Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа

по курсу «Компьютерная графика».

«Каркасная визуализация поверхности»

Студент: И. С. Глушатов

Преподаватель: А. В. Морозов

Группа: М8О-307Б-19

Дата:

Оценка:

Подпись:

Москва, 2021

Содержание:

1. Постановка задачи

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и переключаются из графического интерфейса. Должна быть обеспечена возможность тестирования программы на различных наборах подготовленных исходных данных и их изменение.
Программа должна обеспечивать выполнение аффинных преобразований для заданной порции поверхности, а также возможность управлять количеством изображаемых параметрических линий. Для визуализации параметрических линий поверхности разрешается использовать только функции отрисовки отрезков в экранных координатах или буфер вершин OpenGL.
Реализовать возможность отображения опорных точек, направляющих и других данных по которым формируется порция поверхности и отключения каркасной визуализации.

Вариант: интерполяционная бикубическая поверхность Эрмита

2. Математическая модель

Интерполяционная бикубическая поверхность Эрмита – это поверхность, описываемая сплайнами Эрмита.

Сплайн Эрмита – это сплайн, построенный из кубических полиномов с использованием эрмитовой интерполяции, в соответствии с которой интерполируемая функция задается не только своими значениями в точках, но и её первыми производными.

Пусть у нас есть четыре крайних точки поверхности $P_{00}, P_{01}, P_{10}, P_{11}$ и набор из 12 векторов:

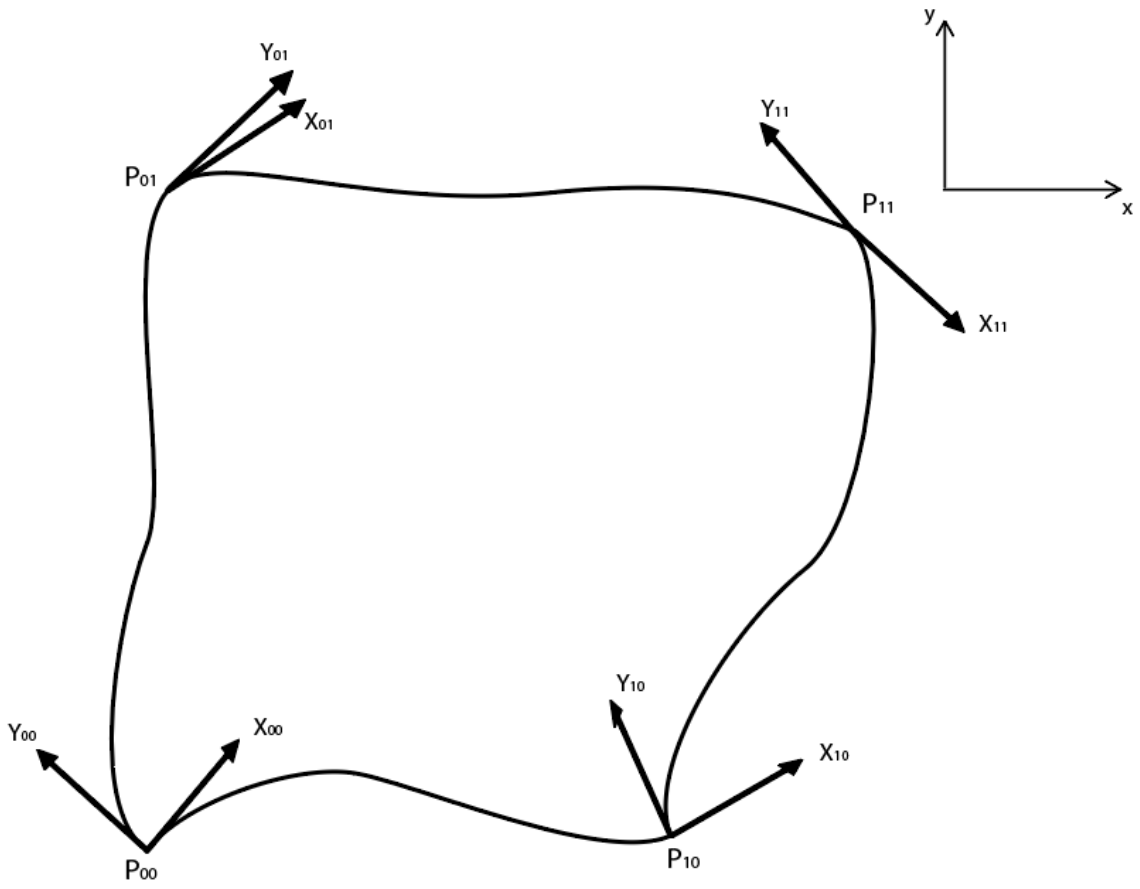
$$X_{00}, X_{01}, X_{10}, X_{11} \;\; Y_{00}, Y_{01}, Y_{10}, Y_{11} \;\; Z_{00}, Z_{01}, Z_{10}, Z_{11}$$

