



```
Head = new Cell;
Head->LeftTag = true;
Head->RightTag = true;
Head->Left = Head;
Head->Right = Head;
current = Head;
}

template <class T> void BinaT(parent, T, el)
{
    Cell *temp, *node;
    node = ToCell(el);
    temp = parent->Left;
    parent->Left = node;
    node->Left = temp;
    node->Right = parent;
    node->LeftTag = parent->LeftTag;
    parent->LeftTag = false;
    node->RightTag = true;

    if(!node->LeftTag){
        temp = node->Left;
        temp = GoToDownRight(temp);
        temp->Right = node;
    }

    template <class T> void BinaT(parent, T, el)
    Cell *temp, *node = ToCell(
    temp = parent->Right;
    parent->Right = node;
    node->Right = temp;
    node->RightTag = parent->RightTag;
    parent->RightTag = false;
    node->LeftTag = true;
```

Базовые растровые алгоритмы

Лекция № 4

Введение

Алгоритм вывода прямой линии

Инкрементные алгоритмы

- ***Алгоритм вывода линии***
- ***Алгоритм вывода окружности***
- ***Алгоритм вывода эллипса***

Кривая Безье

Алгоритм вывода фигур

Алгоритм закрашивания линиями

Алгоритмы заполнения, которые используют математическое описание контура

Наложение текстур на полигон

Фракталы

- ***Фрактал Мандельброта***
- ***Фрактал Джулия***

Геометрические фракталы

- ***Кривая Коха***

Фракталы, формируемые системой итеративных функций

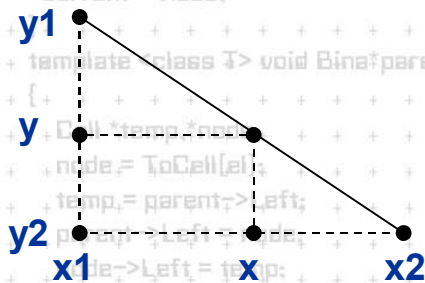
- ***Веточка папоротника***

Алгоритм вывода прямой линии

Пусть заданы координаты отрезка прямой (x_1, y_1) , (x_2, y_2) .

Необходимо закрасить все пиксели, вдоль прямой.

Для этого необходимо вычислить их координаты.



$$\frac{x - x_1}{y - y_1} = \frac{x_2 - x_1}{y_2 - y_1}$$



$$x = x_1 + (y - y_1) * \frac{x_2 - x_1}{y_2 - y_1} = f(y);$$

$$y = y_1 + (x - x_1) * \frac{y_2 - y_1}{x_2 - x_1} = F(x);$$

В зависимости от угла наклона прямой выполняется цикл по оси x или по оси y

Упрощенный алгоритм вывода линии:

```
float k;
```

```
k = (float) (y2-y1) / (float) (x2-x1);
```

```
y = y1;
```

```
for (x = x1; x <= x2; x++)
```

```
{
```

```
    Пиксель  $(x, y)$  ;
```

```
    y+=k;
```

```
}
```

Этот вариант наиболее быстрый.

Но, возникает погрешность округления.

Может произойти так, что $y \neq y_2$ на последнем шаге.

Инкрементные алгоритмы

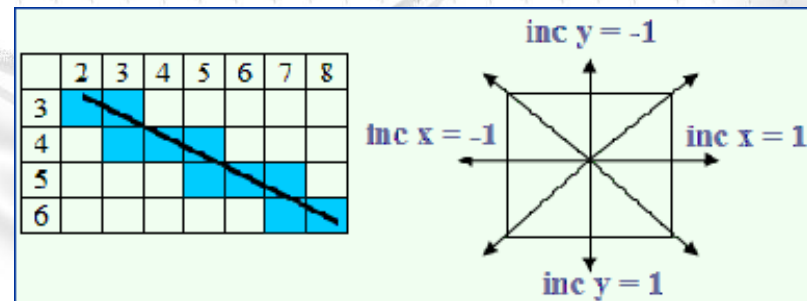
прямая

Брезенхэм предложил инкрементные алгоритмы растеризации.

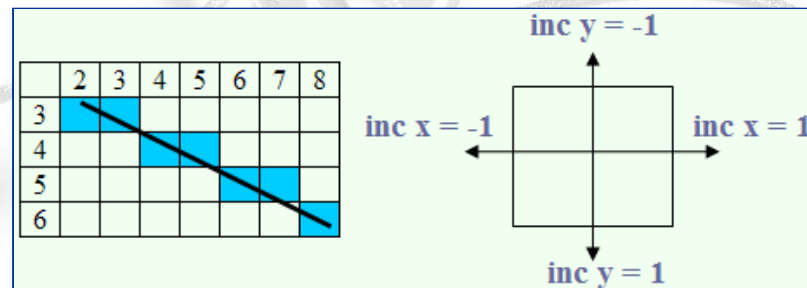
Цель: построение алгоритма только с целочисленными вычислениями, основанными на операциях сложения и вычитания, без умножения и деления.

