19.06.14-----------------------------------------------------------------------

**Алгоритм Робертса удаления невидимых рёбер**

Работает с выпуклыми телами.

Этапы:

0. подготовка исходных данных

I. удаление рёбер, экранируемых самим телом

//работа может закончиться уже на этом этапе

II. нахождение и удаление рёбер, которые экранируются другими объектами сцены

//наиболее интересный вопрос с позиции математики

III. удаление рёбер новых рёбер (возникающих если объекты проходят друг сквозь друга), экранируемых самии телами, и другими объектами сцены

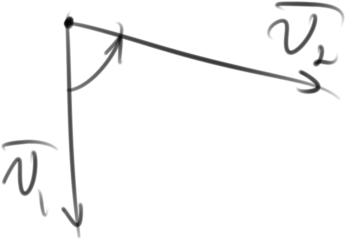
*0 Подготовка данных.*

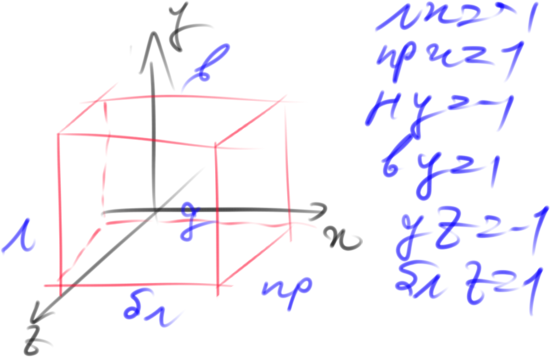
Каждое тело сцены описывается матрицей тела: 4xN – 4 параметра плоскости, n многогранников.

, ax+by+cz+d=0. Каждый столбец – уравнение плоскосоти, проходящей через грань. Решаем систему уравнений ax\_i + by\_i + cz\_i + d=0, i=1..4

Уравнение можно пронормировать, поделив на d, a’x+b’y+c’z+1=0;

В тоже время, можно умножить векторно V1 x V2:





. Далее необходимо проверить корректность задания матрицы. Любая точка, лежащая внутри тела, должна располагаться по «положительную» сторону от любой его грани. Если это будет не так, то соответствующий столбец нужно умножить на -1. Нужно взять произвольую точку внутри тела, сформировать вектор однородных координат и умножить на матрицу; в качестве точки можно взять полусумму максимальных и минимальных координат. В нашем случае пробная точка Р имеет координаты [P] = [0 0 0 1]

[P][V] = [1 -1 1 -1 1 -1] – там где -1, нужно умножить соотетствующие столбцы на -1; получим .

Если над телом совершаются преобразования (переносы-масштабирования), то можно заново не составлять матрицу преобразованного тела.

Пусть V – матрица исходного, VT – матрица преобразованного, T – матрица преобразований, B – матрица координат вершин тела, BT – матрица координат вершин преобразованного тела. [BT] = [B][T]; [B][V] = [D] – в Д ненулевой элемент будет с точностью до коэффициента показывать расстояние до данной плоскости.

[BT][VT] = [D] – для двух преобразований, перенос и поворот. Тогда можно [B][V] = [BT][VT];

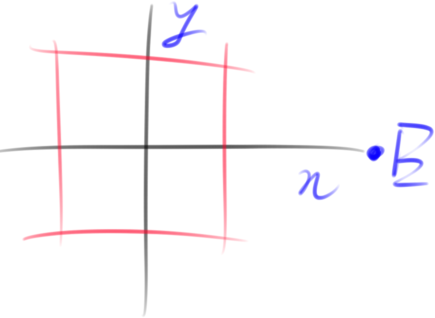
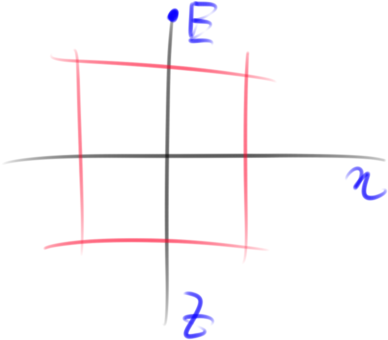
[B][V] = [B][T][VT]; умножая на обратную B, получаем [V] = [T][VT]; умножая на обратную Т, получаем .

Подготовительный этап завершён

*I Удаление рёбер, экранируемых самим телом.*

Нужно определить вектор взгляда [Е]=[0 0 -1 0] – наблюдатель находит на положительной оси З и смотрит к центру. Данную запись можно рассматривать как вектор взгляда, и как однородные кординаты некоторой точки, расположенной в –бесконечности оси З.

Любая точка внутри тела расположена по положительную сторону любой грани. Рассматривая проекции,



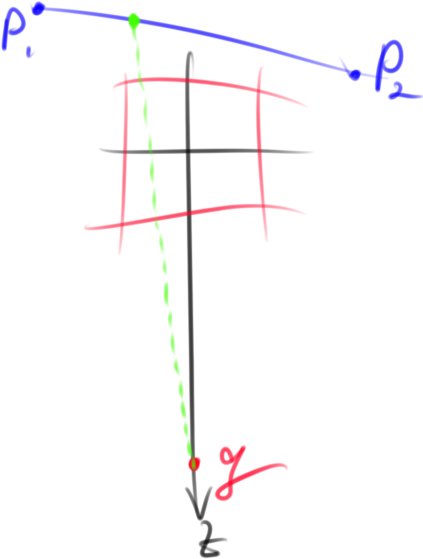
[E][V] = [0 0 0 0 **-1** 1]. Для дальней грани получили отрицательный результат, точка лежит по отрицательную сторону дальней грани – невидимой. Только для невидимых граней наблюдательная точка Е будет лежать по отрицательную сторону.

Таким образом, вектор однородных координат пробной точки (вектора взгляда) нужно умножить на матрицу тела и найти отрицательные компоненты – они будут соответствовать невидимым граням. Нули можно интерпретировать: ребро является граничным случаем между видимостью и невидимостью.

В простом случае, можно вычислить внутреннюю нормаль каждой грани и посмотреть, как она ориентирована относительно вектора взгляда. Если сонаправлена – то грань видима, если противоположна – то невидима, перпендикулярна – пограниченое состояние.

*II Удаление рёбер, экранируемых другими телами.*

Вначале на содержательном уровне сформулируем условие, при котором тело загораживает ребро. G – вектор координат точки, в которой находится наблюдатель, [g] = [0 0 1 0]. Проведём луч из произвольной точки в точку наблюдения. Если он встречает препятствие (проходит через тело), то точка невидима, а луч расположен по положительную сторону от каждой грани тела. //зелёный луч идёт параллельно, если точка расположена на бесконечности.



Проверить луч на расположенность по положительную сторону от каждой грани тела. Запишем уравнение луча через уравнение отрезка; P(t) = P1 + (P2-P1)t, 0<=t<=1 – уравнение анализируемого отрезка.

Отсюда уравнение луча Q(t,alpha) = P(t) + alpha\*g = P1 + d\*t + alpha\*g , где d=p2-p1, и alpha>=0 (тело находится между исследуемой точкой и точкой наблюдения). По сути, Q(t,alpha) будетуравнением плоскости, проходящей через точки. Альфа служит своего рода «расстоянием» от g до точки.

[H]=[Q][V]; . Луч должен располагаться по положительную сторону каждой грани тела; если все hj больше нуля, то это условие выполняется.

[H] = [P1][V] + t[d][V] + apha[g][V]; отсюда .

Здесь [Q] = [d][V]; [W] = [g][V].

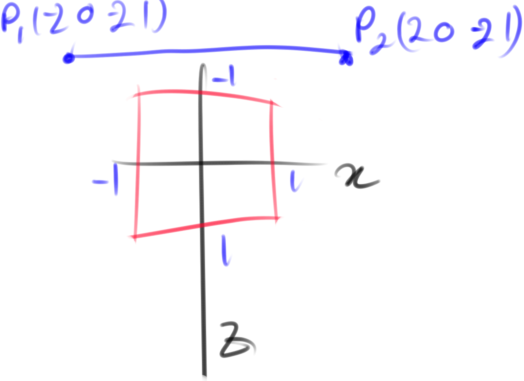
Имеем систему неравенств в общем случае: , m ограничений, n неизвестных.

В нашем случае, n ограничений, 2 неизвестных.

Зададим целевую функцию: . Целевая функция линейна, производная (которая может применяться для поиска минимума) равна нулю. Здесь С – некоторые коэффициенты. Стационарных точек здесь нет, минимумы надо искать на границах области. От ^ неравенств надо перейти к равенствам.

В данной ситуации функция, задающая ограничение, представляет собой прямую, определяемую переменными t,alpha; условия – неравенства. Точки, лежающие на прямой, обращают запись в равенство, точки лежащие по ту ли иную сторону прямой – в неравенства.

Рассмотрим пример.



[P1] = [-2 0 -2 1], [d] = [4 0 0 0]; [g] = [0 0 1 0];

[P1][V] = [P] = [-1 3 1 1 -1 3]

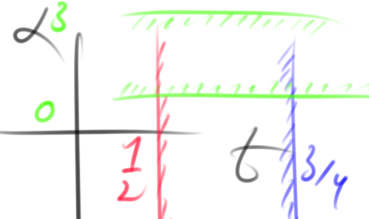
[d][V] = [Q] = [4 -4 0 0 0 0]

[g][V] = [W] = [0 0 0 0 1 -1]

Получаем упомянутые ранее ограничения.

1,2 – ближ-дальние грани, 3,4 – ниж-верхние грани (отрезок лежит между нижней и верхней гранями), 5,6 – лев-правые грани

От неравенств перейдём к равенствам – получим уравнения границ. В этой прямоугольной области все точки удовлетворяют неравенству и являются невидимыми.



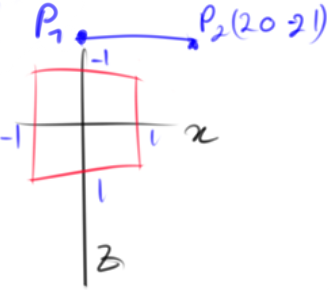
Целевая функция линейна – решения находим на найденных границах. . Выбираем t=const и сдвигаем её, пока не выйдем за область допустимых решений, . Аналогично с другой границей . Обозначим точками Р3 и Р4 границы невидимой области отрезка:

Р3 = (-2 0 -2 1) + ¼(4 0 0 0) = (-1 0 -2 1)

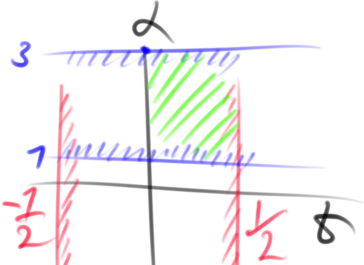
Р4 = (-2 0 -2 1) + ¾(4 0 0 0) = (1 0 -2 1).

В общем случае, может понадобиться учитывать ограничения параметров: t[0..1], alpha>0.

Начало исходного отрезка передвинем в середину: [P1]=[0 0 -2 1], [d]=[2 0 0 0].



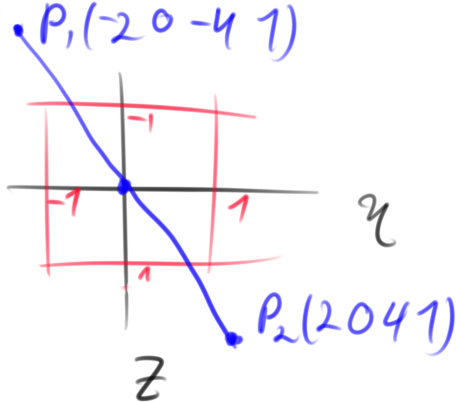
Получаем [P] = [1 1 1 1 -1 3]; [Q] = [2 -2 0 0 0 0]; [W] = [0 0 0 0 1 -1].

Отсюда ; 

Ограничение параметра альфа нужно в том случае, если отрезок проходит сквозь объект.

*III.1 нахождение точек протыкания*

Нас интересует видимые части рёбер – если связать ребром две точки протыкания, видимым оно не будет.

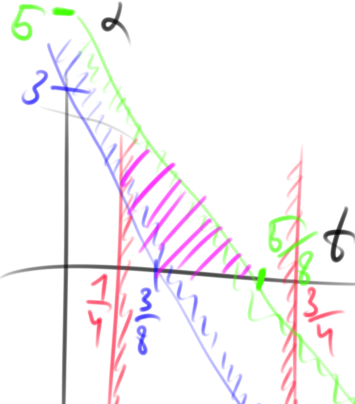


[p1] = [-2 0 -4 1]; [d] = [4 0 8 0]; [g] = [0 0 1 0]

[P] = [-1 3 1 1 -3 5]

[Q] = [4 -4 0 0 8 -8]

[W] = [0 0 0 0 1 -1]

Имеем .

Отсюда .

Точки протыкания – пересечения прямой с телом, при alpha!=0 мы бы получли точку на другой прямой. В этом случае получим – начало невидимой части, – конец невидимой части.