Тогда бикубическая поверхность Эрмита определяется матричным уравнением:

$$R(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix} = U^T M^T G M V, \quad 0 \le u, v \le 1, \text{где}$$

$$U = \begin{pmatrix} 1 \\ u \\ u^2 \\ u^3 \end{pmatrix}, V = \begin{pmatrix} 1 \\ v \\ v^2 \\ v^3 \end{pmatrix}, G = \begin{pmatrix} P_{00} & P_{01} & Y_{00} & Y_{01} \\ P_{10} & P_{11} & Y_{10} & Y_{11} \\ X_{00} & X_{01} & Z_{00} & Z_{01} \\ X_{10} & X_{11} & Z_{10} & Z_{11} \end{pmatrix}, M = \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

Вектора X и Y несут в себе геометрический смысл, как производные в точках (касательные), а вектора Z отвечают за “скручивание” в точках.



3. Листинг

Генерация поверхности:

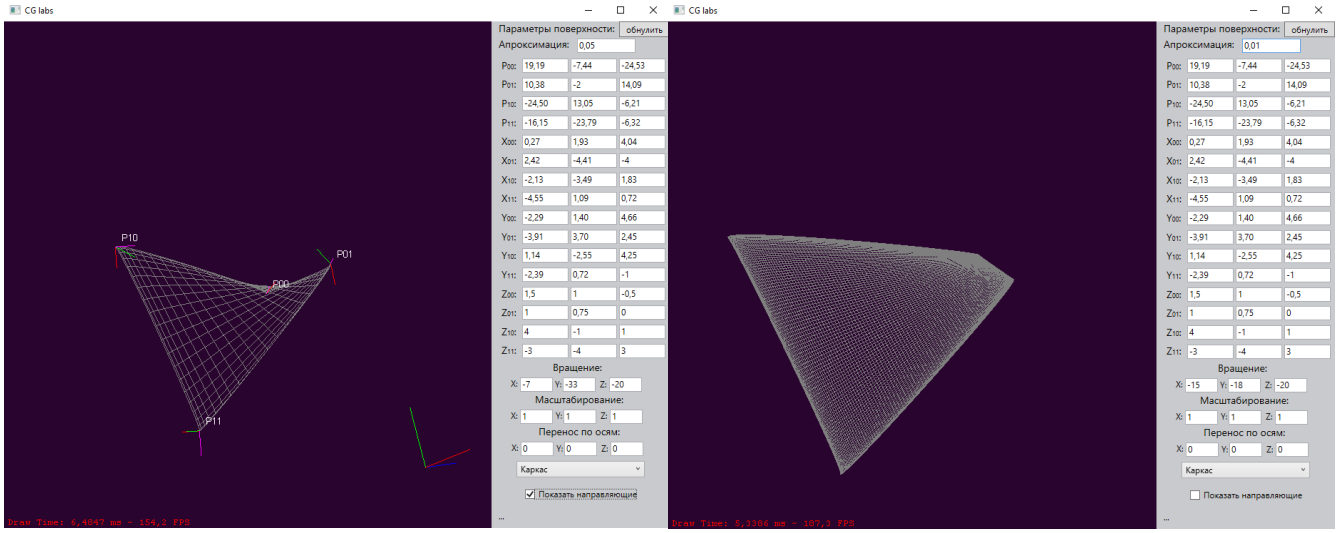
```
private Object4D GetSurface()
        {
            int nopboc = (int)(1 / Approximation) + 1;
            int number_of_points = nopboc * nopboc;

            Point4D[] points = new Point4D[number_of_points];
            uint[] points_indexes = new uint[(nopboc - 1) * (nopboc - 1) * 4];

            Matrix4D_p MGM = Matrix4D.Transpose(Hermit_Matrix) * surface_data * Hermit_Matrix;

            // Создание точек поверхности
            double u = 0.0d;
            for (int i = 0; i < nopboc; i++, u += Approximation)
            {
                double v = 0.0d;
                for (int j = 0; j < nopboc; j++, v += Approximation)
                {
                    Vector4D pu = new Vector4D(1, u, u * u, u * u * u);
                    Vector4D pv = new Vector4D(1, v, v * v, v * v * v);

                    points[i * nopboc + j] = pu * MGM * pv;
                }
            }

            // создание индексов полигонов
            uint k = 0;
            for (uint i = 0; i < nopboc - 1; i++)
            {
                for (uint j = 0; j < nopboc - 1; j++)
                {
                    points_indexes[k] = i * (uint)nopboc + j;
                    points_indexes[k + 1] = i * (uint)nopboc + j + 1;
                    points_indexes[k + 2] = i * (uint)nopboc + j + (uint)nopboc + 1;
                    points_indexes[k + 3] = i * (uint)nopboc + j + (uint)nopboc;
                    k += 4;
                }
            }

            return new Object4D(points, points_indexes);
        }
```

