Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Компьютерная графика»
Тема: Построение изображений 2D-кривых.

Студент: Д. С. Ляшун
Преподаватель: А. В. Морозов
Группа: М8О-307Б

Дата:
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Цель: Приобретение практический знаний для отрисовки 2D изображений кривых в приложениях.

Задание: Написать и отладить программу, строящую изображение заданной замечательной кривой. Обеспечить автоматическое масштабирование и центрирование кривой при изменении размеров окна.

Вариант: $\rho = a * (1 - cos\phi)$, где a - константа, вводимая пользователем, ϕ, ρ - полярные координаты.

2 Теоретические сведения

Для перевода исходного уравнения кривой в параметрическую запись воспользуемся следующей формулой:

$$\begin{cases} x = a * (1 - cos\phi) * cos\phi \\ y = a * (1 - cos\phi) * sin\phi \end{cases}$$

После формирования точек, задающих нашу кривую, повернем её на угол θ по такой формуле:

$$\begin{cases} x = x * cos\theta - y * sin\theta \\ y = x * sin\theta + y * cos\theta \end{cases}$$

Далее выполняется операция преобразование точек в видовой системе координат к физической системе координат экрана. Для этого необходимо определить минимальные и максимальные координаты по осям x и y для всех точек - для задания размера области отрисовки экрана, далее посчитать ширину/высоту экрана и по этим величинам задать коэффициенты, использующиеся для преобразования координат.

3 Листинг программы

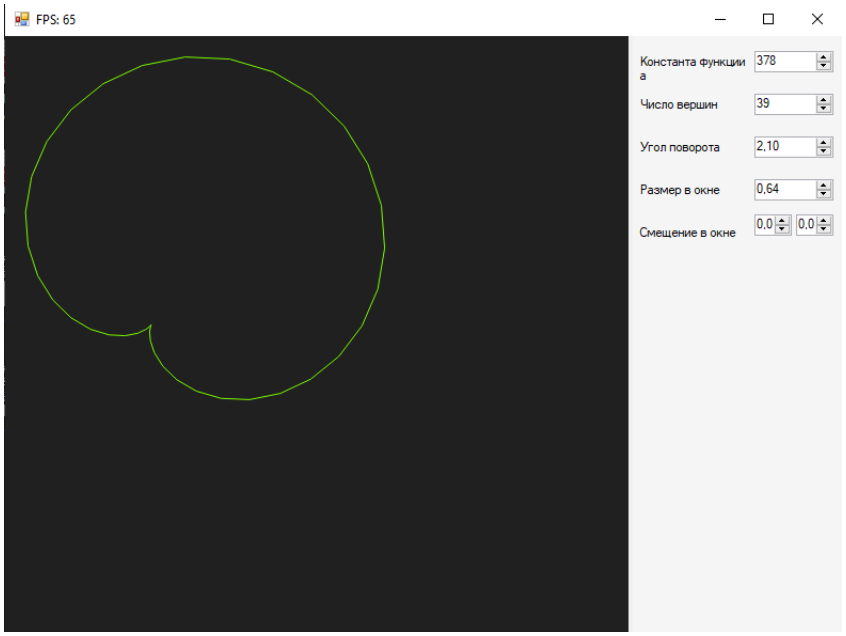
Исходный код Program.cs:

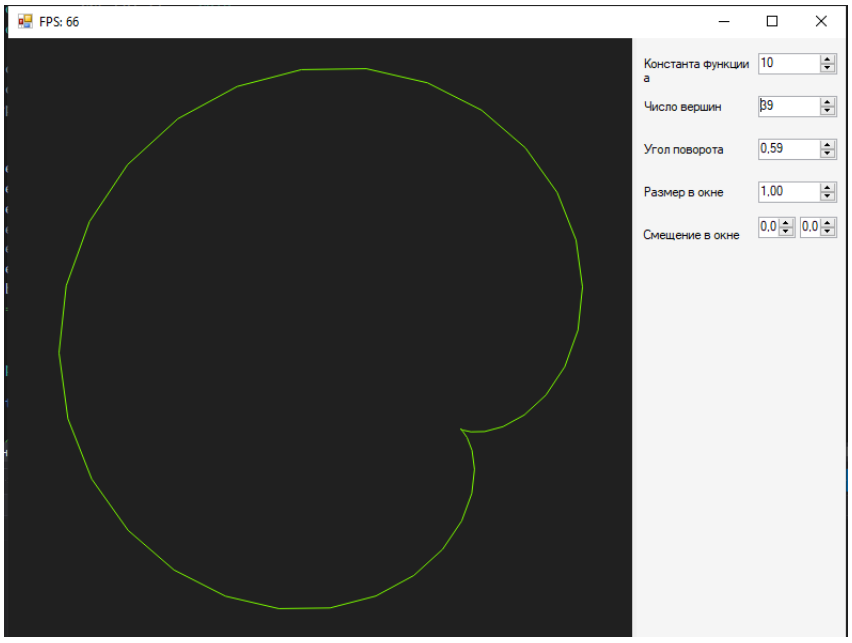
```
1  //#define UseOpenGL //    OpenGL
2  #if (!UseOpenGL)
3  using Device = CGLabPlatform.GDIDevice;
4  using DeviceArgs = CGLabPlatform.GDIDeviceUpdateArgs;
5  #else
6  using Device = CGLabPlatform.OGLDevice;
7  using DeviceArgs = CGLabPlatform.OGLDeviceUpdateArgs;
8  using SharpGL;
9  #endif
10
11 using System;
12 using System.Linq;
13 using System.Drawing;
14 using System.Windows.Forms;
15 using System.Drawing.Imaging;
16 using System.Collections.Generic;
17 using CGLabPlatform;
18 // ================================================================================
19
20
21 using CGApplication = MyApp;
22 public abstract class MyApp: CGApplicationTemplate<CGApplication, Device, DeviceArgs>
23 {
24     // TODO:  ,
25     #region
26
27     [DisplayNumericProperty(Default: 10, Increment: 1, Minimum: 0, Maximum: 1000, Name:
           "  ")]
28     public abstract double A { get; set; }
29
30     [DisplayNumericProperty(Default: 4, Increment: 1, Minimum: 1, Maximum: 10000, Name:
           " ")]
31     public abstract int VertexCount { get; set; }
32
33     [DisplayNumericProperty(Default: 0, Increment: 0.01, Minimum: 0, Maximum: 2 * Math.
         PI, Name: " ")]
34     public abstract double Angle { get; set; }
35
36     [DisplayNumericProperty(Default: 1, Increment: 0.01, Minimum: 0.01, Maximum: 2,
         Name: "  ")]
37     public abstract double WindowScale { get; set; }
38
39     [DisplayNumericProperty(Default: new[] { 0d, 0d }, Increment: 0.1, Name: "C  ")]
40     public abstract DVector2 WindowMove{ get; set; }
41
42
```

```
43      public DVector2 ViewSize;
44      public DVector2 Automove;
45      public DVector2 AutoScale;
46      #endregion
47
48      #region
49      protected DVector2 CoordinateTransformation(DVector2 point)
50      {
51          DVector2 result = new DVector2();
52          var X = point.X;
53          var Y = point.Y;
54          result.X = X * Math.Cos(Angle) - Y * Math.Sin(Angle); //
55          result.Y = X * Math.Sin(Angle) + Y * Math.Cos(Angle);
56          return result;
57      }
58      protected DVector2 FromViewToPhysicalSpace(DVector2 point)
59      {
60          point.X = point.X * AutoScale.X + Automove.X; //
61          point.Y = point.Y * AutoScale.Y + Automove.Y;
62          point.X *= WindowScale; //
63          point.Y *= WindowScale;
64          point.X += WindowMove.X; //
65          point.Y += WindowMove.Y;
66          return point;
67      }
68      #endregion
69
70
71      protected override void OnMainWindowLoad(object sender, EventArgs args)
72      {
73          // TODO:
74          base.RenderDevice.BufferBackCol = 0x20;
75      }
76
77
78      protected override void OnDeviceUpdate(object s, DeviceArgs e)
79      {
80          // TODO:
81          double step = 2 * Math.PI / VertexCount;
82          double angle = 0;
83          double X, Y;
84          List<DVector2> points = new List<DVector2>();
85          while (angle < 2 * Math.PI)
86          {
87              X = A * (1 - Math.Cos(angle)) * Math.Cos(angle);
88              Y = A * (1 - Math.Cos(angle)) * Math.Sin(angle);
89              points.Add(new DVector2(X, Y));
90              angle += step;
91          }
```

```
92          X = A * (1 - Math.Cos(0)) * Math.Cos(0);
93          Y = A * (1 - Math.Cos(0)) * Math.Sin(0);
94          points.Add(new DVector2(X, Y));
95
96          for (int i = 0; i < points.Count; ++i)
97          {
98              points[i] = CoordinateTransformation(points[i]);
99          }
100         var x_min = points.Min(p => p.X);
101         var x_max = points.Max(p => p.X);
102         var y_min = points.Min(p => p.Y);
103         var y_max = points.Max(p => p.Y);
104         ViewSize.X = points.Max(p => p.X) - points.Min(p => p.X);
105         ViewSize.Y = points.Max(p => p.Y) - points.Min(p => p.Y);
106         AutoScale.X = .9 * e.Width / ViewSize.X;
107         AutoScale.Y = .9 * e.Heigh / ViewSize.Y;
108         AutoScale.X = AutoScale.Y = Math.Min(AutoScale.X, AutoScale.Y);
109         Automove.X = e.Width / 2 - (x_min + x_max) / 2 * AutoScale.X;
110         Automove.Y = e.Heigh / 2 - (y_min + y_max) / 2 * AutoScale.Y;
111
112         for (int i = 1; i < points.Count; ++i)
113         {
114             e.Surface.DrawLine(Color.LawnGreen.ToArgb(), FromViewToPhysicalSpace(points
                    [i]), FromViewToPhysicalSpace(points[i - 1]));
115         }
116     }
117
118 }
119 // ===========================================================================
120 public abstract class AppMain : CGApplication
121 { [STAThread] static void Main() { RunApplication(); } }
```

