Предварительные замечания:

1. Описание алгоритма состоит из описания пяти нижеприведенных процедур (пятая процедура — дополнительная, на которую нет прямой ссылки в алгоритме).

2. Предполагается, что к началу исполнения алгоритма был произведен переход к системе координат наблюдателя и выполнены все манипуляции с изображением (т. е. ко всем точкам применено преобразование $V \cdot T$, описанное в методических рекомендациях к задаче 9, но не применено преобразование $U$ ). Слова «Начертить многоугольник» в алгоритме DrawBSPTree означают, что здесь нужно перейти к экранной системе координат (преобразование $U$), отбросить третью координату, отсечь многоугольник (2-мерным алгоритмом) и изобразить то, что получилось.

3. При сравнении значений координаты $y$ на равенство следует округлять эти значения до целого числа.

4. Предполагается, что каждый многоугольник $P_i$ в списке $\mathscr{P}$ представлен семеркой $(L, n, C, a, b, c, d)$, где $L$ — список вершин многоугольника при обходе в некотором порядке, $n$ — количество вершин, $C$ — цвет многоугольника, $a$, $b$, $c$, $d$ — коэффициенты уравнения несущей плоскости (для их вычисления можно использовать приведенный ниже пятый алгоритм).

5. Для изображения многоугольника в программе на Visual C++ желательно использовать процедуру `Graphics::FillPolygon(SolidBrush^, array<PointF>^)`.

6. Непосредственно перед изображением должно производиться отсечение многоугольника относительно области видимости. Для отсечения должен использоваться алгоритм, реализованный при выполнении задания 7.

---

**Алгоритм 1:** Алгоритм Художника с использованием BSP-деревьев

---

**Вход**: $\mathscr{P}$ — список многоугольников трехмерной сцены
**начало алгоритма**
  Присвоить $T$ пустое дерево;
  **цикл для каждого** $P$ *в* $\mathscr{P}$ **выполнить**
    Выполнить процедуру PutPolygonToBSP($P, T$);
  Выполнить процедуру DrawBSPTree($T$);
**конец алгоритма**

---

**Алгоритм 2:** PUTPOLYGONTOBSP Добавление многоугольника в BSP-дерево

**Вход**: $P$ — многоугольник, $T$ — бинарное дерево многоугольников.

**Выход**: Измененное бинарное дерево многоугольников $T$ с внесенным в него многоугольником $P$.

**начало алгоритма**

  **если** $T$ — *пустое дерево* **то**

    · Создать вершину дерева с пустыми левым и правым поддеревьями. Присвоить её переменной $T$;

    · В соданную вершину записать $P$; **закончить алгоритм**;

  **иначе**

    · Пусть $P_T$ — многоугольник в корне дерева $T$;

    · Выполнить процедуру VERTEXCOUNT$(P, P_T)$ для подсчета величин $PosCount$, $NegCount$ и получения многоугольников $P_{pos}$ и $P_{neg}$;

    **если** $PosCount > 0$ **то**

      Выполнить PUTPOLYGONTOBSP$(P_{pos}, T \to Left)$. Результат присвоить $T$;

    **если** $NegCount > 0$ *или* $PosCount = 0$ **то**

      Выполнить PUTPOLYGONTOBSP$(P_{neg}, T \to Right)$. Результат присвоить $T$;

**конец алгоритма**

**Алгоритм 3:** VERTEXCOUNT

**Вход**: $P, P_T$ — многоугольники.

**Выход**: $PosCount$, $NegCount$ — количество вершин многоугольника $P$, расположенных, соответственно, в положительном и отрицательном полупространстве относительно несущей плоскости многоугольника $P_T$; $P_{pos}$, $P_{neg}$ — многоугольники, результат разбиения многоугольника $P$ несущей плоскостью многоугольника $P_T$.