*Замечания.*

На ЭВМ используется перебор: n уравнений, 3 ограничения; n+3 уравнения для 2 неизвестных. Составляются все возможные комбинации систем по 2 уравнения. Составили очередную комбинацию, решили – убеждаемся, что полученное решение является решением оставшимся уравнений, необходимо подставить. Если полученное решение является решением всей системы, то оно рассматривается как канидат на поиск tmin и tmax.

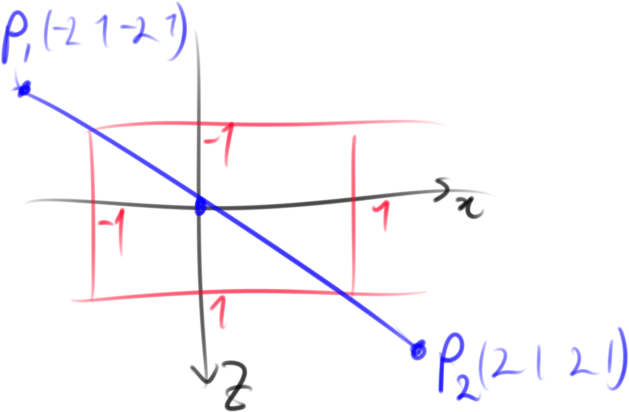
По формуле арифметической прогрессии: n+2 системы для первого уравнения, 1 для последнего; .

**Сокращение трудоёмкости**

Даная процедура достаточно трудоёмка.

Можно попробовать определить заведомо-невидимые рёбра – обе вершины лежат позади(на) невидимой грани. Тем не менее, грань конечна, а плоскость бесконечна – уравнение плоскости использовать нельзя.

Можно попытаться найти целиком видимые рёбра – обе вершины лежат на(перед) видимой грани; на плоскости или по отрицательную её сторону, при этом сама плоскость должна быть видимой. Наблюдатель также должен находиться по отрицательную сторону плоскости.



[p1] = [-2 1 -2 1]; [d] = [4 0 4 0]; [g] = [0 0 1 0]

[P] = [-1 3 2 0 -1 3]

[Q] = [4 -4 0 0 4 -4]

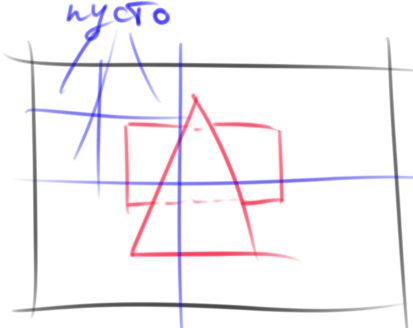
[W] = [0 0 0 0 1 -1]

Pj<=0 (перемножили вектор координат начальной точки на матрицу тела); pj+qj <= 0 – в этих случаях вершина видима. Если Wj<=0, то плоскость видима.

Анализируем компоненты первого вектора – нас интересуют неположительные значения. P1=-1 : <0, P1+Q1 > 0; не интересует. P2, P3>0. P4=0, P4+Q4=0; вдобавок к этому W4=0. Для четвёртой грани выполняются все условия. Это верхняя грань; отрезок расположен на верхней граи – он всегда видим.

**Алгоритм Варнока**

В то время как алгоритм Робертса работает в пространстве объектов, остальные алгоритмы работают в пространстве изображений.



Экран представляется в виде окна. Если в текущем окне непонятно, что изображать, то оно делится на четыре части, процесс повторяется.

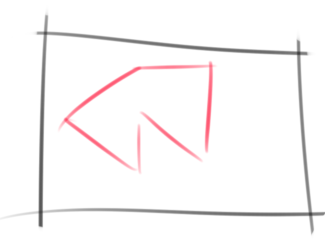
Единой версии алгоритма не существует. В простейшем варианте – окно делится на части каждый раз, если оно не пусто. Деление производится до тех пор, пока не получится окна размером 1х1 пиксель. Для окна размером в 1 пиксель мы можем однозначно сказать, что отрисовать надо элемент ближайшего к наблюдателю многоугольника (при размере 2 пикселя может возникнуть неоднозначность – многоугольники могут протыкаться). Окно является пустым, если все многоугольники являются внешними по отношению к этому окну.

В более сложных вариантах алгоритма предпринимается попытка ответа на вопрос на более ранних стадиях, не доходя до окна в 1 пиксель.