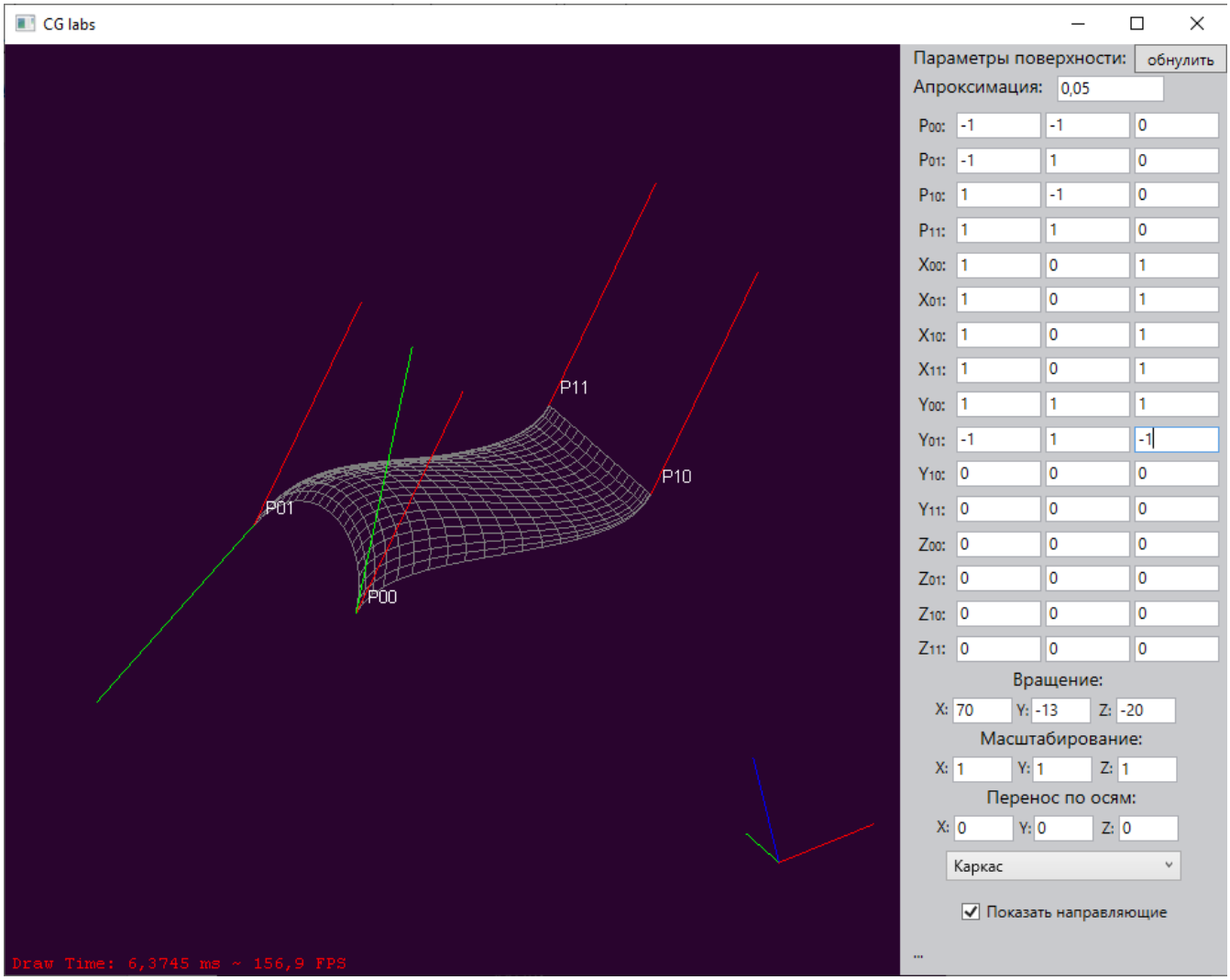
Отрисовка:

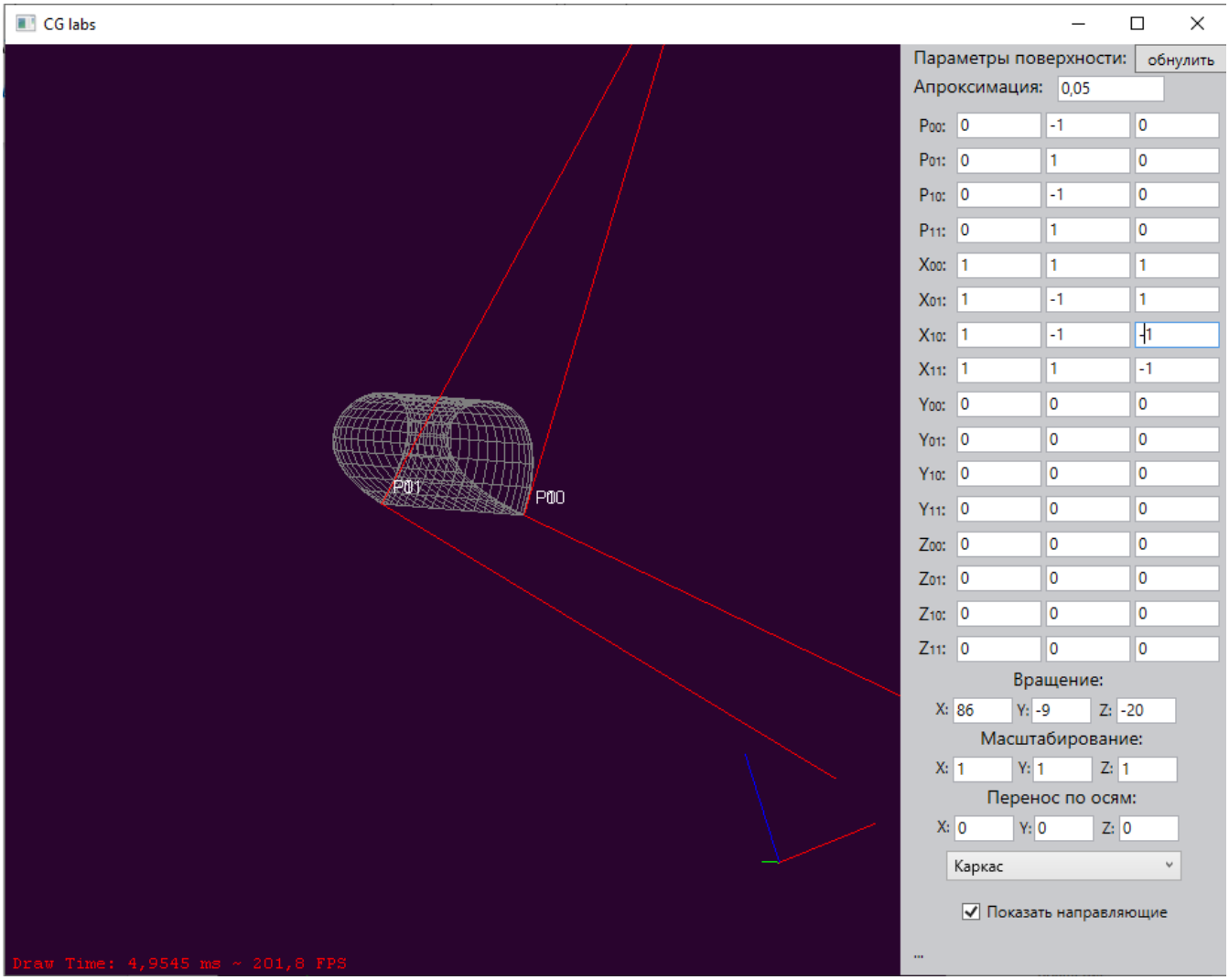
```
private void OpenGLControl_OpenGLDraw(object sender, OpenGLEventArgs args)
        {
            OpenGL gl = args.OpenGL;
            gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT | OpenGL.GL_STENCIL_BUFFER_BIT);

            if (ChangeSettings)
            {
                if (((int)FSettingChanges & (int)ESettingChanges.MatrixModelView) != 0)
                {
                    gl.MatrixMode(MatrixMode.Modelview);
                    gl.LoadMatrix(NewAllTransformationsMatrix().ToArray());
                }
                if (((int)FSettingChanges & (int)ESettingChanges.MatrixProjection) != 0)
                {
                    double koefHeight = ActualHeight / BasicHeight;
                    double koefWidth = (ActualWidth - 200) / BasicWidth;

                    gl.MatrixMode(MatrixMode.Projection);
                    gl.LoadMatrix(new double[] {
                        1/koefWidth, 0, 0, 0,
                        0, 1/koefHeight, 0, 0,
                        0, 0, 0, 0,
                        0, 0, 0, 1,
                    });
                }
                if (((int)FSettingChanges & (int)ESettingChanges.PolygonMode) != 0)
                {
                    if (FDrawMode == EDrawMode.Polyline)
                    {
                        if (gl.IsEnabled(OpenGL.GL_CULL_FACE))
                        {
                            gl.Disable(OpenGL.GL_CULL_FACE);
                        }

                        gl.PolygonMode(FaceMode.FrontAndBack, PolygonMode.Lines);
                    }
                    else if (FDrawMode == EDrawMode.PolylineVisible)
                    {
                        gl.Enable(OpenGL.GL_CULL_FACE);
```

```
                    gl.CullFace(OpenGL.GL_FRONT);
                    gl.PolygonMode(FaceMode.FrontAndBack, PolygonMode.Lines);
                }
                else if (FDrawMode == EDrawMode.Polygon)
                {
                    if (gl.IsEnabled(OpenGL.GL_CULL_FACE))
                        gl.Disable(OpenGL.GL_CULL_FACE);
                    gl.PolygonMode(FaceMode.FrontAndBack, PolygonMode.Filled);
                }
            }

            FSettingChanges = 0;
            ChangeSettings = false;
        }

        if (ChangeObject) { Surface = GetSurface(); ChangeObject = false; }

        unsafe
        {
            gl.EnableClientState(OpenGL.GL_VERTEX_ARRAY);
            gl.EnableClientState(OpenGL.GL_COLOR_ARRAY);

            fixed (Point4D* p = Surface.Points)
            {
                gl.VertexPointer(3, OpenGL.GL_DOUBLE, sizeof(Point4D), (IntPtr)(&p->X));
                gl.ColorPointer(3, OpenGL.GL_DOUBLE, sizeof(Point4D), (IntPtr)(&p->R));
            }