4 Демонстрация работы





5 Выводы

В результате выполнения данной лабораторной работы я познакомился с возможностью отрисовки 2D изображений кривых посредством вызова методов рисования отрезков в форме приложения. Также я научился применять такие простые аффинные преобразования как сдвиг, поворот и масштабирование.

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Компьютерная графика»

Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий.

Студент: Д. С. Ляшун
Преподаватель: А. В. Морозов
Группа: М8О-307Б
Дата:
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Цель: Приобретение практический знаний для отрисовки простых 3D фигур и их проецирования на плоскость.

Задание: 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: 8–гранная прямая правильная пирамида.

2 Теоретические сведения

Формироваться заданная вариантом фигура будет следующим образом. Будем считать, что в основании её лежит окружность, аппроксимированная до 8 вершин, что дает в результате правильный восьмиугольник. Для задания же 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.

3 Листинг программы

Формирование матрицы преобразований:

```
1  private void UpdateTransformMatrix()
2  {
3      _PointTransform = new DMatrix4(new double[] {1, 0, 0, 0,
4                                                   0, 1, 0, 0,
5                                                   0, 0, 1, 0,
6                                                   0, 0, 0, 1 });
7      DMatrix4 x_rotate = new DMatrix4(new double[] {1, 0, 0, 0,
8                                                     0, Math.Cos(Math.PI / 180 * _Rotation.X)
                                                         , -Math.Sin(Math.PI / 180 * _Rotation
                                                         .X), 0,
9                                                     0, Math.Sin(Math.PI / 180 * _Rotation.X)
                                                         , Math.Cos(Math.PI / 180 * _Rotation.
                                                         X), 0,
10                                                    0, 0, 0, 1 });
11     DMatrix4 y_rotate = new DMatrix4(new double[] {Math.Cos(Math.PI / 180 * _Rotation.Y
           ), 0, Math.Sin(Math.PI / 180 * _Rotation.Y), 0,
12                                                    0, 1, 0, 0,
13                                                    -Math.Sin(Math.PI / 180 * _Rotation.Y),
                                                          0, Math.Cos(Math.PI / 180 * _Rotation
                                                         .Y), 0,
14                                                    0, 0, 0, 1 });
15     DMatrix4 z_rotate = new DMatrix4(new double[] {Math.Cos(Math.PI / 180 * _Rotation.Z
           ), -Math.Sin(Math.PI / 180 * _Rotation.Z), 0, 0,
16                                                    Math.Sin(Math.PI / 180 * _Rotation.Z),
                                                          Math.Cos(Math.PI / 180 * _Rotation.Z
                                                         ), 0, 0,
17                                                    0, 0, 1, 0,
18                                                    0, 0, 0, 1 });
19     _PointTransform *= x_rotate;
20     _PointTransform *= y_rotate;
21     _PointTransform *= z_rotate;
22
23     _PointTransform *= new DMatrix4(new double[] {_Scale.X, 0, 0, 0,
24                                                   0, _Scale.Y, 0, 0,
25                                                   0, 0, _Scale.Z, 0 ,
26                                                   0, 0, 0, 1 });
27
28
29
30     DMatrix4 offset = new DMatrix4(new double[] { 1, 0, 0, _Offset.X,
31                                                  0, 1, 0, _Offset.Y,
32                                                  0, 0, 1, _Offset.Z,
33                                                  0, 0, 0, 1 });
34     _PointTransform *= offset;
35
36     _Commands |= Commands.Transform;
```

```
37
38 }
```

Создание и генерация фигуры:

```
1  public void Create()
2  {
3      vertices = new Vertex[9];
4      polygons = new Polygon[14];
5      var random = new Random();
6      for (int i = 0; i < vertices.Length; ++i)
7      {
8          vertices[i] = new Vertex();
9      }
10     for (int i = 0; i < polygons.Length; ++i)
11     {
12         polygons[i] = new Polygon();
13         polygons[i].Color = random.Next();
14     }
15     Generate();
16 }
17 public void Generate()
18 {
19     for (int i = 0; i < 8; ++i)
20     {
21         vertices[i].Point_InLocalSpace.X = BaseRadius * Math.Cos(2 * i * Math.PI / 8);
22         vertices[i].Point_InLocalSpace.Y = BaseRadius * Math.Sin(2 * i * Math.PI / 8);
23         vertices[i].Point_InLocalSpace.Z = 1;
24         vertices[i].Point_InLocalSpace.W = 1;
25     }
26     vertices[8].Point_InLocalSpace.X = 0;
27     vertices[8].Point_InLocalSpace.Y = 0;
28     vertices[8].Point_InLocalSpace.Z = 1 + PyrHeight;
29     vertices[8].Point_InLocalSpace.W = 1;
30     for (int i = 2; i <= 8; i += 2)
31     {
32         polygons[i / 2 - 1].vertecies = new []{ vertices[i % 8], vertices[i - 1],
               vertices[i - 2] };
33     }
34     polygons[4].vertecies = new[]{ vertices[4], vertices[2], vertices[0] };
35     polygons[5].vertecies = new[]{ vertices[6], vertices[4], vertices[0] };
36
37     for (int i = 1; i <= 8; ++i)
38     {
39         polygons[5 + i].vertecies = new []{ vertices[8], vertices[i - 1], vertices[i %
               8] };
40     }
41     foreach (var p in polygons)
42     {
43         DVector4 first = new DVector4(new double[] {p.vertecies[1].Point_InLocalSpace.X
```