Необходимо выполнять анализ взаимных расположений многоугольника и окна.

*Классификации многоугольников*, рассматриваемых в алгоритме; анализ возможных случаев взаимного расположения:

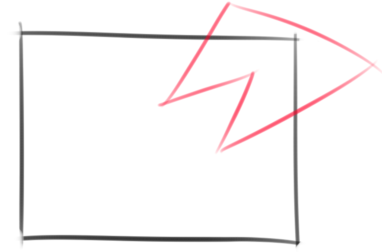
1) Один внутренний многоугольник.



а) окно закрасить цветом фона

б) выполнить растровую развёртку многоугольника

2) один пересекающий многоугольник



а) окно закрасить цветом фона

б) выполнить отсечение многоугольника по границам окна

в) растровая развёртка результата отсечения

3) все многоугольники внешние

Окно закрашивается цветом фона.

4) один охватывающий многоугольник (по размеру окна или больше)

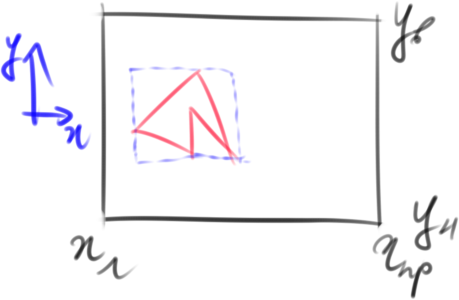
Окно закрашивается цветом многоугольника.

5) несколько многоугольников, охватывающий ближе всего

-''-

*Идентификация многоугольников.*

1) внутренний

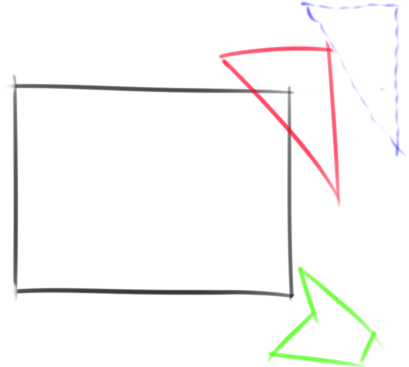


Используется объемлющая оболочка. (xmin >= xL) ^ (xmax <= xR) ^ (ymin >= yD) ^ (ymax <= yU).

2) часть внешних многоугольников

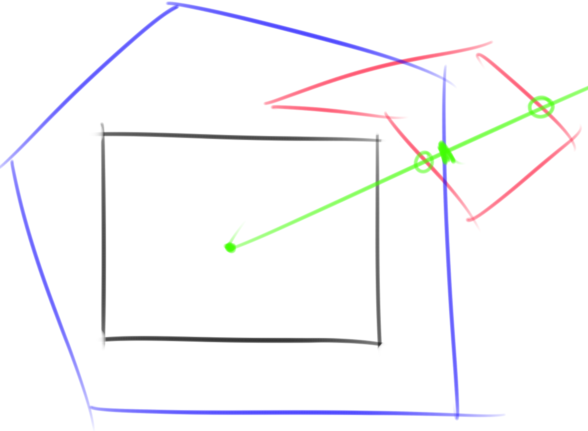
(xmax < xL) v (xmin > xR) v (ymax < yD) v (ymin > yU). Если оболочка пересекается с окном, то такая простая проверка не сработает.

3) пересекающие многоугольники



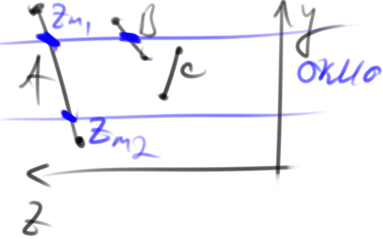
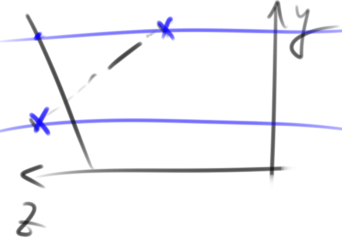
Можно использовать пробную функцию F\_пр = ax + by + c; для каждой стороны анализировать многоугольники. Если для всех вершин окна знак один и тот же, то они лежат по одну сторону многоугольника, пересекающим не является (синий). В противном случае ищутся точки пересечения; определяется их «корректность», лежат они на сторонах многоугольника, или на продолжениях многоугольника.