            fixed (uint* p1 = Surface.PolygonsIndexes)
            {
                gl.DrawElements(OpenGL.GL_QUADS, Surface.PolygonsIndexes.Length, OpenGL.GL_UNSIGNED_INT, (IntPtr)p1);
            }

            gl.DisableClientState(OpenGL.GL_VERTEX_ARRAY);
            gl.DisableClientState(OpenGL.GL_COLOR_ARRAY);
        }

        if (BoolDirects)
        {
            #region Отрисовка направляющих

            double koefHeight = ActualHeight / BasicHeight;
            double koefWidth = (ActualWidth - 200) / BasicWidth;

            Matrix4D project = new Matrix4D(new double[,] {
                    { 1/koefWidth, 0, 0, 0, },
                    { 0, 1/koefHeight, 0, 0, },
                    { 0, 0, 0, 0, },
                    { 0, 0, 0, 1, },
             });

            for (int i = 0; i < 4; i++)
            {
                int x = i / 2, y = i % 2;
                Point4D p = surface_data[x, y] * NewAllTransformationsMatrix() * project;
                int px = (int)(p.X * OpenGLGrid.ActualWidth / 2 + OpenGLGrid.ActualWidth / 2), py = (int)(p.Y *
OpenGLGrid.ActualHeight / 2 + OpenGLGrid.ActualHeight / 2);
                gl.DrawText(px + 9, py + 9, 1f, 1f, 1f, "Arial", 12, $"P{x}{y}");
                gl.Begin(OpenGL.GL_LINES);
                gl.Color(1d, 0d, 0d);
                gl.Vertex(surface_data[x, y].ToDoubleArray());
                gl.Vertex((surface_data[x, y] + surface_data[x + 2, y]).ToDoubleArray());
                gl.Color(0d, 1d, 0d);
                gl.Vertex(surface_data[x, y].ToDoubleArray());
                gl.Vertex((surface_data[x, y] + surface_data[x, y + 2]).ToDoubleArray());
                gl.Color(1d, 0d, 1d);
                gl.Vertex(surface_data[x, y].ToDoubleArray());
                gl.Vertex((surface_data[x, y] + surface_data[x + 2, y + 2]).ToDoubleArray());
                gl.End();
            }

            #endregion
            #region Отрисоыка осей

            gl.MatrixMode(MatrixMode.Modelview);

            Matrix4D mm = Matrix4D.ScaleMatrix(0.25, 0.25, 0.25) *
                Matrix4D.RotateMatrix(RotateXAngle, RotateYAngle, RotateZAngle) *
                Matrix4D.TranslationMatrix(0.8 * OpenGLGrid.ActualWidth / BasicWidth, -0.8 * OpenGLGrid.ActualHeight /
BasicHeight, 0);
            gl.LoadMatrix(mm.ToArray());

            gl.Begin(OpenGL.GL_LINES);
                gl.Color(0f, 0f, 1f); gl.Vertex(0, 0, 0); gl.Vertex(0, 0, 1);
                gl.Color(0f, 1f, 0f); gl.Vertex(0, 0, 0); gl.Vertex(0, 1, 0);
                gl.Color(1f, 0f, 0f); gl.Vertex(0, 0, 0); gl.Vertex(1, 0, 0);
            gl.End();

            gl.MatrixMode(MatrixMode.Modelview);
            gl.LoadMatrix(NewAllTransformationsMatrix().ToArray());
            #endregion
        }
    }
```

4. Демонстрация






CG labs
Параметры поверхности:
обнулить
Апроксимация:
0,05
P00: 0 -1 0
P01: 0 1 0
P10: 0 -1 0
P11: 0 1 0
X00: 1 1 1
X01: 1 -1 1
X10: 1 -1 -1
X11: 1 1 -1
Y00: 0 0 0
Y01: 0 0 0
Y10: 0 0 0
Y11: 0 0 0
Z00: 0 0 0
Z01: 0 0 0
Z10: 0 0 0
Z11: 0 0 0
Вращение:
X: 86 Y: -9 Z: -20
Масштабирование:
X: 1 Y: 1 Z: 1
Перенос по осям:
X: 0 Y: 0 Z: 0
Каркас
Показать направляющие
P01
P00
Draw Time: 4,9545 ms ~ 201,8 FPS

5. Выводы

В ходе курсовой работы я реализовал построение и отображение интерполяционной бикубической поверхности Эрмита. Небольшим недостатком мне показалось трудность в расчетах точек при интерполяции в силу громоздких матричных умножений, а также первичное понимание процесса, из-за чего конечный результат кажется скорее “магией”, нежели чем четкой математической моделью. Однако мне очень понравился этот способ построения поверхности из-за использования производных (касательных), так как правильность построения легко прослеживается при отображении, что, как по мне делает этот метод выразительнее относительно поверхностей Безье.

Ссылка на GitHub репозиторий с проектом: https://github.com/Igor743646/CompGrap/tree/master/KP

6. Список литературы

1. WPF документация Microsoft [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/dotnet/desktop/wpf (дата обращения 25.12.2021)
2. Про бикубическую поверхность Эрмита [Электронный ресурс]. URL: https://scask.ru/g_book_cpsp.php?id=90 (дата обращения 25.12.2021)
3. Видео о сплайнах Эрмита в 2D пространстве [Электронный ресурс]. URL: https://www.youtube.com/watch?v=CwvNlEWVLqg (дата обращения 25.12.2021)