Рассмотрим пример построения отрезка (2,3)-(8,6):

Этот алгоритм восьмисвязный



Известны также четырехсвязные алгоритмы.



Четырехсвязные алгоритмы проще, но в нем больше тактов и качество изображений хуже.

Инкрементные алгоритмы

окружность

Окружность $(X-x_c)^2 + (Y-y_c)^2 = R^2$

```
void Circle(int xc, int yc, int radius)
{
    int x,y,dxt;
    long r2,dst,t,s,e,ca,cd,index;
    r2 = (long)radius*(long)radius;
    dst = 4*r2;
    dxt = (double)radius/1.414213562373;
    t = 0;
    s = -4*r2*(long)radius;
    e = (-s/2)-3*r2;
    ca = -6*r2;
    cd = -10*r2;
    x = 0;
    y = radius;
    Пиксель(xc,yc+radius);
    Пиксель(xc,yc-radius);
    Пиксель(xc+radius,yc);
    Пиксель(xc-radius,yc);
    . . . temp = parent->Right;
    parent->Right = node;
    node->Right = temp;
    node->Left = parent;
    node->RightTag = parent->RightTag;
    parent->RightTag = false;
    node->LeftTag = true;
```

```
for (index = 1; index <= dxt;
index++)
{
    x++;
    if (e >= 0) e+ = t+ca;
    else
    {
        y--;
        e+ = t-s+cd;
        s+ = dst;
    }
    t- = dst;
    Пиксель(xc+x,yc+y);
    Пиксель(xc+y,yc+x);
    Пиксель(xc+y,yc-x);
    Пиксель(xc+x,yc-y);
    Пиксель(xc-x,yc-y);
    Пиксель(xc-y,yc-x);
    Пиксель(xc-y,yc+x);
    Пиксель(xc-x,yc+y);
}
}
```

Инкрементные алгоритмы

эллипс

```
Void Ellipse(int xc,int yc,int eux,int euy)
{
    Head->LeftTag = true;
    int x,y;
    long a2,b2,dds,ddt,dxt,t,s,e,ca,cd,index;
    int a,b;
    a = ads(eux-xc);
    b = ads(euy-yc);
    a2 = (long)a*(long)a;
    b2 = (long)b*(long)b;
    dds = 4*a2;
    ddt = 4*b2;
    dxt = (float)a2/sqrt(a2+b2);
    t = 0;
    s = -4*a2*b;
    e = (-s/2)-2*b2-a2;
    ca = -6*b2;
    cd = ca-4*a2;
    x = xc;
    y = yc+b;
    Пиксель(x,y);
    Пиксель(x,2*yc-y);
    Пиксель(2*xc-x,2*yc-y);
    Пиксель(2*xc-x,y);
    for (index = 1; index <= dxt; index++)
    {
        x++;
        if (e >= 0) e+ = t+ca;
        else
        {
            e+ = t-s+cd;
            s+ = dds;
            t- = ddt;
            Пиксель(x,y);
            Пиксель(x,2*yc-y);
            Пиксель(2*xc-x,2*yc-y);
            Пиксель(2*xc-x,y);
        }
    }
}
```

```
else {y--;
    e+ = t-s+cd;
    s+ = dds; }
t- = ddt;
Пиксель(x,y);
Пиксель(x,2*yc-y);
Пиксель(2*xc-x,2*yc-y);
Пиксель(2*xc-x,y);
}
dxt = abs(y-yc);
e- = t/2+s/2+b2+a2;
ca = -6*a2;
cd = ca-4*b2;
for (index = 1; index <= dxt; index++)
{
    y--;
    if (e <= 0) e+ = -s+ca;
    else
    {
        x++;
        e+ = t-s+cd;
        t- = ddt;
    }
    s+ = dds;
    Пиксель(x,y);
    Пиксель(x,2*yc-y);
    Пиксель(2*xc-x,2*yc-y);
    Пиксель(2*xc-x,y);
}
}
```


Кривая Безье

Разработана математиком Пьером Безье.