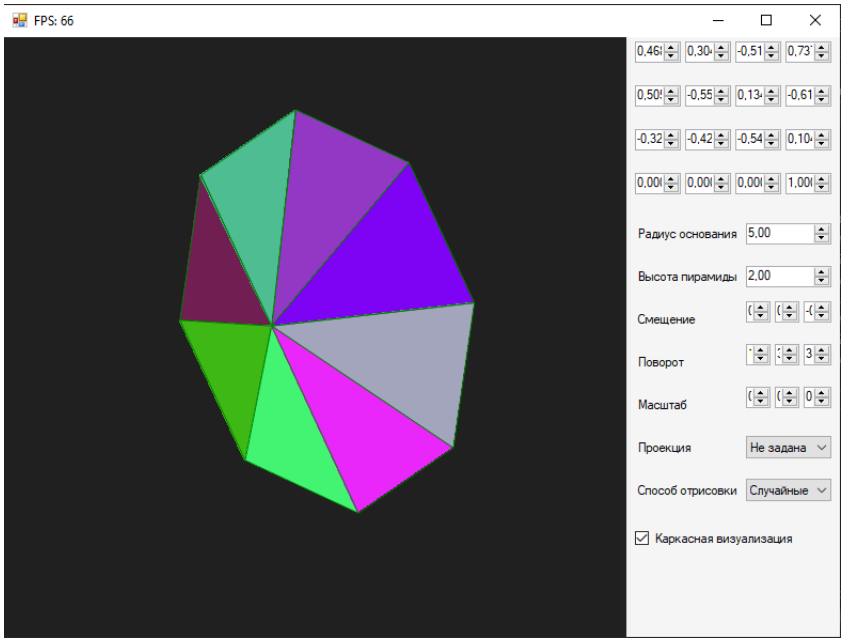
```
            - p.vertecies[0].Point_InLocalSpace.X,
                                                p.vertecies[1].Point_InLocalSpace.Y - p
                                                    .vertecies[0].Point_InLocalSpace.Y,
                                                p.vertecies[1].Point_InLocalSpace.Z - p
                                                    .vertecies[0].Point_InLocalSpace.Z,
                                                p.vertecies[1].Point_InLocalSpace.W - p
                                                    .vertecies[0].Point_InLocalSpace.W})
                                                    ;
        DVector4 second = new DVector4(new double[] {p.vertecies[2].Point_InLocalSpace.
            X - p.vertecies[0].Point_InLocalSpace.X,
                                                p.vertecies[2].Point_InLocalSpace.Y - p
                                                    .vertecies[0].Point_InLocalSpace.Y,
                                                p.vertecies[2].Point_InLocalSpace.Z - p
                                                    .vertecies[0].Point_InLocalSpace.Z,
                                                p.vertecies[2].Point_InLocalSpace.W - p
                                                    .vertecies[0].Point_InLocalSpace.W})
                                                    ;
        p.Normal_InLocalSpace = first * second;
        p.Normal_InLocalSpace.Normalize();
    }

}
```

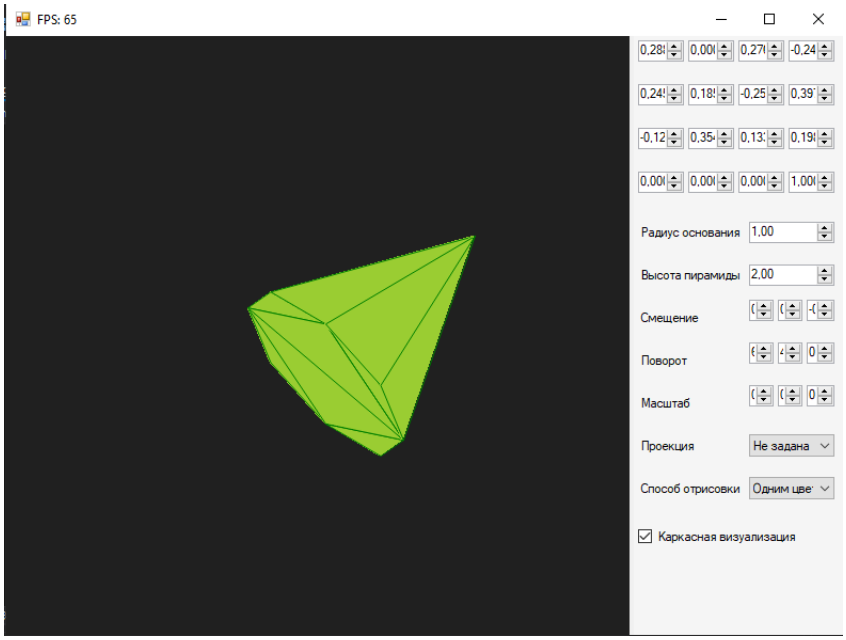
Основной цикл работы:

```
protected override void OnDeviceUpdate(object s, DeviceArgs e)
{
    if (0 != ((int)_Commands & (int)Commands.FigureChange))
    {
        _Commands ^= Commands.FigureChange;
        Generate();
    }

    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;
        NormalTransform = DMatrix3.NormalVecTransf(PointTransform);

```

```
26          foreach (var v in vertices)
27          {
28              v.Point_InWorldSpace = PointTransform * v.Point_InLocalSpace;
29          }
30
31          foreach (var p in polygons)
32          {
33              p.Normal_InWorldSpace = NormalTransform * p.Normal_InLocalSpace;
34              p.IsVisible = p.Normal_InWorldSpace.Z < 0;
35          }
36          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)));
37      }
38
39      foreach (var p in polygons)
40      {
41          if (!p.IsVisible)
42              continue;
43
44          if (CurVisual == Visualization.OneColor)
45          {
46              e.Surface.DrawTriangle(Color.YellowGreen.ToArgb(), FromViewToPhysicalSpace(
                    p.vertecies[0]), FromViewToPhysicalSpace(p.vertecies[1]),
47                  FromViewToPhysicalSpace(p.vertecies[2]));
48          }
49          else if (CurVisual == Visualization.RandomColor)
50          {
51              e.Surface.DrawTriangle(p.Color, FromViewToPhysicalSpace(p.vertecies[0]),
                    FromViewToPhysicalSpace(p.vertecies[1]),
52                  FromViewToPhysicalSpace(p.vertecies[2]));
53          }
54
55          if (IsCarcass)
56          {
57              e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.
                    vertecies[0]), FromViewToPhysicalSpace(p.vertecies[1]));
58              e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.
                    vertecies[1]), FromViewToPhysicalSpace(p.vertecies[2]));
59              e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.
                    vertecies[2]), FromViewToPhysicalSpace(p.vertecies[0]));
60          }
61      }
62  }
```

4 Демонстрация работы





5 Выводы

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.

Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Компьютерная графика»
Тема: Основы построения фотореалистичных изображений.

Студент: Д. С. Ляшун
Преподаватель: А. В. Морозов
Группа: М8О-307Б

Дата:
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Цель: Приобретение практический знаний для отрисовки фотореалистичных 3D фигур, освещаемых различными источниками света.

Задание: Используя результаты Л.Р.№2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

Вариант тела: Прямой эллиптический цилиндр.

2 Теоретические сведения

В лабораторной работе требуется реализовать плоскую модель затенения и модель затенения по Гуро. Различия между ними заключаются в том, что первая раскрашивает каждый полигон в один соответствующий цвет, в то время как модель Гуро интерполирует значения цвета, посчитанного для каждой вершины, составляющей полигон.

Значения интенсивности для вершин/полигонов будут определяться по следующей формуле, которая объединяет интенсивности диффузной, рассеянной и зеркальной составляющих:

$$I = i_a k_a + i_l \frac{k_d(\vec{L} \cdot \vec{N}) + k_s cos^p(\vec{R}, \vec{S})}{d + K}$$

Для задания координат вершин прямого эллиптического цилиндра будем считать его центр масс также и центром видовой системы координат. Далее все очень просто - имеем два уравнения для эллиптических оснований вида:

$$\begin{cases} x = r_1 cos\phi \\ y = r_2 sin\phi \\ z = h/2 \end{cases}$$

где r_1, r_2 - радиусы полуосей, h - высота цилиндра, при задании нижнего основания считаем $z = -h/2$.

