**Билет №7**

## 1. Проектирование программ: связность и цельность программных модулей.

Программное обеспечение проектируется по следующим принципам:
1. Принцип системного единства
2. Принцип развития
3. Принцип совместимости
4. Принцип стандартизации

Принцип системного единства подразумевает, что при создании, развитии и функционировании САПР связи между компонентами ПО должны обеспечивать ее целостность.

Принцип развития. ПО САПР должно создаваться и функционировать с учетом пополнения, совершенствования и обновления ее компонентов.

Принцип совместимости. Языки, символы, коды, информация и связи между компонентами системы должны обеспечивать их совместное функционирование и сохранять открытую структуру системы в целом.

Принцип стандартизации. При проектировании ПО САПР необходимо максимально унифицировать, типизировать и стандартизировать ПО, которое должно быть инвариантным (независимым) к проектируемым объектам.

*Связанность (coupling) модуля* является мерой взаимозависимости модулей. При создании систем необходимо стремиться к максимальной независимости модулей, т.е. связанность модулей должна быть минимальной.

При проектировании систем допустимыми являются: *связанность (сцепление) по данным*, *связанность по образцу* и *связанность по управлению*.

**Модули связаны по данным**, если они взаимодействуют через передачу параметров и при этом каждый параметр является элементарным информационным объектом. Это наиболее предпочтительный тип связанности (сцепления).

**Модули связаны по образцу** если один модуль посылает другому составной информационный объект (например, объект – библиографическая запись, которая содержит имя автора, название книги и т.д.).

**Модули связаны по управлению**, если один посылает другому информационный объект – флаг, предназначенный для управления его внутренней логикой.

**Модули связаны по общей области** в том случае, если они ссылаются на одну и туже область глобальных данных. Связанность (сцепление) по общей области является нежелательным, так как, во-первых, ошибка в модуле, использующем глобальную область, может неожиданно проявиться в любом другом модуле; во-вторых, такие программы трудны для понимания, так как программисту трудно определить какие именно данные используются конкретным модулем.

**Модули связаны по содержимому** в том случае, если один из них ссылается внутрь другого. Это недопустимый тип сцепления, ибо полностью противоречит принципу модульности, т.е. представления модуля в виде черного ящика.

## 2. Постоянные запоминающие устройства. Область применения.

ПЗУ предназначены для хранения постоянной или редко изменяющейся информации, которую можно считать также просто, как из ОЗУ.

***По технологии изготовления кристалла:***

- **ROM** — (англ. *read-only memory*, постоянное запоминающее устройство), масочное ПЗУ, изготавливается фабричным методом. В дальнейшем нет возможности изменить записанные данные.
- **PROM** — (англ. *programmable read-only memory*, программируемое **ПЗУ (ППЗУ**)) — **ПЗУ**, однократно «прошиваемое» пользователем.
- **EPROM** — (англ. *erasable programmable read-only memory*, перепрограммируемое/репрограммируемое **ПЗУ (ППППЗУ/РПЗУ**)).
- **EEPROM** — (англ. *electrically erasable programmable read-only memory*, электрически стираемое перепрограммируемое **ПЗУ**). Память такого типа может стираться и заполняться данными несколько десятков тысяч раз. Одной из разновидностей EEPROM является **флеш-память** (англ. *flash memory*).

***По виду доступа:***

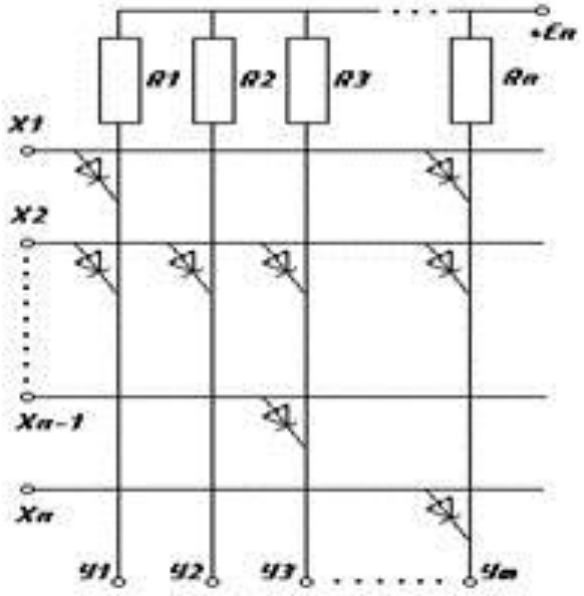
- **С параллельным доступом** – такое ПЗУ может быть доступно в системе в адресном пространстве ОЗУ;
- **С последовательным доступом** – такие ПЗУ часто используются для однократной загрузки констант или прошивки в процессор или ПЛИС, используются для хранения настроек каналов телевизора, и др.