В параметрической записи имеет вид:

$x = P_x(t)$, где P_x, P_y – многочлены

Безье:

$y = P_y(t)$;

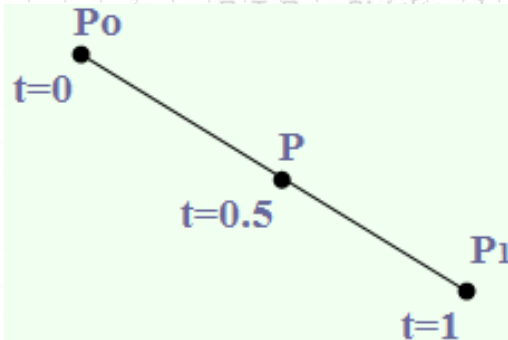
где x_i, y_i – координаты точек-ориентиров,
 m – степень полинома (на 1 меньше количества

точек)

Классифицируем по m :

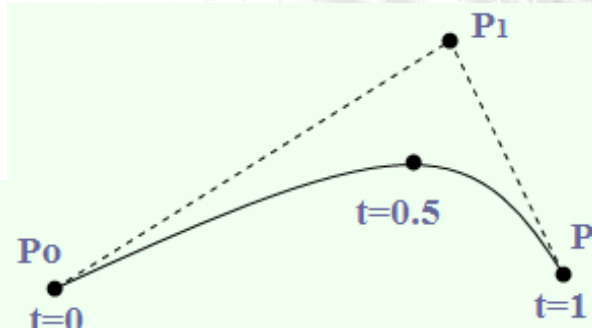
$m = 1$ (по двум точкам) –
 отрезок прямой с
 концами P_0, P_1 .

$P(t) = (1 - t) P_0 + t P_1$

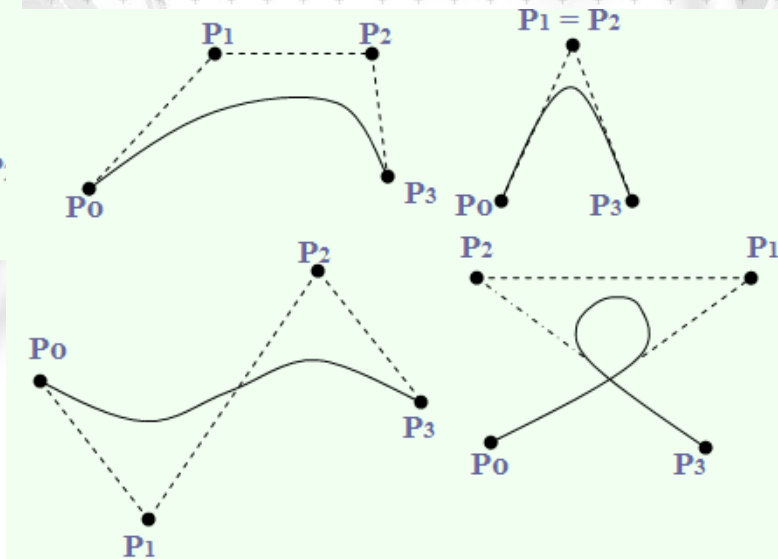


$m = 2$ (по трем точкам)

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



$m = 3$ (по четырем точкам)



Геометрический алгоритм для кривой Безье

Определение координат (x,y) по известному t.

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

Точки деления соединяются отрезками прямых и образуют новый многоугольник. Количество узлов нового контура на 1 меньше значения m.

Стороны нового многоугольника снова делятся на t. И так далее, до тех пор, пока не получится одна точка. Это и есть точка кривой Безье.

```
for (i = 0; i <= m; i++)
```

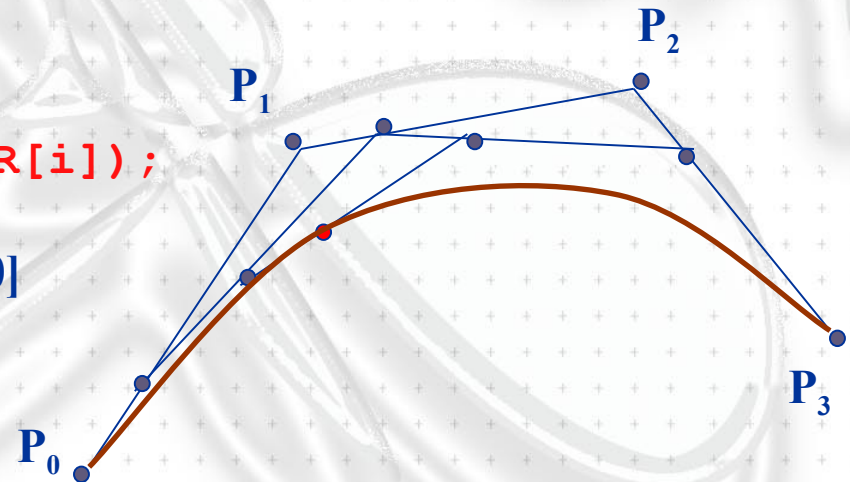
```
    R[i] = P[i];
```

```
    for (j = m; j > 0; j--)
```

```
        for (i = 0; i < j; i++)
```

```
            R[i] = R[i] + t * (R[i+1] - R[i]);
```

```
// Координаты точки Безье запишутся в R[0]
```



Алгоритм вывода фигур

Здесь фигура – плоский геометрический объект, который состоит из линий контура и точек внутри контура. Контур может быть составным.

