отвечала за рисование в окне, надо присоединить ее к GLUT следующим образом: `glutDisplayFunc(Draw);`

И ещё одна функция важная функция - это функция, которая отслеживает изменения окна. Как только у окна изменились размеры, необходимо перестроить вывод графической информации уже в новое окно с другими размерами. Если этого не сделать, то, например, увеличив размеры окна, вывод информации будет производиться в старую область окна, с меньшими размерами. Определить функцию, отвечающую за изменение размеров окна нужно следующей командой:

```
void glutReshapeFunc(void (*func)(int width, int height));
```

Единственный параметр - это указатель на функцию, отвечающую за изменение размеров окна, которая как видно должна принимать два параметра `width` и `height`, соответственно ширина и высота нового (измененного) окна.

Среди других часто используемых функции можно выделить функции работы с мышью и клавиатурой.

### 3.5 Вход в главный цикл GLUT.

Ну и последнее, что необходимо сделать, чтобы запустить программу - это войти в так называемый главный цикл GLUT. Этот цикл запускает на выполнение так называемое ядро GLUT, которое обеспечивает взаимосвязь между операционной системой и теми функциями, которые отвечают за окно, получают информацию от устройств ввода/вывода. Для того, чтобы перейти в главный цикл GLUT, необходимо выполнить единственную команду:

```
void glutMainLoop(void);
```