***По способу программирования микросхем (записи в них прошивки):***

- Непрограммируемые ПЗУ;
- ПЗУ, программируемые только с помощью специального устройства — *программатора ПЗУ* (как однократно, так и многократно прошиваемые). Использование программатора необходимо, в частности, для подачи нестандартных и относительно высоких напряжений (до +/- 27 В) на специальные выводы.
- Внутрисхемно (пере)программируемые ПЗУ (*ISP, in-system programming*) — такие микросхемы имеют внутри генератор всех необходимых высоких напряжений, и могут быть перепрошиты без программатора и даже без выпайки из печатной платы, программным способом.

В ПЗУ записывают программы в микроконтроллерах, начальные загрузчики (BIOS) в компьютерах, таблицы коэффициентов цифровых фильтров в сигнальных процессорах.

Для примера рассмотрим устройство масочных ПЗУ. В них накопитель программируется на стадии изготовления и информация, записываемая в него, определяется построением одного из слоев схемы при помощи специальных фотошаблонов. Запоминающая ячейка МПЗУ состоит из одного элемента – диода, и запись информации осуществляется включением этого элемента в определенное перекрестье матрицы. Это осуществляется при изготовлении кристалла МПЗУ с помощью сменной маски.

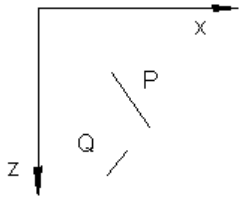



Выбор слова (информации) осуществляется при положительном сигнале на одной из линий шины адреса $X_1$-$X_п$. Если диод присутствует в перекрестии, то на линии У появится 1 (для $X_1$- это линии $У_2$ и $У_m$ ), а если диода нет, то 0 (это линии $У_2$ и $У_{3)}$. Недостаток схемы - плохие динамические характеристики выходного сигнала, чтобы устранить этот недостаток, вместо диодов применяют БТ или МОП транзисторы. Применение такого транзистора приводит к улучшению динамических характеристик и ускорению перезаряда емкостей, что увеличивает быстродействие.

## 3. Приоритетные методы удаления скрытых поверхностей. BSP – деревья.

***Алгоритм, использующий список приоритетов.***

В алгоритме художника все компоненты 3D-сцены сортируются в порядке минимального значения глубины полигона. Дальнейшее развитие этот алгоритм получил в алгоритме, использующем список приоритетов.




Все полигоны 3D-сцены сортируются в порядке увеличения глубины z. Первые два полигона в этом списке именуются P и Q. Если $z^P_{max} < z^Q_{min}$, то P не заслоняет Q. Тогда P выводим в растр. Иначе рассматриваются 5 тестов. Если хоть один из этих тестов срабатывает, то P не загораживает Q, и P выводится в растр.

Тесты:

1. верно ли, что прямоугольные оболочки проекций P и Q не пересекаются по оси x;

2. верно ли, что прямоугольные оболочки проекций P и Q не пересекаются по оси y;

3. верно ли, что P целиком лежит по ту сторону плоскости, в которой лежит Q, которая находится дальше от камеры;

4. верно ли, что Q целиком лежит по ту сторону плоскости, в которой лежит P, которая находится ближе от камеры;

5. верно ли, что проекции полигонов P и Q на ось OXY не пересекаются.