4) часть внешних и все охватывающие



Используется тест с лучом; испускается из центра и считается количество пересечений. Чётное – многоугольник внешний, нечётное – охватывающий. Необходмо анализировать вариант, когда луч проходит через вершину внешнего прямоугольника – нужно взять на луче две точки, соседние с точкой пересечения. Если их принадлежность многоугольнику будет одинаковая, то это точка касания, если принадлежность разная – точка пересечения.

Тест проверки расположения охватывающих: достаточное условие

Вычислить глубины (Z-коорд) всех многоугольников в вершине окна. Если Zохв\_i = max(j) Zij; i=1..4, j=1..m. Условие достаточное, но необходимое – может не выполняться.

Рассмотренный на лекции алгоритм Вейлера-Азертона является обобщением алгоритма Варнока. В варноке происходит много разбиений окна на подокна – форма и расположения окна не связаны с формой и расположением многоугольников. В алгоритме ВА мы берём за отсекатель один из многоугольник и отсекаем по его границам.

Алгоритм Z-буфера (разбираться самому).

Буфер можно выбрать размером в одну сканирующую строку. Поочередно рассматриваются сканирующие строки, для очередной строки рассматриваются все многоугольники; больше объём исходных данных.

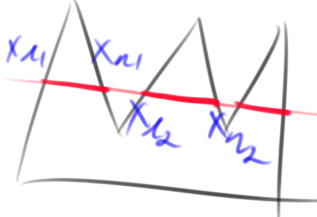
Может использоваться аналог алгоритма заполнения с активным ребром, однако также необходимо использовать активные грани.

Задание исходных данных:

- для каждого ребра каждого многоугольника:

• х верш; dX; dY

• z верш; dZ\_dx(-A/C) - изменение глубины при проходе по строке; dZ\_dy(-B/C) - изменение глубины при проходе по разным строкам. отсюда dZ=dZ\_dy\*1 + dZ\*dX

• образовать пары "пределов" по х 

- для каждого многоугольника:

• вычислить ymax, ymin и количество строк

Для каждой сканирующей строки выполнить следующие действия:

1. инициализировать й-буфер размером в одну строку

2. инициализировать буфер кадра, соответствующий текущей строке, фоном цвета

3. проверить список многоугольников и добавить при необходимости очередной многоугольник в список активных многоугольников (САМ) на текущей строке

4. если многоугольник добавлен в САМ, то к списку активных рёбер (САР) добавить активные рёбра многоугольника. Для каждого активного ребра сформировать информацию: Хл, dXл, dYл; Xпр, dXпр, dYпр; dZхл, dZyл, dZxп, dZyп; Zл, Zпр

5. в произвольном порядке обработать пары левых и правых рёбер в интервале Хл <= Х <= Хпр вычислять глубину каждой точки z(x,y) и сравнивать с глубиной z\_буф(x,y).

- если z(x,y) > z\_буф(x,y), то цвет(х,у) := цвет тек. многоугольника

6. скорректировать список активных ребер: dYл -= 1; dYпр -=1.

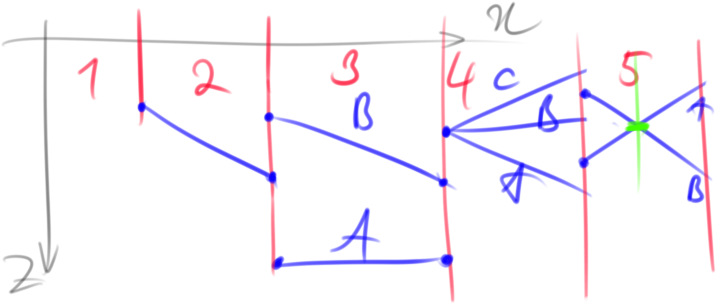
Если dYл < 0 или dYпр < 0, то удалить ребро из САР.

Если dY >=0, то Хл += дХ; Хп += дХ; Z += dZx\*dx + dZy\*dy.

Если ребро удаляется из САР, то проверить необходимость удаления многоугольника из САМ (dYмн<0). dYмн -= 1.

Если многоугольник не удаляется из САМ, а ребро удалено, то укомплектовать пару рёбер (если удалилось одно - добавить одно, если два - добавить два).