Для формирования боковых граней также будем использовать уравнения эллипса, только периодически смещаясь по z координате, задавая тем самым последовательные "этажи"из граней:

$$\begin{cases} x = r_1 cos\phi \\ y = r_2 sin\phi \\ z = h/2 - \frac{h}{cnt-1} i, i = 1...cnt \end{cases}$$

где cnt - число этажей, это значение аппроксимации будет задаваться пользователем в программе.

3 Листинг программы

Формирование фигуры:

```
public void Create()
{
    approx0 = (int)Approximation[0];
    approx1 = (int)Approximation[1];
    vertices = new Vertex[approx0 * approx1 + 2];
    polygons = new Polygon[approx0 * 2 + (approx1 - 1) * approx0 * 2];
    var random = new Random();
    for (int i = 0; i < vertices.Length; ++i)
    {
        vertices[i] = new Vertex();
    }
    for (int i = 0; i < polygons.Length; ++i)
    {
        polygons[i] = new Polygon();
        polygons[i].RandomColor = random.Next();
    }
    Generate();
}
public void Generate()
{
    double first_step = 2 * Math.PI / approx0;
    double second_step = CylHeight / (approx1 - 1);
    double height = 0;
    double middle = CylHeight / 2;

    for (int k = 0; k < approx1; ++k)
    {
        double angle = 0;
        for (int i = 0; i < approx0; ++i)
        {
            int indx = k * approx0 + i;
            vertices[indx] = new Vertex(new[] { Radius[0] * Math.Cos(angle), Radius[1]
                * Math.Sin(angle), middle - height, 1 });
            angle += first_step;
        }
        height += second_step;
    }

    vertices[vertices.Length - 2] = new Vertex(new[] { 0, 0, middle, 1 });
    vertices[vertices.Length - 1] = new Vertex(new[] { 0, 0, middle - height +
        second_step, 1 });

    int pol_indx = 0;
    for (int k = 1; k < approx1; ++k)
    {
        for (int i = 0; i < approx0; ++i)
```

```
45          {
46              polygons[pol_indx] = new Polygon(new Vertex[] { vertices[(k - 1) * approx0
                    + (i + 1 == approx0 ? 0 : i + 1)],
47                                               vertices[(k - 1) * approx0 + i],
48                                               vertices[k * approx0 + i] }, polygons[
                                                     pol_indx].RandomColor);
49              ++pol_indx;
50              polygons[pol_indx] = new Polygon(new Vertex[] { vertices[k * approx0 + (i +
                     1 == approx0 ? 0 : i + 1)],
51                                                   vertices[(k - 1) * approx0 + (i + 1
                                                       == approx0 ? 0 : i + 1)],
52                                                   vertices[k * approx0 + i] }, polygons
                                                       [pol_indx].RandomColor);
53              ++pol_indx;
54          }
55      }
56
57      for (int i = 1; i < approx0; ++i)
58      {
59          polygons[pol_indx] = new Polygon(new Vertex[] { vertices[i - 1], vertices[i],
                vertices[vertices.Length - 2] }, polygons[pol_indx].RandomColor);
60          ++pol_indx;
61      }
62      polygons[pol_indx] = new Polygon(new Vertex[] { vertices[vertices.Length - 2],
            vertices[approx0 - 1], vertices[0] }, polygons[pol_indx].RandomColor);
63      ++pol_indx;
64
65      int down_indx = approx0 * (approx1 - 1);
66      for (int i = 1; i < approx0; ++i)
67      {
68          polygons[pol_indx] = new Polygon(new Vertex[] { vertices[vertices.Length - 1],
                vertices[down_indx + i], vertices[down_indx + i - 1] }, polygons[pol_indx].
                RandomColor);
69          ++pol_indx;
70      }
71      polygons[pol_indx] = new Polygon(new Vertex[] { vertices[down_indx + 0], vertices[
            down_indx + approx0 - 1], vertices[vertices.Length - 1] }, polygons[pol_indx].
            RandomColor);
72      ++pol_indx;
73
74      foreach (var p in polygons)
75      {
76          DVector4 first = new DVector4(new double[] {p.vertecies[1].Point_InLocalSpace.X
                - p.vertecies[0].Point_InLocalSpace.X,
77                                                      p.vertecies[1].Point_InLocalSpace.Y - p
                                                          .vertecies[0].Point_InLocalSpace.Y,
78                                                      p.vertecies[1].Point_InLocalSpace.Z - p
                                                          .vertecies[0].Point_InLocalSpace.Z,
```

```
79                                                  p.vertecies[1].Point_InLocalSpace.W - p
                                                        .vertecies[0].Point_InLocalSpace.W})
                                                        ;
80          DVector4 second = new DVector4(new double[] {p.vertecies[2].Point_InLocalSpace.
                X - p.vertecies[0].Point_InLocalSpace.X,
81                                                  p.vertecies[2].Point_InLocalSpace.Y - p
                                                        .vertecies[0].Point_InLocalSpace.Y,
82                                                  p.vertecies[2].Point_InLocalSpace.Z - p
                                                        .vertecies[0].Point_InLocalSpace.Z,
83                                                  p.vertecies[2].Point_InLocalSpace.W - p
                                                        .vertecies[0].Point_InLocalSpace.W})
                                                        ;
84          p.Normal_InLocalSpace = first * second;
85          p.Normal_InLocalSpace.Normalize();
86      }
87
88  }
```

Вычисления освещения:

```
1   public double[] CalculateIntensity(DVector4 point, DVector4 normal)
2   {
3
4       DVector4 L = new DVector4(LightPos_InWorldSpace.X - point.X, LightPos_InWorldSpace.
            Y - point.Y, LightPos_InWorldSpace.Z - point.Z, 0);
5       double distance = L.GetLength();
6       L.Normalize();
7
8       double I_red = Ia_Material.X * Ka_Material.X;
9       double I_green = Ia_Material.Y * Ka_Material.Y;
10      double I_blue = Ia_Material.Z * Ka_Material.Z;
11
12      I_red += Math.Min(1, Math.Max(0, Il_Material.X * (Kd_Material.X * DVector4.
            DotProduct(L, normal)) / (Parameters[0] * distance + Parameters[1])));
13      I_green += Math.Min(1, Math.Max(0, Il_Material.Y * (Kd_Material.Y * DVector4.
            DotProduct(L, normal)) / (Parameters[0] * distance + Parameters[1])));
14      I_blue += Math.Min(1, Math.Max(0, Il_Material.Z * (Kd_Material.Z * DVector4.
            DotProduct(L, normal)) / (Parameters[0] * distance + Parameters[1])));
15
16      if (DVector4.DotProduct(L, normal) > 0)
17      {
18          DVector4 S = new DVector4(Center.X - point.X, Center.Y - point.Y, -1000 - point
                .Z, 0);
19          DVector4 R = new DVector4(DVector3.Reflect((DVector3)(-L), (DVector3)normal),
                0);
20          S.Normalize();
21          R.Normalize();
22
23          if (DVector4.DotProduct(R, S) > 0)
24          {
```

```
            I_red += Math.Min(1, Math.Max(0, Il_Material.X * Ks_Material.X * Math.Pow(
                DVector4.DotProduct(R, S), P_Material) / (Parameters[0] * distance +
                Parameters[1])));
            I_green += Math.Min(1, Math.Max(0, Il_Material.Y * Ks_Material.Y * Math.Pow
                (DVector4.DotProduct(R, S), P_Material) / (Parameters[0] * distance +
                Parameters[1])));
            I_blue += Math.Min(1, Math.Max(0, Il_Material.Z * Ks_Material.Z * Math.Pow(
                DVector4.DotProduct(R, S), P_Material) / (Parameters[0] * distance +
                Parameters[1])));
        }
    }

    I_red = Math.Min(1, I_red);
    I_green = Math.Min(1, I_green);
    I_blue = Math.Min(1, I_blue);

    return new double[] { I_red, I_green, I_blue };

}

public void LightCalculation()
{
    if (CurVisual == Visualization.FlatShading)
    {
        foreach (var p in polygons)
        {
            if (!p.IsVisible) continue;
            DVector4 polMiddle = new DVector4(p.vertecies.Sum(v => v.Point_InWorldSpace
                .X) / 3,
                                    p.vertecies.Sum(v => v.Point_InWorldSpace.Y) / 3,
                                    p.vertecies.Sum(v => v.Point_InWorldSpace.Z) / 3,
                                    1d);
            double[] result = CalculateIntensity(polMiddle, p.Normal_InWorldSpace);


            p.LightColor = Color.FromArgb((int)Math.Max(0, Math.Min(255, 255 * result
                [0] * MaterialColor.X)), (int)Math.Max(0, Math.Min(255, 255 * result[1]
                * MaterialColor.Y)),
                (int)Math.Max(0, Math.Min(255, 255 * result[2] * MaterialColor.Z)));
        }
    }
    else if (CurVisual == Visualization.GuroShading)
    {
        foreach (var v in vertices)
        {
            DVector4 v_normal = new DVector4(0, 0, 0, 0);
            foreach (var p in v.polygons)
            {
                v_normal += p.Normal_InWorldSpace;
```

```
            }
            v_normal.Normalize();
            double[] result = CalculateIntensity(v.Point_InWorldSpace, v_normal);
            v.LightColor = Color.FromArgb((int)Math.Max(0, Math.Min(255, 255 * result
                [0] * MaterialColor.X)), (int)Math.Max(0, Math.Min(255, 255 * result[1]
                * MaterialColor.Y)),
                (int)Math.Max(0, Math.Min(255, 255 * result[2] * MaterialColor.Z)));
        }
    }
}
```