**начало алгоритма**

- Пусть $(L_P, n, C, a, b, c, d)$ — пятерка, представляющая многоугольник $P$, $(L_T, n_T, C_T, a_T, b_T, c_T, d_T)$ пятерка, представляющая многоугольник $P_T$;
- Присвоить $PosCount = 0$; $NegCount = 0$; $L_{pos} = \varnothing$; $L_{neg} = \varnothing$;
- Положить $A = L_P[n]$; $intersectionVertex = False$; $ZeroCount = 0$;
- Вычислить $SpacePartA = A_x \cdot a_T + A_y \cdot b_T + A_z \cdot c_T + d_T$; $ActiveSign = signum(SpacePartA)$;

**если** $ActiveSign = 0$ **то** $intersectionVertex = True$;

**цикл для** $i$ *от* $1$ *до* $n$ **выполнять**

- $B = L_P[i]$;
- Вычислить $SpacePartB = B_x \cdot a_T + B_y \cdot b_T + B_z \cdot c_T + d_T$; $NewSign = signum(SpacePartB)$;

**если** $NewSign = 0$ **то**

- Добавить вершину $B$ в списки $L_{pos}$ и $L_{neg}$;
- $ZeroCount = ZeroCount + 1$;
- $intersectionVertex = True$;

**иначе**

**если** $NewSign \neq ActiveSign$ **то**

**если** $!intersectionVertex$ **то**

- Вычислить координаты точки пересечения $D$:
  $t = SpacePartA/(SpacePartA - SpacePartB)$
  $D_x = A_x + (B_x - A_x)t$;
  $D_y = A_y + (B_y - A_y)t$;
  $D_z = A_z + (B_z - A_z)t$;
- Добавить вершину $D$ в списки $L_{pos}$ и $L_{neg}$;
- $ZeroCount = ZeroCount + 1$;

- Присвоить $ActiveSign = NewSign$; $intersectionVertex = False$;

**если** $NewSign > 0$ **то**

- Добавить вершину $B$ в список $L_{pos}$ ;
- $PosCount = PosCount + 1$;

**иначе**

- Добавить вершину $B$ в список $L_{neg}$ ;
- $NegCount = NegCount + 1$;

- $A = B$; $SpacePartA = SpacePartB$;

- Присвоить
  $P_{pos} = (L_{pos}, PosCount + ZeroCount, C, a, b, c, d)$;
  $P_{neg} = (L_{neg}, NegCount + ZeroCount, C, a, b, c, d)$;

**конец алгоритма**

**Алгоритм 4:** DrawBSPTree Изображение многоугольников в порядке от наиболее удаленного до наблюдателя

**Вход**: $T$ — BSP-дерево многоугольников.

**начало алгоритма**

**если** $T$ — *пустое дерево* **то закончить алгоритм**;

Пусть $(L_P, n, C, a, b, c, d)$ — семерка, для многоугольника $P$ в корне дерева $T$;

**если** $d < 0$ **то**

· Выполнить процедуру DrawBSPTree$(T \to Right)$;

· Начертить многоугольник $P$;

· Выполнить процедуру DrawBSPTree$(T \to Left)$;

**иначе**

· Выполнить процедуру DrawBSPTree$(T \to Left)$;

· Начертить многоугольник $P$;

· Выполнить процедуру DrawBSPTree$(T \to Right)$;

**конец алгоритма**

**Алгоритм 5:** InitializePolygon Вычисление коэффициентов уравнения плоскости

**Вход**: $P$ — список вершин многоугольника в трехмерном пространстве

**Выход**: $a$, $b$, $c$, $d$ — коэффициенты уравнения несущей плоскости

**начало алгоритма**

Пусть $(x_1, y_1, z_1)$, $(x_2, y_2, z_2)$, $(x_3, y_3, z_3)$ — первые три вершины в списке $P$. Предполагаем, что эти вершины не лежат на одной прямой. Тогда

$$a = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix}; \quad b = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}; \quad c = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}; \quad d = -\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}.$$

**конец алгоритма**