Геометрический вывод фигур:

2. Вывод контура;
3. Вывод точек заполнения.

Точки заполнения:

5. Алгоритмы закрашивания от внутренней точки к границам произвольного контура;
2. Алгоритмы, использующие математическое описание контура.

Простейший алгоритм закрашивания:

Шаг 1. Определить x_0, y_0 .

Шаг 2. Выполнить функцию закрашивания (x_0, y_0) .

Функция закрашивания (x, y)

{ Если цвет пикселя (x, y) не равен цвету границы, то

{ Установить для пикселя (x, y) цвет заполнения;

Закрасить $(x+1, y)$;

Закрасить $(x-1, y)$;

Закрасить $(x, y+1)$;

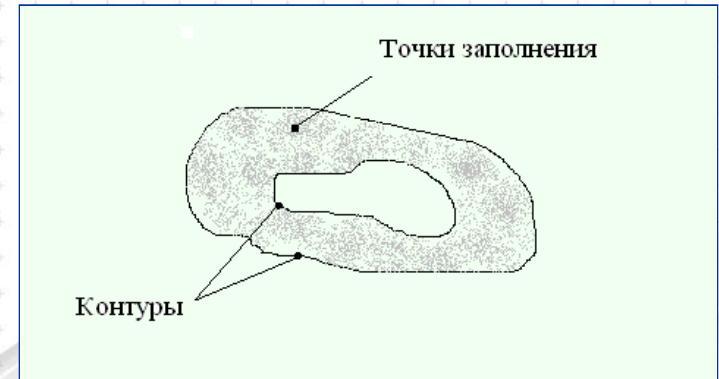
Закрасить $(x, y-1)$;

}

} parent->RightTag = parent->RightTag;

} parent->RightTag = false;

node->LeftTag = true;



Алгоритм закрашивания линиями

Вместо одного пикселя закрашивается целая прямая (горизонтальная линия).

```
int LineFill(int x, int y, int dir, int preXL, int preXR)
{
    int xl = x, xr = x;
    COLORREF clr;
    do clr = GetPixel(udc, --xl, y); while (clr != BORDER);
    do clr = GetPixel(udc, ++xl, y); while (clr != BORDER);
    xl++;
    xr--;
    MoveToEx(udc, xl, y, NULL);
    LineTo(udc, xr+1, y);
    for (x = xl; x <= xr; x++)
    {
        clr = GetPixel(udc, x, y+dir);
        if (clr != BORDER) x = LineFill(x, y+dir, dir, xl, xr);
    }
    for (x = PreXL; x < PreXL; x++)
    {
        clr = GetPixel(udc, x, y-dir);
        if (clr != BORDER) x = LineFill(x, y-dir, -dir, xl, xr);
    }
    for (x = PreXR; xl; x < xr; x++)
    {
        clr = GetPixel(udc, x, y-dir);
        if (clr != BORDER) x = LineFill(x, y-dir, -dir, xl, xr);
    }
    return xr;
}
```

Алгоритмы заполнения, которые используют математическое описание контура

Математическим описанием может служить уравнение $y = f(x)$ для окружности, эллипса и другой кривой

3. Заполнение прямоугольника

Пусть заданы координаты $(x1, y1)$ – левая верхняя, $(x2, y2)$ – правая нижняя.

```
for (y = y1; y <= y2; y++)
```

```
{ node = ToCell[x1];
```

Рисуем горизонтальную линию с координатами $(x1, y)$ – $(x2, y)$.

```
} node->Left = temp;
```