Идеи интервальных методов построчного сканирования



0 – пустой интервал

1 – один многоугольник

2 – несколько многоугольников, но один лежит ближе других. ZA1 > ZB1, ZA2 > ZB2

3 – ZA1 < ZB1 < ZC1

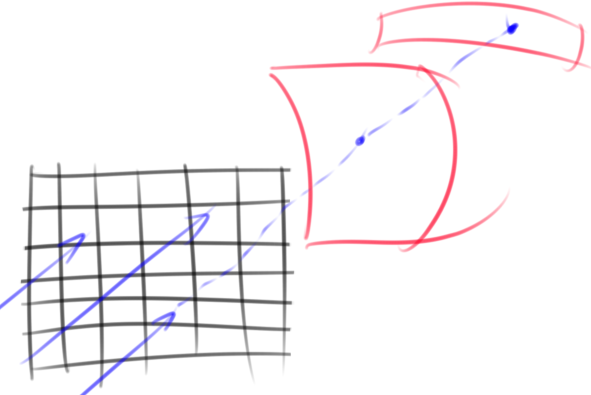
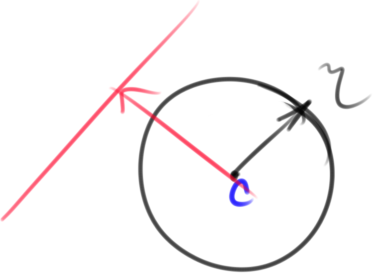
4 – ZA2 > ZB2, ZA2 > ZB2

**Алгоритм определения видимых поверхностей методом трассировки лучей**

Рассматриваем простейший вариант для решения задачи о видимых поверхностей. Используются принципы оптики.

Источник света испускает лучи света во все стороны, в камеру попадают отражённые лучи, благодаря которым видно поверхность. Прямая трассировка

Используют обратную трассировку – источником считается камера. Задача решается в задаче изображений – рассматриваем лучи, которые испускаются камерой и проходят через очередной пиксель экрана. Используется матрица пикселей, каждый пиксель является источником луча. Задача сводится к поиску пересечений лучей и поверхностей.

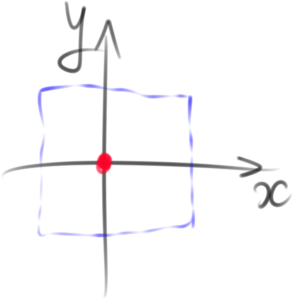
Можно использовать оболочки, хотя они и не дают однозначных результатов. Чаще всего используется сферическая оболочка.

Для луча P(t) = P1 + (P2-P1)t, t>=0. X(t) = x1 + at; Y(t) = y1 + bt; Z(t) = z1 + ct; a=x2-x1, b=y2-y1, c=z2-z1.

.

*.* Приравниваем производную нулю – получаем минимум, минимальное расстояние от центра до луча. Выразим t, . – пересечение есть.

Можно использовать и прямоугольную оболочку – но там сложнее найти точку пересечения с гранью. Для упрощения удобно выполнить преобразование, в результате которого трассируемый луч будет совмещён с осью Z

 Пересечение есть, если оболочка охватывает начало координат.

Поверхность можно задать уравнением второго порядка:

В результате преобразований, т.к. в точке пересечения х=у=0, получаем уравнение

Из которого можно определить точки пересечения.

Алгоритм:

I подготовка исходных данных:

- создать список объектов, содержащий следующую информацию: описание поверхности объекта (тип объекта) и цвет; описание сферической оболочки (центр, радиус)

- при использовании прямоугольной оболочки: x,y,z\_max,min

II для каждого трассирующего луча:

1. выполнить тест со сферической оболочкой. Если пересечение есть, то занести объект в список активных.

2. если САО пуст, то текущий пиксель (лучевой) высвечиваем цветом фона.

3. если САО не пуст, то находим преобразования, совмещающие трассируемый луч с осью Z и запоминаем это преобразование

4. для каждого объекта из списка активных выполнить:

4.1. если используется прямоугольная оболочка, то преобразовать её к новой системе координат. Для преобразованной оболочки выполнить тест: если пересечение есть, то заносим объект в САО

4.2. если не используется, то преобразовать очередной объект (из САО) в новую систему координат

4.3. находим пересечение луча с каждым активным объектом и заносим в список пересечений

4.4. для списка пересечений находим ближайшее к наблюдателю (Zmax)

4.5. если список пересечений пуст, отображаем пиксель фоновым цветом

4.6. если пересечение есть, то отобразить пиксель цветом ближайшего объекта (не используя модель освещения. Используя: вычислить положение точки пересечения в исходной системе координат по обратным преобразованиям; вычислить интенсивность и высветить пиксель.