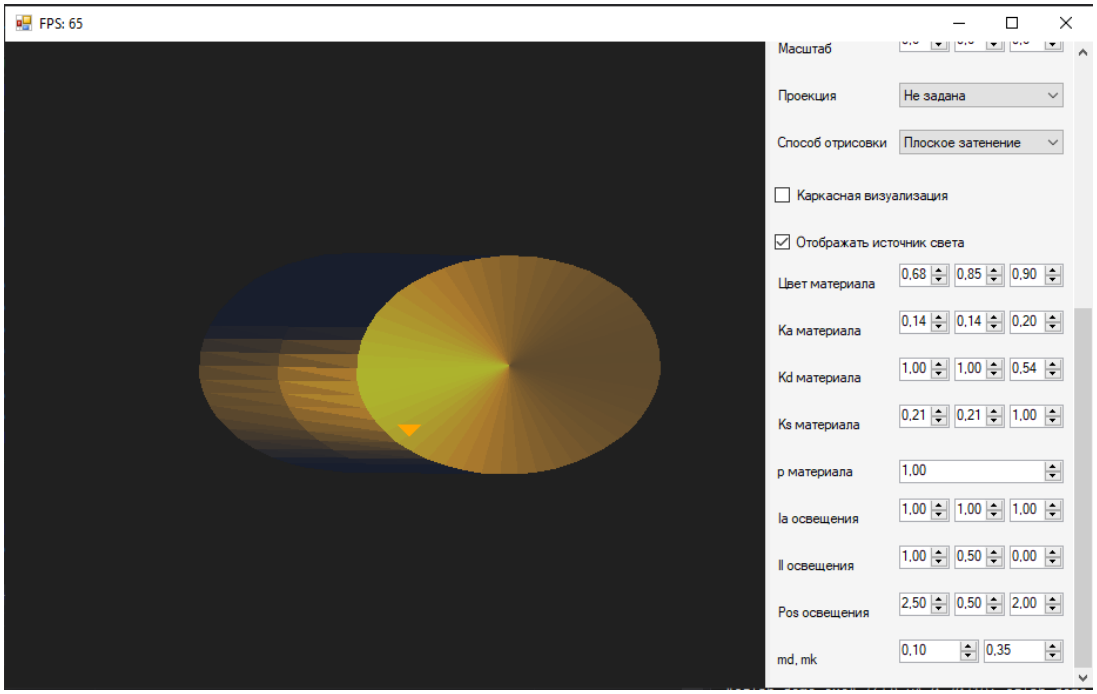
Основной цикл работы:

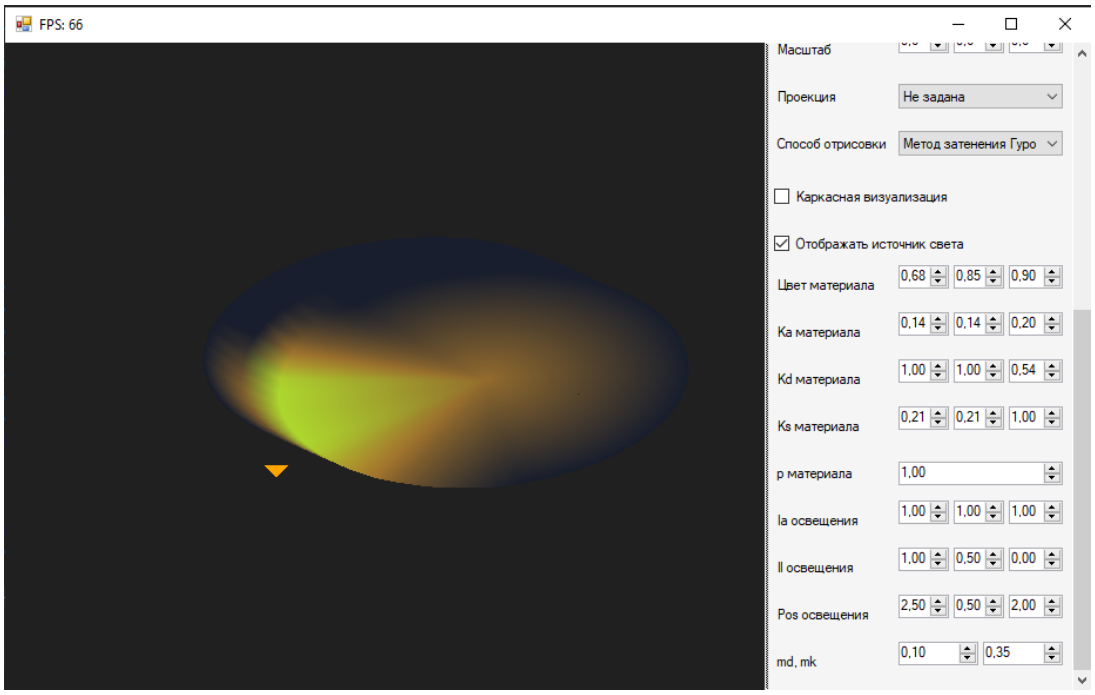
```
protected override void OnDeviceUpdate(object s, DeviceArgs e)
{
    if (0 != ((int)_Commands & (int)Commands.NewFigure))
    {
        _Commands ^= Commands.NewFigure;
        Create();
        _Commands |= Commands.FigureChange;
    }

    if (0 != ((int)_Commands & (int)Commands.FigureChange))
    {
        _Commands ^= Commands.FigureChange;
        Generate();
        _Commands |= Commands.Transform;
    }

    var x_min = vertices.Min(p => (p.Point_InLocalSpace.X));
    var x_max = vertices.Max(p => (p.Point_InLocalSpace.X));
    var y_min = vertices.Min(p => (p.Point_InLocalSpace.Y));
    var y_max = vertices.Max(p => (p.Point_InLocalSpace.Y));
    ViewSize.X = x_max - x_min;
    ViewSize.Y = y_max - y_min;
    Center.X = x_min + ViewSize.X / 2;
    Center.Y = y_min + ViewSize.Y / 2;
    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;
        //
        NormalTransform = DMatrix3.NormalVecTransf(PointTransform);

        foreach (var v in vertices)
```

```
38          {
39              v.Point_InWorldSpace = PointTransform * v.Point_InLocalSpace;
40          }
41          DVector4 LightPosV4 = new DVector4(LightPos.X, LightPos.Y, LightPos.Z, 1);
42          LightPos_InWorldSpace = PointTransform * LightPosV4;
43          foreach (var p in polygons)
44          {
45              p.Normal_InWorldSpace = NormalTransform * p.Normal_InLocalSpace;
46              p.Normal_InWorldSpace.Normalize();
47              p.IsVisible = p.Normal_InWorldSpace.Z < 0;
48          }
49          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)));
50          _Commands |= Commands.ShadingChange;
51      }
52
53      if (0 != ((int)_Commands & (int)Commands.ShadingChange))
54      {
55          _Commands ^= Commands.ShadingChange;
56          LightCalculation();
57      }
58
59      foreach (var p in polygons)
60      {
61          if (!p.IsVisible)
62              continue;
63
64          if (CurVisual == Visualization.OneColor)
65          {
66              e.Surface.DrawTriangle(Color.YellowGreen.ToArgb(), FromViewToPhysicalSpace(
                    p.vertecies[0].Point_InWorldSpace),
67                                                          FromViewToPhysicalSpace(p.
                                                              vertecies[1].
                                                              Point_InWorldSpace),
68                                                          FromViewToPhysicalSpace(p.
                                                              vertecies[2].
                                                              Point_InWorldSpace));
69          }
70          else if (CurVisual == Visualization.RandomColor)
71          {
72              e.Surface.DrawTriangle(p.RandomColor, FromViewToPhysicalSpace(p.vertecies
                    [0].Point_InWorldSpace),
73                                                  FromViewToPhysicalSpace(p.vertecies[1].
                                                      Point_InWorldSpace),
74                                                  FromViewToPhysicalSpace(p.vertecies[2].
                                                      Point_InWorldSpace));
75          }
76          else if (CurVisual == Visualization.FlatShading)
77          {
```

```
78              e.Surface.DrawTriangle(p.LightColor.ToArgb(), FromViewToPhysicalSpace(p.
                    vertecies[0].Point_InWorldSpace),
79                                                  FromViewToPhysicalSpace(p.
                                                        vertecies[1].
                                                        Point_InWorldSpace),
80                                                  FromViewToPhysicalSpace(p.
                                                        vertecies[2].
                                                        Point_InWorldSpace));
81          }
82          else if (CurVisual == Visualization.GuroShading)
83          {
84              DVector2 v1 = FromViewToPhysicalSpace(p.vertecies[0].Point_InWorldSpace);
85              DVector2 v2 = FromViewToPhysicalSpace(p.vertecies[1].Point_InWorldSpace);
86              DVector2 v3 = FromViewToPhysicalSpace(p.vertecies[2].Point_InWorldSpace);
87              e.Surface.DrawTriangle(p.vertecies[0].LightColor.ToArgb(), v1.X, v1.Y,
88                                     p.vertecies[1].LightColor.ToArgb(), v2.X, v2.Y,
89                                     p.vertecies[2].LightColor.ToArgb(), v3.X, v3.Y);
90          }
91
92          if (IsCarcass)
93          {
94              e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.
                    vertecies[0].Point_InWorldSpace), FromViewToPhysicalSpace(p.vertecies
                    [1].Point_InWorldSpace));
95              e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.
                    vertecies[1].Point_InWorldSpace), FromViewToPhysicalSpace(p.vertecies
                    [2].Point_InWorldSpace));
96              e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.
                    vertecies[2].Point_InWorldSpace), FromViewToPhysicalSpace(p.vertecies
                    [0].Point_InWorldSpace));
97          }
98      }
99
100     if (IsLightSource)
101     {
102         DVector2 LightInPhysicalSpace = FromViewToPhysicalSpace(LightPos_InWorldSpace);
103         e.Surface.DrawTriangle(Color.Orange.ToArgb(), new DVector2(LightInPhysicalSpace
                .X, LightInPhysicalSpace.Y),
104                                                 new DVector2(LightInPhysicalSpace.X +
                                                        20, LightInPhysicalSpace.Y),
105                                                 new DVector2(LightInPhysicalSpace.X +
                                                        10, LightInPhysicalSpace.Y + 10));
106     }
107 }
```

4 Демонстрация работы





5 Выводы

В результате выполнения данной лабораторной работы я познакомился с возможностью отрисовки фотореалистичных изображений посредством вычисления интенсивностей цвета для вершин или полигонов фигуры в зависимости от применяемой модели освещения - плоской или Гуро.

Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4-5 по курсу «Компьютерная графика»
Тема: Ознакомление с технологией OpenGL

Студент: Д. С. Ляшун
Преподаватель: А. В. Морозов
Группа: М8О-307Б

Дата:
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Цель: Приобретение практический знаний по работе с библиотекой OpenGL, в частности использованием буферов вершин и шейдеров для отрисовки фотореалистичных 3D фигур в различных моделях освещения.

Задание: Создать графическое приложение с использованием OpenGL. Используя результаты Л.Р.№3, изобразить заданное тело (то же, что и в л.р. №3) с использованием средств OpenGL 2.1. Использовать буфер вершин. Точность аппроксимации тела задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель освещения на GLSL. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

Вариант тела: Прямой эллиптический цилиндр.

2 Теоретические сведения

Возьмем за основу способ генерации фигуры и вычисление интенсивностей цвета такие же, какие они были в Л.Р. 3. При этом важно понимать, что затенение Фонга отличается от Гуро тем, что при его использовании для определения цвета в каждой точке интерполируются не интенсивности отраженного света, а векторы нормалей - это будет осуществляться автоматически при загрузке нормалей как выходные атрибуты в вершинном шейдере OpenGL. Стоит также отметить, что все вычисления, связанные с освещением, будут происходить исключительно во фрагментном шейдере OpenGL.

Для корректной отрисовки фигуры в среде OpenGL необходимо задать матрицу проекций (свойство OpenGL.GL_PROJECTION) и объектно-видовую матрицу (свойство OpenGL.GL_MODELVIEW) - их следует предварительно вычислить и только затем подгрузить в OpenGL. После задания этих матриц будет автоматически производиться следующее преобразование для координат некоторой вершины P :

$$P_{transformed} = M_{projection} * M_{modelview} * P$$

Чтобы вся информация о фигуре хранилось исключительно в GPU необходимо загружать её в буфер вершин VBO. При этом для вызова метода отрисовки OpenGL необходимо знать, в каком порядке следуют вершины в массиве. При работе с прямым эллиптическим цилиндром будем использоваться порядок обхода GL_TRIANGLE_FAN - для оснований, и GL_TRIANGLE_STRIP - для боковой поверхности (вызывая этот метод отрисовки таким обходом для каждого этажа соответственно).

3 Листинг программы

Вершинный шейдер shader1.vert:

```
#version 150 core

attribute vec4 Normal;
attribute vec4 Coord;

out vec3 FragNormale;
out vec3 FragVertex;

uniform mat4 Projection;
uniform mat4 ModelView;
uniform mat4 NormalMatrix;
uniform mat4 ModelMatrix;
uniform mat4 PointMatrix;

void main(void)
{
    FragVertex = vec3(PointMatrix * Coord);
    FragNormale = vec3(NormalMatrix * Normal);
    FragNormale = normalize(FragNormale);
    gl_Position = (Projection * ModelView) * Coord;
}
```