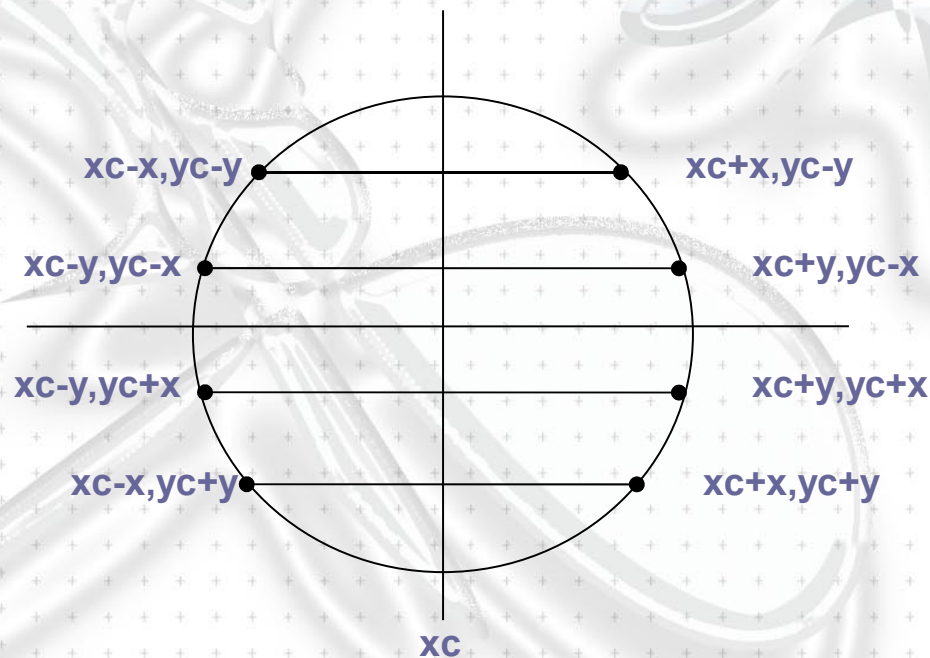
```
node->Right = parent;
```

2. Заполнение круга

Используем алгоритм вывода контура.

Там вычисляются координаты пикселей контура в границах одного.

Нужно соединить пары точек на контуре, расположенные симметрично относительно оси y .



Алгоритмы заполнения, которые используют математическое описание контура

3. Заполнение полигона

Закрашиваем полигон отрезками прямых линий (горизонтали).

1. Найти $\min\{y_i\}$, $\max\{y_i\}$ среди всех вершин P_i .

2. Выполнить цикл по y от y_{\min} до y_{\max}

{

3. Нахождение точек пересечения всех отрезков контура с горизонталью y .

Координаты x_j точек записать в массив.

4. Сортировка массива $\{x_j\}$ по возрастанию x .

5. Вывод горизонтальных отрезков

$(x_0, y) - (x_1, y)$

$(x_2, y) - (x_3, y)$

$(x_0, y) - (x_1, y)$

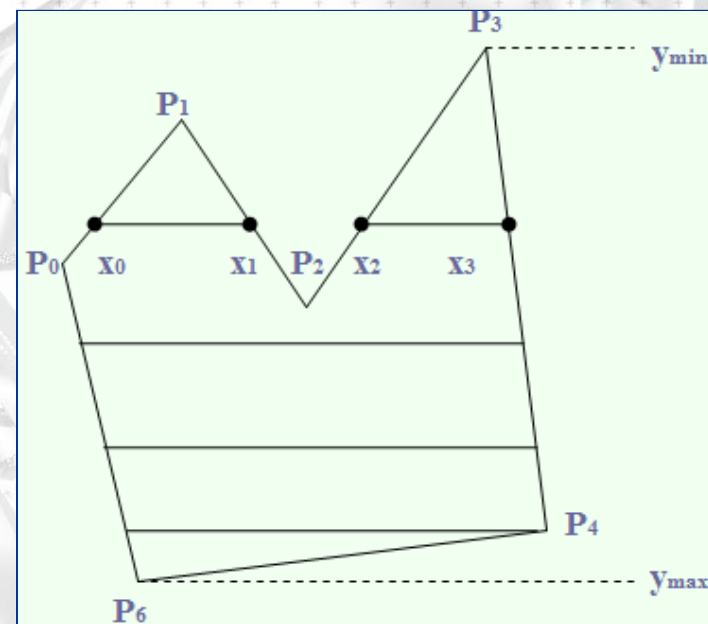
Свойство топологии контура фигуры:

* любая прямая пересекает любой замкнутый контур четное число раз.

* если горизонталь y совпадает с y_i вершины P_i , тогда анализируем точку:

а) если горизонталь пересекает контур (P_0, P_4), то в массив запишем одну точку;

б) если горизонталь касается точки (P_1, P_2, P_3, P_5), тогда, либо запишем две, либо ни одной.



Наложение текстур на полигон

Общий алгоритм заполнения:

Пробегаем все точки полигона (x,y).

Для каждой точки вычисляем координаты текстуры (xт.ут).

По координатам текстуры определяем цвет пикселя.

Преобразование координат (x,y) => (xт.ут)

имеет аффинный вид:

Для трех точек укажем соответствие

$$\begin{cases} x_{Ti} = A \cdot x_i + B \cdot y_i + C; \\ y_{Ti} = D \cdot x_i + E \cdot y_i + F; \end{cases}$$

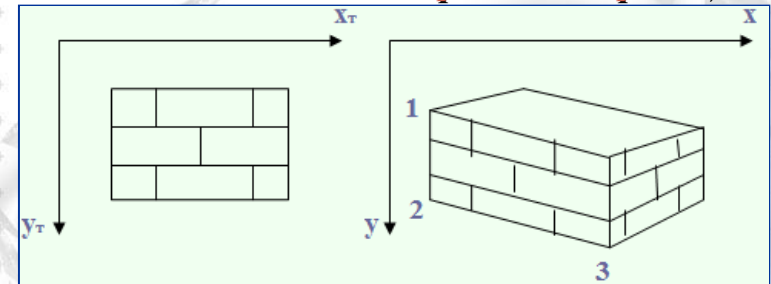
Пусть $x_{Ti} = X_i$, $y_{Ti} = Y_i$. Получаем систему

$$\begin{cases} X_1 = A \cdot x_1 + B \cdot y_1 + C; \\ X_2 = A \cdot x_2 + B \cdot y_2 + C; \\ X_3 = A \cdot x_3 + B \cdot y_3 + C; \\ \text{и} \\ Y_1 = D \cdot x_1 + E \cdot y_1 + F; \\ Y_2 = D \cdot x_2 + E \cdot y_2 + F; \\ Y_3 = D \cdot x_3 + E \cdot y_3 + F; \end{cases}$$

Методом Крамера находим

$$\begin{cases} A = (\det A) / (\det t); \\ B = (\det B) / (\det t); \\ C = (\det C) / (\det t); \end{cases}$$

Аксонетрическая проекция



$$\det A = \begin{vmatrix} X_1 & y_1 & 1 \\ X_2 & y_2 & 1 \\ X_3 & y_3 & 1 \end{vmatrix}$$

$$\det t = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

$$\det B = \begin{vmatrix} X_1 & X_1 & 1 \\ X_2 & X_2 & 1 \\ X_3 & X_3 & 1 \end{vmatrix}$$

$$\det C = \begin{vmatrix} X_1 & y_1 & X_1 \\ X_2 & y_2 & X_2 \\ X_3 & y_3 & X_3 \end{vmatrix}$$

Аналогично
вычисляем D,E,F.

Наложение текстур на полигон

Перспективная

проекция

Цепочка преобразований:

Мировые
координаты

x, y, z

Видовые
координаты

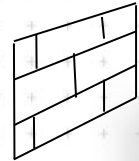
Z, Y, Z

Проецирование

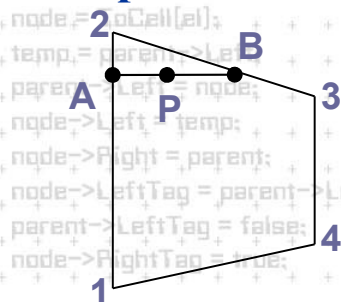
X_n, Y_n, Z_n

Экранные
координаты

$X_э, Y_э, Z_э$



Рассмотрим вывод перспективной проекции полигона с текстурой



AB - горизонталь

1,2,3,4 – вершины полигона (заданы
экранными координатами)

P – произвольная точка полигона

Базовая операция: по известным видовым координатам концов отрезка находим видовые координаты точки отрезка, заданной координатами в плоскости проецирования.

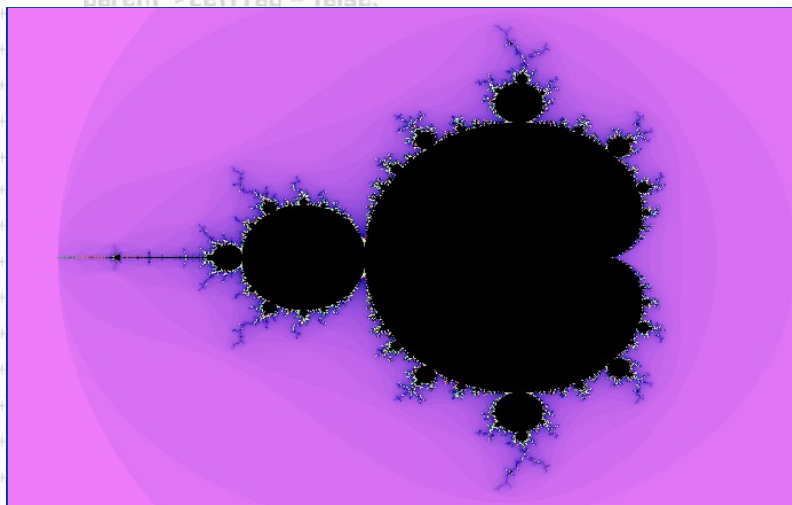
Видовые координаты точки A(X,Y,Z) по известным видовым координатам точек 1 и 2 - (X_1, Y_1, Z_1) и (X_2, Y_2, Z_2) :

$$\frac{X - X_1}{X_2 - X_1} = \frac{Y - Y_1}{Y_2 - Y_1} = \frac{Z - Z_1}{Z_2 - Z_1}$$

Фрактал можно определить как объект довольно сложной формы, получающийся в результате выполнения простого итерационного цикла.

Итерационность (рекурсивность) обуславливает такие свойства фракталов как «самоподобие». Отдельные части похожи по форме на весь фрактал в целом. В 1975 году французский математик Бенуа Мандельброт издал книгу «The fractal. Geometry of Nature».

Фрактал Мандельброта



Для каждого изображения необходимо выполнить цикл итераций

$$Z_{k+1} = Z_k^2 + Z_0, \text{ где } k = 0, 1, 2, \dots$$

$Z_k = X_k + i Y_k$ – комплексное число

X_0, Y_0 – стартовые координаты

Для каждой точки выполняется ограниченное число итераций: до тех пор, пока $|Z_k| \leq 2$.

$$Z_k^2 = (X^2 - Y^2) + i (2XY)$$

Фракталы

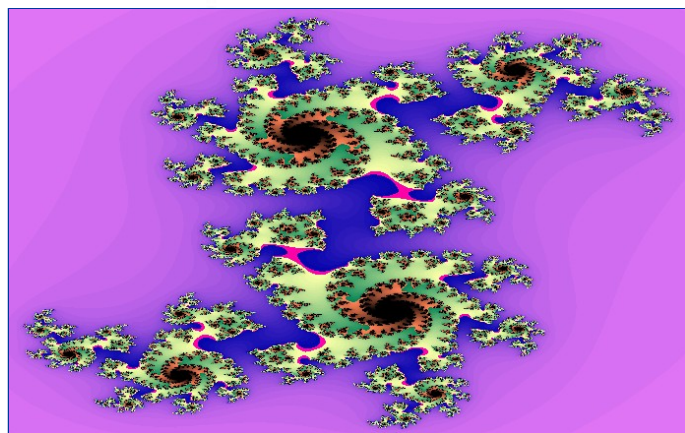
```
#include <class T> Binary<T>::Binary()
```

Фракталы Джулия и Ньютона

```
Head = new Cell;
```

Фрактал Джулия

```
Head->RightTag = true;
```



```
parent->LeftTag = false;
node->RightTag = true;
```

```
if(!node->LeftTag){
```

$$Z_{k+1} = Z_k^2 + C,$$

где C – комплексная константа

```
template <class T> void parent, T el)
```

Условие завершения: $|Z_k| \leq 2$

```
temp = parent->Right;
```

```
parent->Right = node;
```

```
node->Right = temp;
```

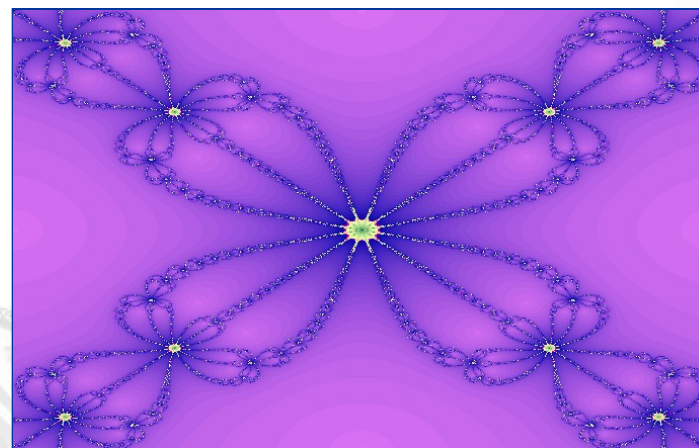
```
node->Left = parent;
```

```
node->RightTag = parent->RightTag;
```

```
parent->RightTag = false;
```

```
node->LeftTag = true;
```

