Лабораторная работа №5

Алгоритмы удаления невидимых поверхностей

Цель работы: изучить алгоритмы удаления невидимых поверхностей и создать программу для визуализации объёмной трёхмерной модели с закрашенными гранями.

Порядок выполнения работы

Разработать алгоритм и составить программу для построения на экране трёхмерной модели с закрашенными гранями в соответствии с номером варианта лабораторной работы №4.

Требования к программе

1. В программе по центру окна должна отображаться только центральная или ортографическая проекция фигуры по выбору пользователя. Грани фигуры должны быть закрашены и отсортированы по глубине, т.е. пользователь должен видеть передние грани. Перед растеризацией грани должны быть отсортированы по алгоритму художника (на пятёрку) или по глубине (на четвёрку). Сортировку методом z-буфера, которая уже реализована в предыдущей лабораторной работе, временно отключить.

2. Пользователь должен иметь возможность поворачивать и перемещать фигуру вдоль оси z с использованием мыши. Поворот фигуры лучше выполнять при обработке события WM_MOUSEMOVE при зажатой кнопке мыши, перенос – при обработке события WM_MOUSEWHEEL. Фигура должна поворачиваться «вслед» за мышью. Это значит, что, если мышь перемещается влево-вправо, то фигура должна поворачиваться вокруг вертикальной оси; если вверх-вниз, то вокруг горизонтальной оси.

3. В программе должна быть предусмотрена возможность изменять прозрачность граней с помощью клавиш или мыши.

4. Наложить на одну или несколько граней объекта шейдер с какой-либо текстурой (кирпичная кладка, сетка, горошек, пчелиные соты или любая другая). Для проверки правильности работы растеризатора наложить на одну отдельную квадратную грань шахматную текстуру. Текстура не должна искажаться перспективой.

Краткие теоретические сведения

Алгоритмы удаления невидимых поверхностей – набор алгоритмов, которые используются для определения видимости как отдельных граней или их частей, так и сложных объектов в целом. У таких алгоритмов две основные цели:

1. Исключить из растеризации невидимые объекты,
2. Определить порядок перекрытия граней относительно наблюдателя.

В данной работе будем достигать только вторую цель. Перечислим алгоритмы, которые требуется изучить:

1. Алгоритм сортировки по глубине.
2. Алгоритм методом z-буфера.
3. Алгоритм художника.

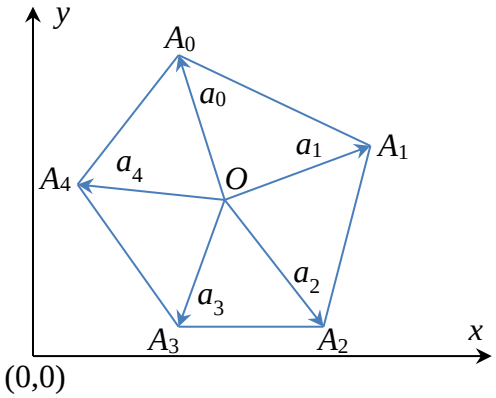
Алгоритм сортировки методом z-буфера является самым естественным для растровой графики и очень простым. Он реализован нами в предыдущей лабораторной работе.

Рассмотрим достоинства и недостатки этих методов:

	Достоинства	Недостатки
Алгоритм сортировки по глубине	Высокая скорость сортировки и простота реализации. Временная сложность равна сложности базового алгоритма сортировки и в лучшем случае равна $O(n \cdot \log_2 n)$	Неправильно сортирует, если грани расположены близко, вытянуты относительно друг друга и сильно отличаются по площади.
Алгоритм методом z-буфера	Имеет линейную алгоритмическую сложность, очень прост в реализации. Реализуется аппаратно на графических процессорах. Выполняется «на лету» во время растеризации.	Не учитывает геометрическое расположение фигур и растеризует невидимые объекты. Даже если грань расположена внутри пирамиды видимости, но не видна, она всё равно будет растеризована, т.е. потратит процессорное время. Не годится для рисования полупрозрачных граней.
Алгоритм художника	Сортирует геометрически правильно за исключением нескольких редких случаев (перекрёстное перекрытие). Позволяет сразу отбросить грани, которые полностью перекрываются другими гранями. Справляется с полупрозрачными гранями.	Большая временная сложность. При использовании дополнительной памяти размером n^2 ячеек сложность равна $O(n^2)$. Без использования дополнительной памяти алгоритмическая сложность равна $O(n^3)$.

Для создания алгоритмов удаления невидимых поверхностей рассмотрим несколько вспомогательных алгоритмов.

Определение нахождения точки внутри выпуклого многоугольника



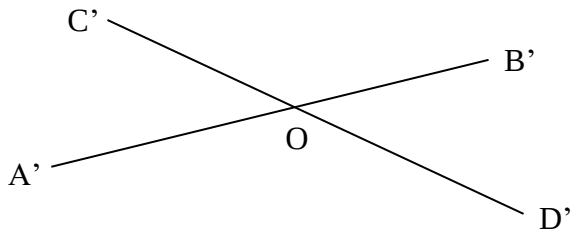
Пусть вершины многоугольника A_i $(i = 0, \dots, n-1)$ предварительно упорядочены по или против часовой стрелки. Определим, находится ли точка O внутри многоугольника. Направим из точки O вектора $\vec{a}_i$ к вершинам многоугольника A_i. Для определения взаимного расположения векторов рассмотрим *векторное произведение* двух произвольных векторов $\vec{a}$ и $\vec{b}$:

$$\vec{c} = \vec{a} \times \vec{b} = \left(a_y b_z - b_y a_z, a_z b_x - b_z a_x, a_x b_y - b_x a_y\right).$$

Достаточно рассматривать только третью компоненту c_z. Если вектор $\vec{b}$ находится правее вектора $\vec{a}$ в порядке обхода по часовой стрелке, то значение $c_z = a_x b_y - b_x a_y$ будет отрицательным, если левее – положительным. Если точка O находится внутри выпуклого многоугольника, то для любого i $(i = 0, \dots, n-2)$ выполняется условие $(\vec{a}_i \times \vec{a}_{i+1})_z < 0$ и $(\vec{a}_{n-1} \times \vec{a}_0)_z < 0$ для последней пары вершин. Для проверки нахождения произвольной точки внутри многоугольника необходимо для каждой вершины A_i, за исключением последней, вычислить векторное произведение $\vec{a}_i \times \vec{a}_{i+1}$. Для последней вершины нужно находить произведение $\vec{a}_{n-1} \times \vec{a}_0$ (на рисунке $\vec{a}_4 \times \vec{a}_0$). Для того, чтобы не рассматривать отдельно последнюю пару вершин, можно записать единое условие для всех вершин: $\left(\vec{a}_i \times \vec{a}_{(i+1) \bmod n}\right)_z < 0$. Если все компоненты z рассмотренных векторных произведений будут иметь одинаковый знак, то точка O находится внутри многоугольника.

Алгоритм художника

Прежде чем создать алгоритм сортировки граней трёхмерных фигур по глубине, рассмотрим более простой случай. Пусть в трёхмерном пространстве расположены два отрезка AB и CD, проекции которых на экранную (картинную) плоскость пересекаются. Необходимо определить, какой из них расположен ближе к экранной плоскости. При описании алгоритмов будем считать, что экранная плоскость имеет уравнение $z = 0$, а ось z направлена вглубь экрана. Во всех описанных ниже алгоритмах сортировки предполагается, что используемые координаты являются готовыми к растеризации. Т.е. это те экранные координаты (однородные или нормализованные), которые будут переданы в функцию растеризации. В принципе, можно использовать координаты, полученные после выполнения всех аффинных преобразований до этапа проектирования. На результат сортировки это не повлияет.



Проекции отрезков AB и CD на плоскости z=0

Обозначим проекции отрезков AB и CD на экранную плоскость как A’B’ и C’D’, а точку пересечения проекции отрезков на экранной плоскости как O. Проекции точек отличаются только тем, что они имеют координату z, равную нулю. Координаты точки O определяются в результате решения системы линейных уравнений:

$$\frac{O_x - A_x}{B_x - A_x} = \frac{O_y - A_y}{B_y - A_y}, \qquad \frac{O_x - C_x}{D_x - C_x} = \frac{O_y - C_y}{D_y - C_y}.$$

Если записанные пропорции в левой и правой части находятся в диапазоне от нуля до единицы, то отрезки A’B’ и C’D’ пересекаются.

Вернём координату z и рассмотрим отрезки AB и CD в пространстве. Необходимо найти точки пересечения прямой z*, выходящей из точки O на экране в направлении оси z с отрезками AB и CD в пространстве. Та точка пересечения, которая имеет меньшую координату z, расположена ближе к экранной плоскости. Обозначим точку пересечения прямой z* и AB как (O_x, O_y, z_{AB}).

Построим уравнение прямой в пространстве, на которой лежит отрезок AB, и подставим в него координаты точки (O_x, O_y, z_{AB}):

$$\frac{O_x - A_x}{B_x - A_x} = \frac{O_y - A_y}{B_y - A_y} = \frac{z_{AB} - A_z}{B_z - A_z}.$$

В данном уравнении неизвестная – z_{AB}. Для её нахождения необходимо сначала определить, какой знаменатель больше по модулю. Если $|B_x - A_x| > |B_y - A_y|$, то находить решение системы:

$$\frac{O_x - A_x}{B_x - A_x} = \frac{\boldsymbol{z_{AB}} - A_z}{B_z - A_z}.$$

Иначе, решать систему

$$\frac{O_y - A_y}{B_y - A_y} = \frac{\boldsymbol{z_{AB}} - A_z}{B_z - A_z}.$$

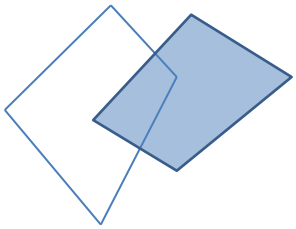
Аналогично нужно определить удалённость точки пересечения отрезка *CD* и прямой z* как решение системы:

$$\frac{O_x - C_x}{D_x - C_x} = \frac{O_y - C_y}{D_y - C_y} = \frac{\boldsymbol{z_{CD}} - C_z}{D_z - C_z}.$$

Если $z_{AB} < z_{CD}$, то отрезок *AB* перекрывает *CD* с точки зрения наблюдателя, т.е. *AB* расположен ближе, чем *CD*.

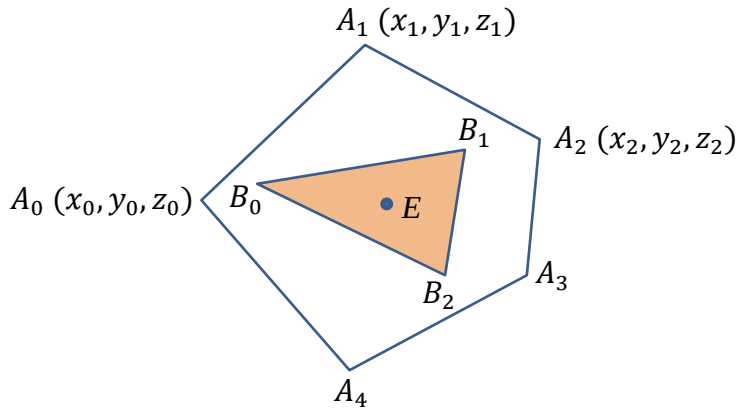
Рассмотрим более сложную задачу. Предположим, необходимо отсортировать по глубине два многоугольника. Возможны следующие случаи взаимного расположения проекций данных многоугольников:

1. Стороны многоугольников пересекаются.



В этом случае одна из пар сторон двух многоугольников обязательно пересекается. В общем случае нужно рассмотреть каждую пару отрезков (сторон) двух многоугольников и найти хотя бы одну пересекающуюся. Далее, используя вышеописанный алгоритм, не трудно определить какой отрезок (и соответственно многоугольник, к которому он принадлежит) расположен ближе к экранной плоскости.

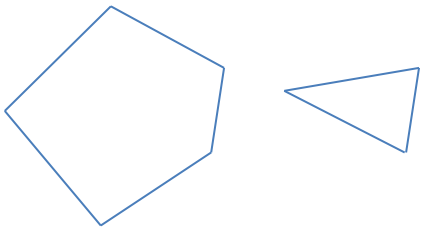
2. Один из многоугольников находится внутри другого.



Для обнаружения этого случая нужно проверить каждую вершину первого многоугольника на расположение внутри второго, используя свойство векторного произведения векторов. В таком случае все вершины первого многоугольника находятся внутри второго. Порядок следования граней можно определить следующим образом:

1. Найти центр тяжести E первого многоугольника. Он определяется как среднее арифметическое отдельных координат вершин B_i многоугольника в пространстве:
$$E_x = \frac{1}{n}\sum_{i=0}^{n-1} B_{ix}\,, \qquad E_y = \frac{1}{n}\sum_{i=0}^{n-1} B_{iy}\,, \qquad E_z = \frac{1}{n}\sum_{i=0}^{n-1} B_{iz}.$$
2. Найти уравнение плоскости (например, по любым трём точкам), которой принадлежит второй многоугольник в виде $Ax + By + Cz + D = 0$.
3. Найти точку пересечения прямой в пространстве $\frac{x}{E_x} = \frac{y}{E_y} = 1$ с плоскостью $Ax + By + Cz + D = 0$. Найти z, полученное из решения уравнения $AE_x + BE_y + Cz + D = 0$. Это равносильно решению линейного уравнения с одной неизвестной z:
$$\begin{vmatrix} E_x - x_0 & E_y - y_0 & z - z_0 \\ x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \end{vmatrix} = 0.$$
4. Если $E_z < z$, то первый многоугольник расположен ближе к экранной плоскости, иначе – дальше.

3. Проекции многоугольников не пересекаются.



В данном случае порядок рисования многоугольников не важен.

Общий алгоритм художника. Используя рассмотренные ранее выражения для определения порядка видимости граней, построим общий алгоритм сортировки. Пусть на сцене необходимо отрисовать m граней $G_i (i = 0, \ldots, m-1)$, каждая из которых является

выпуклым многоугольником. Перед сортировкой граней нужно создать матрицу H размером $m \times m$ и заполнить её следующим образом:

если грань G_i перекрывает грань G_j, то $H_{ij} = 1, H_{ji} = -1$,

если грань G_j перекрывает грань G_i, то $H_{ij} = -1, H_{ji} = 1$,

если грани G_i и G_j никак не перекрываются то $H_{ij} = 0, H_{ji} = 0$.

По матрице H очень просто определять грани, которые находятся дальше остальных. Самой дальней гранью будет та, которая ничего не перекрывает. Т.е., если в строке с номером i все значения равны -1 или 0, то грань G_i находится позади остальных и её нужно отрисовать первой. После отрисовки этой грани, нужно исключить столбец и строку с номером i из дальнейшего рассмотрения. Повторять процедуру, пока не будут исключены все грани.

С использованием матрицы временная сложность алгоритма художника равна $O(n^2)$. Уменьшить алгоритмическую сложность практически невозможно, потому что даже для того, чтобы проверить, правильно ли отсортирован массив граней, нужно сравнить в массиве каждую грань с каждой.

Сортировка объектов по глубине

Самый простой алгоритм удаления невидимых поверхностей – сортировка граней объекта по глубине. Но этот алгоритм формирует изображение правильно только в том случае, если поверхность фигуры разбита на достаточное количество маленьких непересекающихся граней. Если это условие не выполнено, то в изображении возможны артефакты (нежелательные эффекты). Тогда пользователь увидит неправильный порядок сортировки граней. В этом случае просто необходимо каждую проблемную грань разбить на несколько граней.

Рассмотрим словесное описание данного алгоритма в совокупности с требуемыми аффинными преобразованиями:

1. При сортировке граней нужно использовать координаты, полученные после выполнения всех аффинных преобразований и проектирования:

```
  Matrix general_matrix =
            Matrix::RotationX(angle / 8) *      // Поворот куба вокруг оси x
            Matrix::RotationY(angle / 16) *     // Поворот куба вокруг оси y
            Matrix::Translation(0, 0, -4) *     // Перенос куба против оси z
            projection_matrix *                 // Проектирование
            Matrix::Viewport(0,  0,  frame.width,  frame.height);  //  Преобразование
нормализованных координат в оконные

        Vector B[8];

        for (int i = 0; i < _countof(A); i++)
        {
            B[i] = A[i] * general_matrix;
        }
```

2. В нашем случае массив B будет содержать нужные для сортировки координаты z (B[i].z). При сортировке можно четвёртую компоненту B[i].w игнорировать или выполнить нормализацию координаты z:

```
            B[i].z / B[i].w;
```

   Нормализация координат не влияет на результат сортировки. Обязательно учесть, что в нашей программе ось z инвертирована, т.е. направлена на наблюдателя (камеру).

3. Отсортировать грани фигуры по глубине в текущих координатах. Необходимо вычислить координату z центра (тяжести) каждой грани и отсортировать весь массив граней по этому значению. После сортировки массив будет содержать грани в порядке от ближних к дальним (или наоборот). В дальнейшем будет достаточно нарисовать на экране все грани в порядке от дальних к ближним. Таким образом, ближние к наблюдателю грани будут нарисованы последними. Лучше использовать стандартную функцию сортировки со сложностью $O(n \cdot log_2 n)$.

К примеру, дан массив граней куба, координаты которого уже готовы для растеризации:

```
Vector facets[][4] = { {B[0], B[1], B[2], B[3]},

// Вторая грань куба
{B[4], B[5], B[6], B[7]},

// Третья грань куба
{B[0], B[1], B[5], B[4]},

// Четвёртая грань куба
{B[2], B[3], B[7], B[6]},

// Пятая грань куба
{B[0], B[3], B[7], B[4]},

// Шестая грань куба
{B[1], B[2], B[6], B[5]} };
```

Ключом для сортировки является центр тяжести грани:

```
// Ключи для сортировки
float keys[6];

// Вычисление ключей. Ключ - координата z центра тяжести грани
for (int i = 0; i < 6; i++)
  keys[i] = (facets[i][0].z + facets[i][1].z + facets[i][2].z + facets[i][3].z) / 4;
```

Ключом для сортировки является центр тяжести нормализованных граней:

```
// Вычисление ключей. Ключ – нормализованная координата z центра тяжести грани
for (int i = 0; i < 6; i++)
keys[i] = (
  facets[i][0].z/facets[i][0].w +
  facets[i][1].z/facets[i][1].w +
  facets[i][2].z/facets[i][2].w +
  facets[i][3].z/facets[i][3].w) / 4;
```

4. Растеризовать отсортированные по координате *z* грани от дальних к ближним.