Фрагментный шейдер shader2.frag:

```
#version 150 core

in vec3 FragNormale;
in vec3 FragVertex;

uniform vec3 Ka_Material;
uniform vec3 Kd_Material;
uniform vec3 Ks_Material;
uniform float P_Material;
uniform vec3 Ia_Material;
uniform vec3 Il_Material;
uniform vec3 LightPos;
uniform vec2 Parameters;
uniform vec3 CameraPos;
uniform vec3 FragColor;

void main(void)
{
    vec3 L = vec3(LightPos.x - FragVertex.x, LightPos.y - FragVertex.y, LightPos.z -
        FragVertex.z);
    float dist = length(L);
    L = normalize(L);
```

```
22      vec3 FragNormaleW = normalize(FragNormale);
23
24      float I_red = Ia_Material.x * Ka_Material.x;
25      float I_green = Ia_Material.y * Ka_Material.y;
26      float I_blue = Ia_Material.z * Ka_Material.z;
27
28      I_red += clamp(0, 1, Il_Material.x * Kd_Material.x * dot(L, FragNormaleW) / (
            Parameters[0] * dist + Parameters[1]));
29      I_green += clamp(0, 1, Il_Material.y * Kd_Material.y * dot(L, FragNormaleW) / (
            Parameters[0] * dist + Parameters[1]));
30      I_blue += clamp(0, 1, Il_Material.z * Kd_Material.z * dot(L, FragNormaleW) / (
            Parameters[0] * dist + Parameters[1]));
31
32      if (dot(L, FragNormaleW) > 0)
33      {
34          vec3 S = vec3(CameraPos.x - FragVertex.x, CameraPos.y - FragVertex.y, CameraPos
                .z - FragVertex.z);
35          vec3 R = vec3(reflect(-L, FragNormale));
36
37          S = normalize(S);
38          R = normalize(R);
39
40          if (dot(R, S) > 0)
41          {
42              I_red += clamp(0, 1, Il_Material.x * Ks_Material.x * pow(dot(R, S),
                    P_Material) / (Parameters[0] * dist + Parameters[1]));
43              I_green += clamp(0, 1, Il_Material.y * Ks_Material.y * pow(dot(R, S),
                    P_Material) / (Parameters[0] * dist + Parameters[1]));
44              I_blue += clamp(0, 1, Il_Material.z * Ks_Material.z * pow(dot(R, S),
                    P_Material) / (Parameters[0] * dist + Parameters[1]));
45          }
46      }
47
48      I_red = min(1, I_red);
49      I_green = min(1, I_green);
50      I_blue = min(1, I_blue);
51
52      vec4 result = vec4(FragColor.x * I_red, FragColor.y * I_green, FragColor.z * I_blue
            , 1);
53
54      gl_FragColor = result;
55  }
```

Задание преобразующих матриц:

```
1   private void UpdateProjectionMatrix(DeviceArgs e)
2   {
3       var gl = e.gl;
4
5       gl.MatrixMode(OpenGL.GL_PROJECTION);
```

```
6       pMatrix = Perspective(FieldVision, 1, ClippingPlanes.X, ClippingPlanes.Y);
7       gl.LoadMatrix(pMatrix.ToArray(true));
8   }
9
10  private void UpdateModelViewMatrix(DeviceArgs e)
11  {
12      var gl = e.gl;
13
14      gl.MatrixMode(OpenGL.GL_MODELVIEW);
15      var deg2rad = Math.PI / 180; //  ,
16      var cameraTransform = (DMatrix3)Rotation(deg2rad * CameraPos.X, deg2rad * CameraPos
            .Y, deg2rad * CameraPos.Z);
17      var cameraPosition = cameraTransform * new DVector3(0, 0, CameraDistance);
18      var cameraUpDirection = cameraTransform * new DVector3(0, 1, 0);
19
20      var mMatrix = _PointTransform;
21
22      var vMatrix = LookAt(DMatrix4.Identity, cameraPosition, DVector3.Zero,
            cameraUpDirection);
23
24      ModelViewMatrix = vMatrix * mMatrix;
25      gl.LoadMatrix(ModelViewMatrix.ToArray(true));
26
27      NormalMatrix = DMatrix3.NormalVecTransf(mMatrix);
28
29  }
30
31  private static DMatrix4 Perspective(double verticalAngle, double aspectRatio, double
        nearPlane, double farPlane)
32  {
33      var radians = (verticalAngle / 2) * Math.PI / 180;
34      var sine = Math.Sin(radians);
35      if (nearPlane == farPlane || aspectRatio == 0 || sine == 0)
36          return DMatrix4.Zero;
37      var cotan = Math.Cos(radians) / sine;
38      var clip = farPlane - nearPlane;
39      return new DMatrix4(
40      cotan / aspectRatio, 0, 0, 0,
41      0, cotan, 0, 0,
42      0, 0, -(nearPlane + farPlane) / clip, -(2.0 * nearPlane * farPlane) / clip,
43      0, 0, -1.0, 1.0
44      );
45  }
46
47  private static DMatrix4 LookAt(DMatrix4 matrix, DVector3 eye, DVector3 center,
        DVector3 up)
48  {
49      var forward = (center - eye).Normalized();
50      if (forward.ApproxEqual(DVector3.Zero, 0.00001))
```

```
51          return matrix;
52      var side = (forward * up).Normalized();
53      var upVector = side * forward;
54      var result = matrix * new DMatrix4(
55      +side.X, +side.Y, +side.Z, 0,
56      +upVector.X, +upVector.Y, +upVector.Z, 0,
57      -forward.X, -forward.Y, -forward.Z, 0,
58      0, 0, 0, 1
59      );
60      result.M14 -= result.M11 * eye.X + result.M12 * eye.Y + result.M13 * eye.Z;
61      result.M24 -= result.M21 * eye.X + result.M22 * eye.Y + result.M23 * eye.Z;
62      result.M34 -= result.M31 * eye.X + result.M32 * eye.Y + result.M33 * eye.Z;
63      result.M44 -= result.M41 * eye.X + result.M42 * eye.Y + result.M43 * eye.Z;
64      return result;
65  }
```

Основной цикл работы:

```
1   protected override void OnDeviceUpdate(object s, DeviceArgs e)
2   {
3       var gl = e.gl;
4
5       gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT | OpenGL.
            GL_STENCIL_BUFFER_BIT);
6
7       if (0 != ((int)_Commands & (int)Commands.ChangeProjectionMatrix))
8       {
9           _Commands ^= Commands.ChangeProjectionMatrix;
10          UpdateProjectionMatrix(e);
11          _Commands |= Commands.Transform;
12      }
13
14      if (0 != ((int)_Commands & (int)Commands.NewFigure))
15      {
16          _Commands ^= Commands.NewFigure;
17          Create();
18          _Commands |= Commands.FigureChange;
19      }
20
21      if (0 != ((int)_Commands & (int)Commands.FigureChange))
22      {
23          _Commands ^= Commands.FigureChange;
24          Generate(e);
25      }
26
27      if (0 != ((int)_Commands & (int)Commands.Transform))
28      {
29          _Commands ^= Commands.Transform;
30          UpdateModelViewMatrix(e);
31      }
```