Фрактал Ньютона



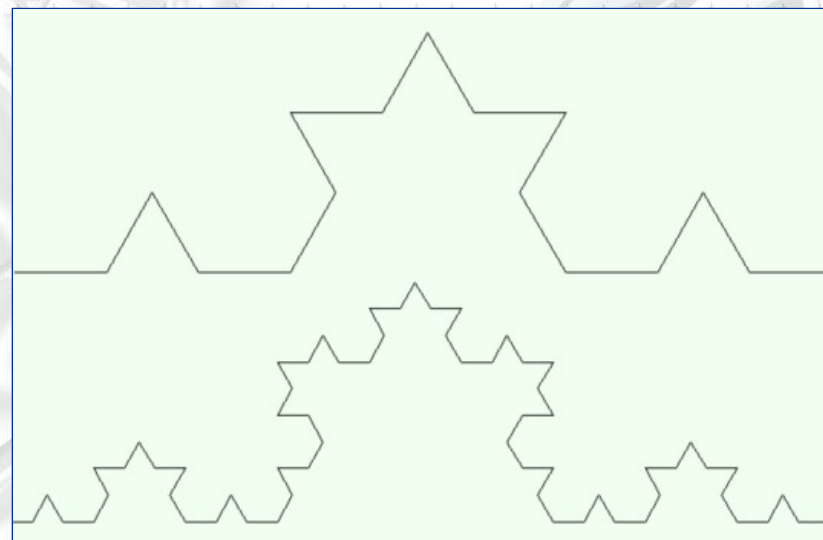
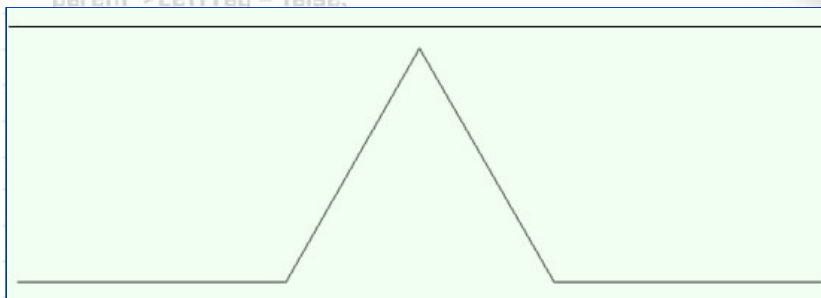
$$Z_{k+1} = \frac{3Z_k^4 + 1}{4Z_k^3}$$

$Z_0 = X_0 + i Y_0$ – координата
точки

Условие завершения: $|Z_k| \rightarrow 0$

Они получаются в результате
последовательность простых
геометрических преобразований.

Кривая Коха



Фракталы

Фракталы, формируемые системой итеративных функций

$$\begin{cases} X_{k+1} = F_x(X_k, Y_k); \\ Y_{k+1} = F_y(X_k, Y_k); \end{cases}$$

Зададим координаты опорных точек 1 и 2.

Находим точку 3. Она образуется при повороте точки 2 на угол α , относительно точки 1.

$$X_3 = (X_1 - X_2) \cos \alpha - (Y_2 - Y_1) \sin \alpha + X_1;$$

$$Y_3 = (X_1 - X_2) \sin \alpha + (Y_2 - Y_1) \cos \alpha + Y_1;$$

Находим точку 4. Она находится на отрезке 1-3.

Пусть $(1-4)/(1-3) = k$, причем $0 < k < 1$.

$$X_4 = X_1(1 - k) + X_3 k;$$

$$Y_4 = Y_1(1 - k) + Y_3 k;$$

Зададим длину и угол наклона ветвей, которые выходят из точки 4.

Найдем координату точки 5. Пусть соотношение отрезков $(4-5)/(5-3) = k_1$. Тогда:

$$X_5 = X_4(1 - k_1) + X_3 k_1;$$

$$Y_5 = Y_4(1 - k_1) + Y_3 k_1;$$

Точки 6 и 7 – это точка 5, повернутая относительно точки 4 на углы β и $-\beta$ соответственно:

$$X_6 = (X_5 - X_4) \cos \beta - (Y_5 - Y_4) \sin \beta + X_4;$$

$$Y_6 = (X_5 - X_4) \sin \beta + (Y_5 - Y_4) \cos \beta + Y_4;$$

$$X_7 = (X_5 - X_4) \cos \beta + (Y_5 - Y_4) \sin \beta + X_4;$$

$$Y_7 = - (X_5 - X_4) \sin \beta + (Y_5 - Y_4) \cos \beta + Y_4.$$

Кроме расчетов опорных точек на каждом шаге будем рисовать один отрезок 1-4. Также будем менять цвет и толщину в зависимости от номера итерации. Затем повторяем цикл рекурсии для отрезков 4-3, 4-6, 4-7.

Веточка папоротника