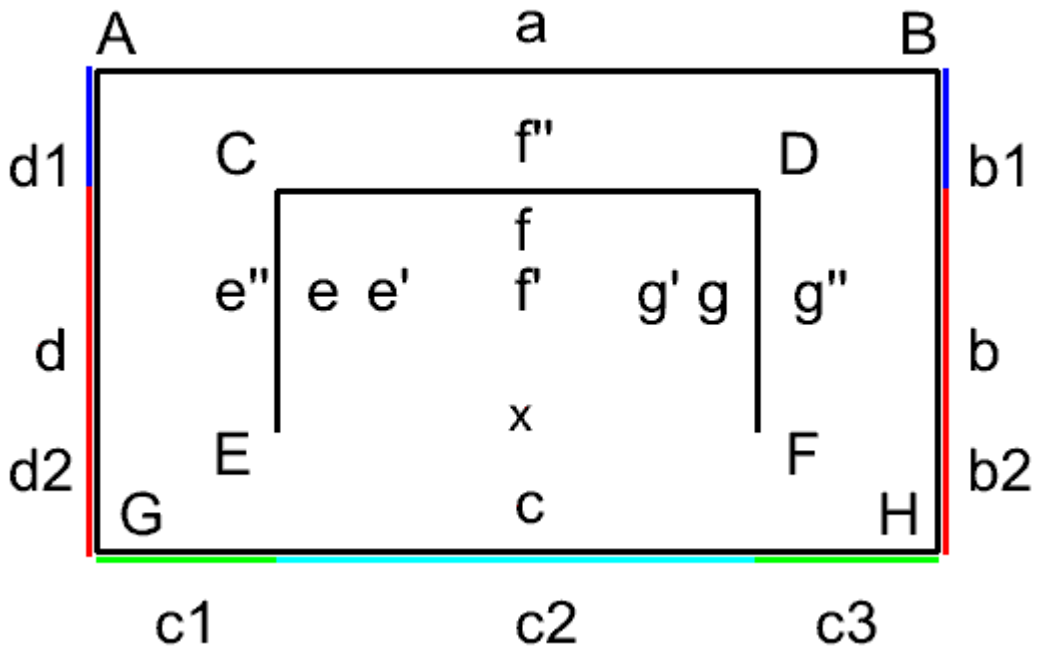
Если ни один из тестов не дает ответ “да”, то меняют местами в списке P и Q и выполняют тесты заново. Если в этом случае тесты не сработали, то произошло зацикливание. В этом случае P разбивает Q на 2 части P1 и P2. После разбиения переходим к первому пункту алгоритма.

***BSP-деревья***

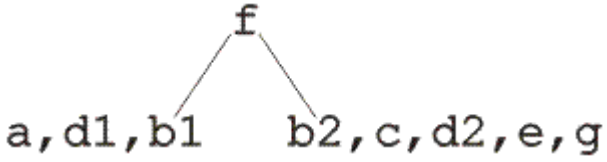
Дальнейшее развитие предыдущий алгоритм получил в алгоритме, использующем BSP – дерево.

***BSP дерево*** – это некая структура данных, которая, будучи построена один раз для некоторого 3D обьекта, позволяет потом без особых затрат времени сортировать по удаленности поверхности этого 3D обьекта при рассмотрении его с разных точек зрения.

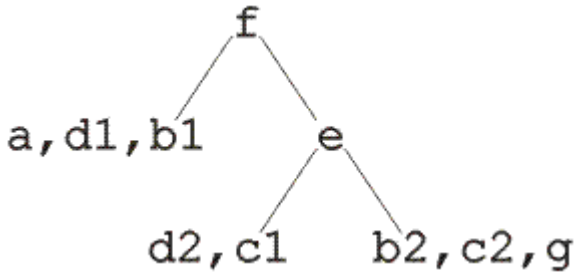
Для того чтобы объяснить принцип работы BSP дерева лучше начать с примера. Рассмотрим рисунок комнаты.



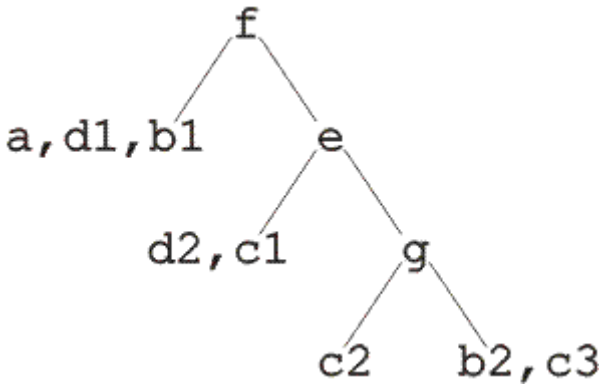

Выберем поверхность **f** в качестве корневой. Разобьём поверхности **b** и **d**, так как они не находятся ни точно слева, ни точно справа от **f**. Разобьём все поверхности на две категории: те, что слева (они все будут в левой ветви) и те, что справа (они будут в правой ветви). Это будет разбиение:




Мы можем завершить формирование левой ветви. Вершины **a**,**d1**,**b1** никогда не будут зрительно перекрывать друг друга. С другой стороны, то есть справа, поверхность **e** может иногда закрывать **d2**. Так что мы выбираем **e** и делим с помощью поверхности **e** как с помощью поверхности **f**. Это заставляет нас разбить **c**. Мы разбили все поверхности так:




Затем, разобьём по поверхности **g**, разбивая опять **c**, и все получившиеся разбиения будут представлять собой листья.




Будем разбивать до тех пор, пока не останется по одной поверхности в вершине: