Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: «Вычислительная математика и программирование»
Дисциплина: «Компьютерная графика»

Лабораторная работа №2 по курсу «Компьютерная графика»

Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий.

Студент: Тимофеев А. В.
Преподаватель: Морозов А. В.
Группа: М80-307Б
Дата:
Оценка:
Подпись:

Москва, 2022

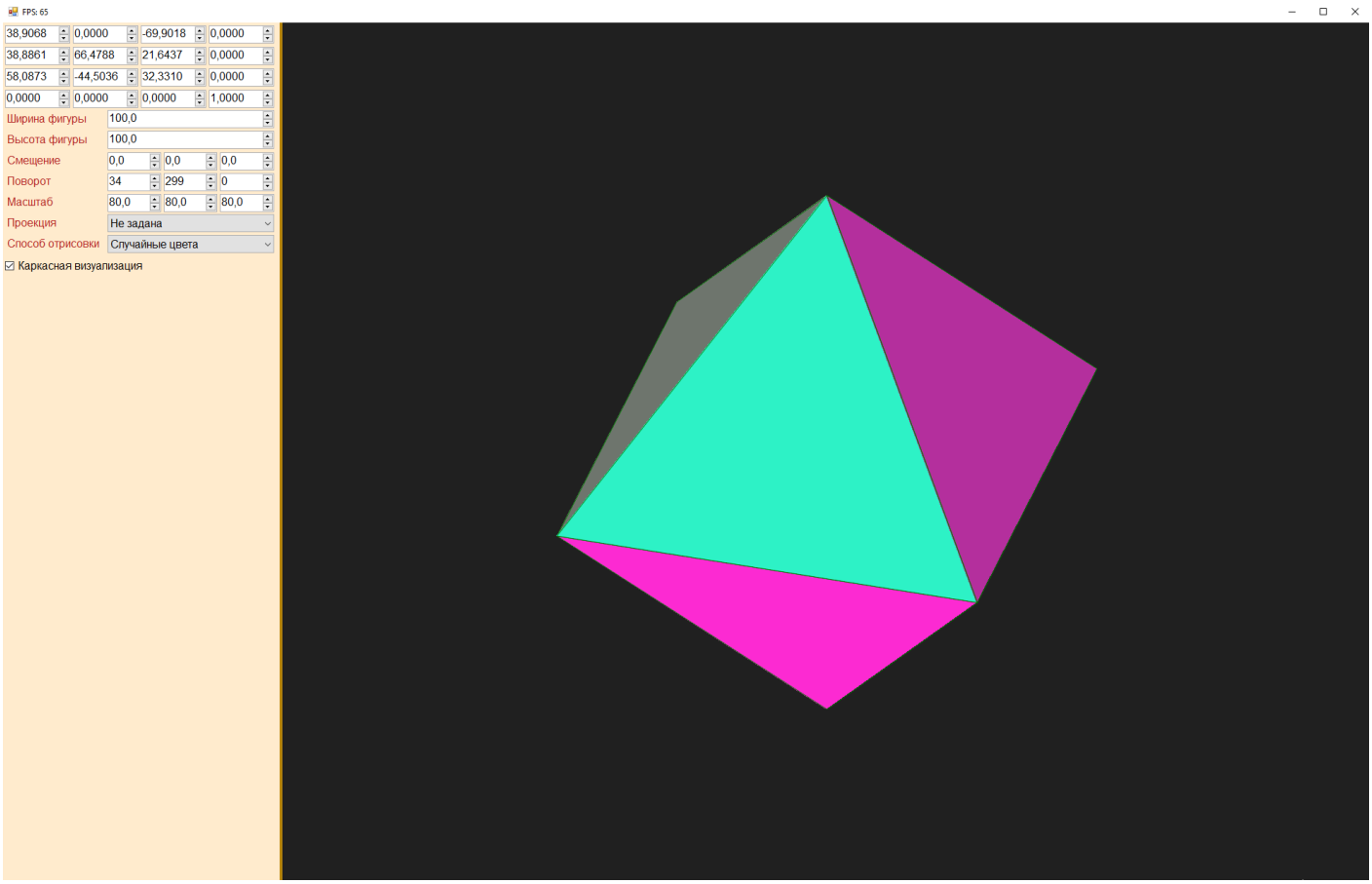
Постановка задачи

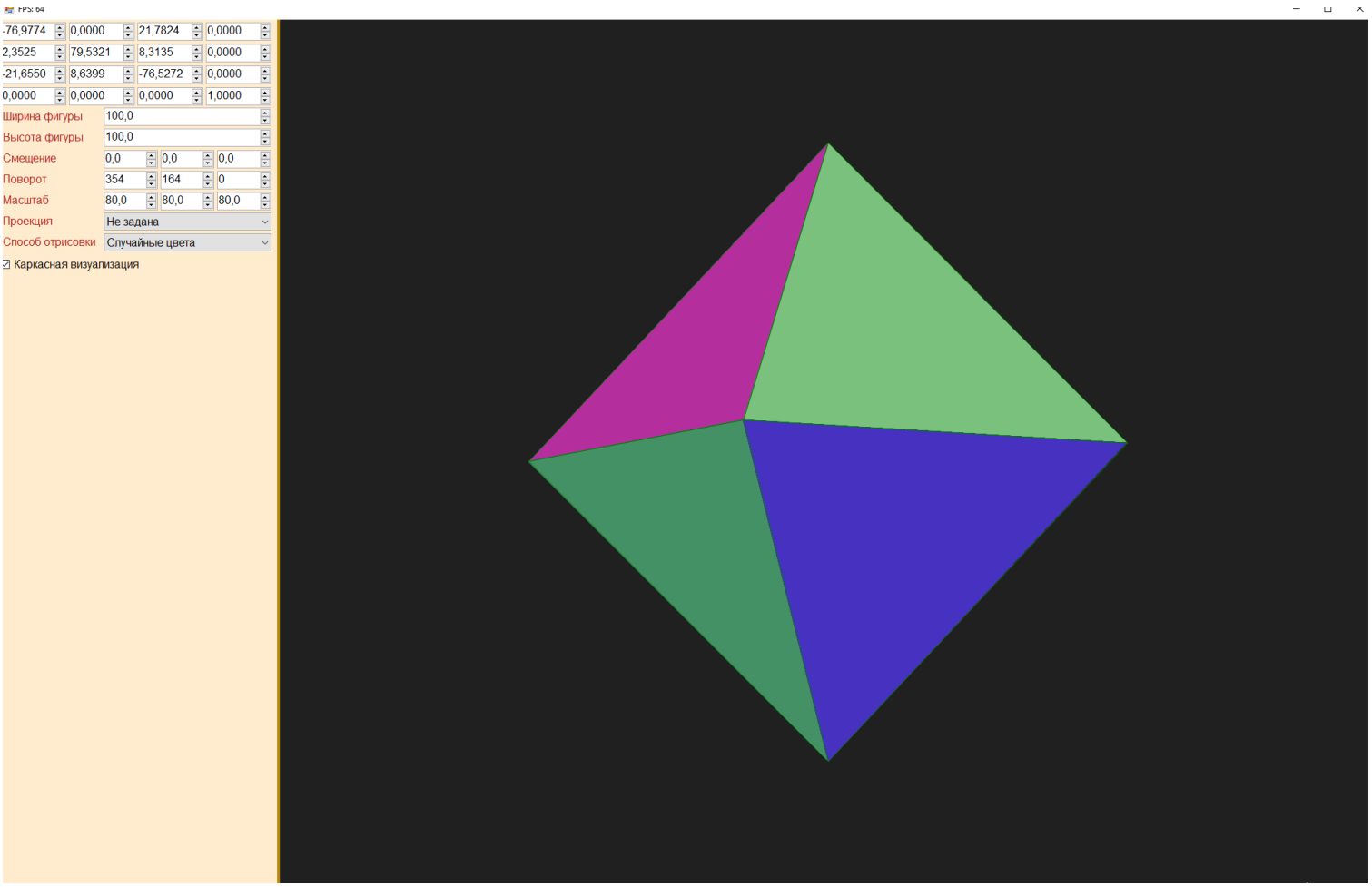
Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна.
Вариант: 2. Правильный октаэдр

Решение задачи

Формироваться заданная вариантом фигура будет следующим образом. Будем считать, что это две пирамиды в основании которых квадрат, они склеины основаниями. Вершина одной пирамиды находится в точке (0, 0, PyrHeight), вершина другой пирамиды находится в точке (0, 0, -PyrHeight). Для задания вершины пирамиды достаточно взять координату центра окружности и к значению z прибавить нужное значение высоты. Тут стоит обратить внимание на то, что для каждого полигона, составляющего фигуру, нужно посчитать нормаль к плоскости для определения при отрисовки видимых и невидимых граней соответственно. Для возможного проведения аффинных преобразований путем матричного перемножения необходимо считать все точки четырехмерными - появляется w координата, всегда равная 1 (для векторов - 0). После такого введения задание преобразующих матриц не составит труда, все их можно перемножить для получения итоговой матрицы одного сложного преобразования. Отдельную матрицу преобразований также требуется получить и для нормалей, чтобы каждый раз их не пересчитывать при любом изменении положения фигуры. После перевода исходных координат фигуры из видового пространства в мировое путем матричного перемножения также требуется перевести полученные координаты в физическую систему окна отрисовки - для этого сперва производим проецирование фигуры на плоскость путем деления x и y координаты каждой точки на w, а далее делаем все тоже самое, что было проделано в ЛР 1 - определяем коэффициент преобразования для каждой координаты в соответствии с размерами окна и области отрисовки.

Пример работы





Листинг программы

(основная часть)

```
protected override void OnMainWindowLoad(object sender, EventArgs args){
        // TODO: Инициализация данных
        RenderDevice.BufferBackCol = 0x20;
        ValueStorage.Font = new Font("Arial", 12f);
        ValueStorage.ForeColor = Color.Firebrick;
        ValueStorage.RowHeight = 30;
        ValueStorage.BackColor = Color.BlanchedAlmond;
        MainWindow.BackColor = Color.DarkGoldenrod;
        ValueStorage.RightColWidth = 50;
        VSPanelWidth = width;
        VSPanelLeft = true;
        MainWindow.Size = new Size(2500, 1380);
        MainWindow.StartPosition = FormStartPosition.Manual;
        MainWindow.Location = Point.Empty;
        RenderDevice.GraphicsHighSpeed = false;
        RenderDevice.BufferBackCol = 0x20;
        RenderDevice.MouseMoveWithRightBtnDown += (s, e)
            => Offset += new DVector3(0.35 * Math.Abs(_Scale.X) * e.MovDeltaX, 0.35 *
Math.Abs(_Scale.Y) * e.MovDeltaY,
                0);
```

```
        RenderDevice.MouseMoveWithLeftBtnDown += (s, e)
            => Rotation += new DVector3(0.1 * e.MovDeltaY, 0.1 * e.MovDeltaX, 0);
        RenderDevice.MouseWheel += (s, e) => Scale += new DVector3(0.05 * e.Delta, 0.05 *
e.Delta, 0.05 * e.Delta);
        Create();
    }
    protected override void OnDeviceUpdate(object s, DeviceArgs e){
        if (0 != ((int) _Commands & (int) Commands.FigureChange)){
            _Commands ^= Commands.FigureChange;
            Generate();
        }
        /* Обновление значений, использующихся для перевода в физ. с. к. */
        var x_min = vertices.Min(p => p.Point_InLocalSpace.X / p.Point_InLocalSpace.Z);
        var x_max = vertices.Max(p => p.Point_InLocalSpace.X / p.Point_InLocalSpace.Z);
        var y_min = vertices.Min(p => p.Point_InLocalSpace.Y / p.Point_InLocalSpace.Z);
        var y_max = vertices.Max(p => p.Point_InLocalSpace.Y / p.Point_InLocalSpace.Z);
        ViewSize.X = x_max - x_min;
        ViewSize.Y = y_max - y_min;
        AutoScale.X = .9 * e.Width / ViewSize.X;
        AutoScale.Y = .9 * e.Heigh / ViewSize.Y;
        AutoScale.X = AutoScale.Y = Math.Min(AutoScale.X, AutoScale.Y);
        Automove.X = e.Width / 2 - (x_min + x_max) / 2 * AutoScale.X;
        Automove.Y = e.Heigh / 2 - (y_min + y_max) / 2 * AutoScale.Y;
        if (0 != ((int) _Commands & (int) Commands.Transform)){
            _Commands ^= Commands.Transform;
            // Пересчет преобразования вектора нормали
            NormalTransform = DMatrix3.NormalVecTransf(PointTransform);
            foreach (var v in vertices) v.Point_InWorldSpace = PointTransform *
v.Point_InLocalSpace;
            foreach (var p in polygons){
                p.Normal_InWorldSpace = NormalTransform * p.Normal_InLocalSpace;
                p.IsVisible = p.Normal_InWorldSpace.Z < 0;
            }
            polygons.OrderBy(p => Math.Min(p.vertecies[0].Point_InWorldSpace.Z,
                Math.Min(p.vertecies[1].Point_InWorldSpace.Z,
p.vertecies[2].Point_InWorldSpace.Z)));
        }
        foreach (var p in polygons){
            if (!p.IsVisible)
                continue;
```

```
            if (CurVisual == Visualization.OneColor)
                e.Surface.DrawTriangle(Color.YellowGreen.ToArgb(),
FromViewToPhysicalSpace(p.vertecies[0]),
                    FromViewToPhysicalSpace(p.vertecies[1]),
                    FromViewToPhysicalSpace(p.vertecies[2]));
            else if (CurVisual == Visualization.RandomColor)
                e.Surface.DrawTriangle(p.Color, FromViewToPhysicalSpace(p.vertecies[0]),
                    FromViewToPhysicalSpace(p.vertecies[1]),
                    FromViewToPhysicalSpace(p.vertecies[2]));
            if (IsCarcass){
                e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.vertecies[0]),
                    FromViewToPhysicalSpace(p.vertecies[1]));
                e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.vertecies[1]),
                    FromViewToPhysicalSpace(p.vertecies[2]));
                e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.vertecies[2]),
                    FromViewToPhysicalSpace(p.vertecies[0]));
            }
        }
    }
}
```

Вывод

В результате выполнения данной лабораторной работы я познакомился с возможностью отрисовки 3D изображений простых фигур посредством вызова методов рисования отрезков в форме приложения. Также я научился применять сложные аффинные преобразования в трехмерном пространстве путем матричного произведения, проецировать фигуры в рабочую плоскость - выполнять изометрические и видовые проекции.