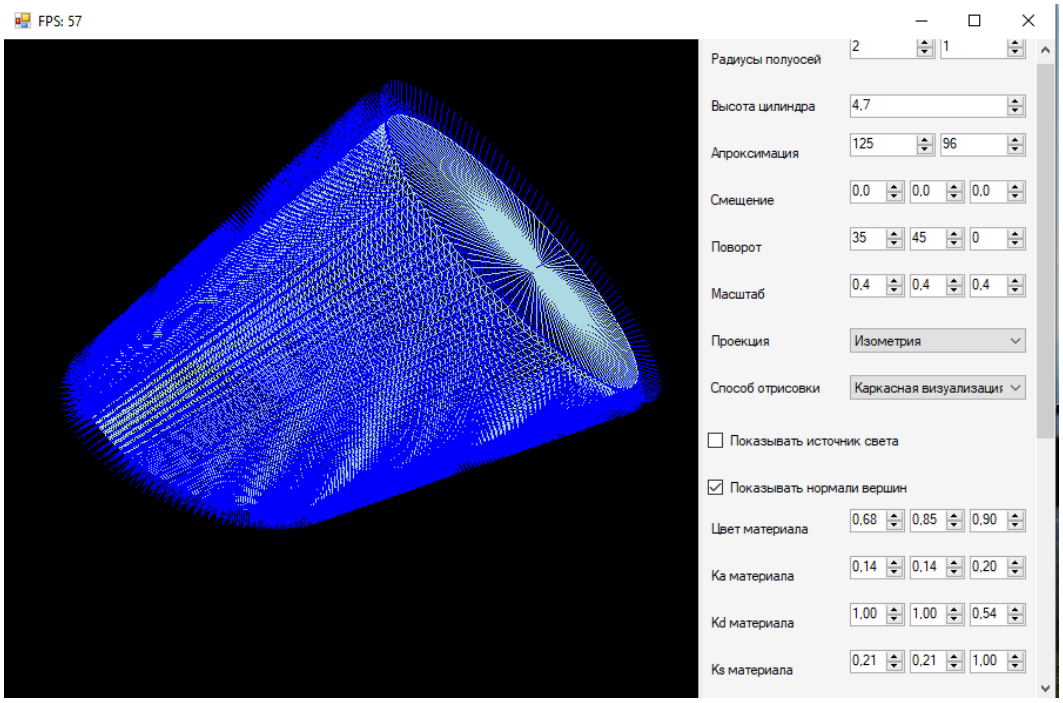
```
32
33      if (0 != ((int)_Commands & (int) Commands.ChangeLightPos))
34      {
35          _Commands ^= Commands.ChangeLightPos;
36
37          gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, LightVertexBuffer[0]);
38          DVector4 LightPosV4 = new DVector4(LightPos, 1);
39          unsafe
40          {
41              LightVertexArray = LightPosV4.ToArray();
42
43              fixed (double* ptr = &LightVertexArray[0])
44              {
45                  gl.BufferData(OpenGL.GL_ARRAY_BUFFER, LightVertexArray.Length * sizeof(
                        double), (IntPtr) ptr, OpenGL.GL_STATIC_DRAW);
46              }
47          }
48
49          gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, LightIndexBuffer[0]);
50          unsafe
51          {
52              fixed (uint* ptr = &LightIndexValues[0])
53              {
54                  gl.BufferData(OpenGL.GL_ELEMENT_ARRAY_BUFFER, LightIndexValues.Length *
                        sizeof(uint), (IntPtr)ptr, OpenGL.GL_STATIC_DRAW);
55              }
56          }
57
58          LightPos_InWorldSpace = _PointTransform * LightPosV4;
59
60      }
61
62      if (CurVisual == Visualization.OneColor)
63      {
64          gl.PolygonMode(OpenGL.GL_FRONT, OpenGL.GL_FILL);
65          gl.Color(MaterialColor.X, MaterialColor.Y, MaterialColor.Z);
66      }
67      else if (CurVisual == Visualization.RandomColor)
68      {
69          gl.PolygonMode(OpenGL.GL_FRONT, OpenGL.GL_FILL);
70          gl.EnableClientState(OpenGL.GL_COLOR_ARRAY);
71          unsafe
72          {
73              gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, vertexBuffer[0]);
74              gl.ColorPointer(3, OpenGL.GL_BYTE, sizeof(Vertex), (IntPtr)(sizeof(float) *
                     8));
75          }
76      }
77      else if (CurVisual == Visualization.NoPolygons)
```

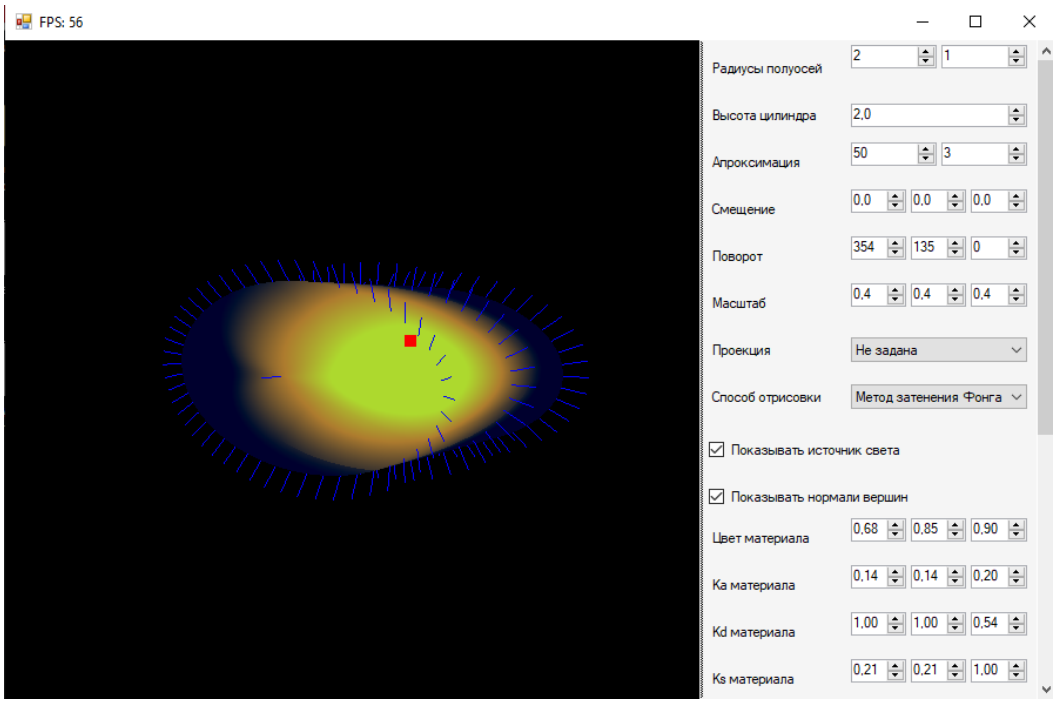
```
78      {
79          gl.PolygonMode(OpenGL.GL_FRONT, OpenGL.GL_LINE);
80          gl.Color(MaterialColor.X, MaterialColor.Y, MaterialColor.Z);
81      }
82      else if (CurVisual == Visualization.PhongShading)
83      {
84          gl.PolygonMode(OpenGL.GL_FRONT, OpenGL.GL_FILL);
85      }
86
87      gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, vertexBuffer[0]);
88      gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, indexBuffer[0]);
89
90      if (CurVisual == Visualization.PhongShading)
91      {
92          gl.UseProgram(prog_shader);
93
94          UpdateLightValues(e);
95          gl.UniformMatrix4(uniform_ModelView, 1, true, ConvertToFloatArray(
                ModelViewMatrix));
96          gl.UniformMatrix4(uniform_Projection, 1, true, ConvertToFloatArray(pMatrix));
97          gl.UniformMatrix4(uniform_NormalMatrix, 1, true, ConvertToFloatArray(
                NormalMatrix));
98          gl.UniformMatrix4(uniform_PointMatrix, 1, true, ConvertToFloatArray(
                _PointTransform));
99
100         gl.EnableVertexAttribArray((uint)attribute_normale);
101         gl.EnableVertexAttribArray((uint)attribute_coord);
102         unsafe
103         {
104             gl.VertexAttribPointer((uint)attribute_normale, 4, OpenGL.GL_FLOAT, false,
                    sizeof(Vertex), (IntPtr)(4 * sizeof(float)));
105             gl.VertexAttribPointer((uint)attribute_coord, 4, OpenGL.GL_FLOAT, false,
                    sizeof(Vertex), (IntPtr)0);
106         }
107     }
108     else
109     {
110         gl.UseProgram(0);
111         gl.EnableClientState(OpenGL.GL_VERTEX_ARRAY);
112         unsafe
113         {
114             gl.VertexPointer(4, OpenGL.GL_FLOAT, sizeof(Vertex), (IntPtr)0);
115         }
116     }
117
118     gl.DrawElements(OpenGL.GL_TRIANGLE_FAN, approx0 + 2, OpenGL.GL_UNSIGNED_INT, (
            IntPtr)0);
119     for (int i = 1; i < approx1; ++i)
120     {
```

```
121         gl.DrawElements(OpenGL.GL_TRIANGLE_STRIP, 2 * (approx0 + 1), OpenGL.
                GL_UNSIGNED_INT, (IntPtr)((approx0 + 2 + (i - 1) * 2 * (approx0 + 1)) *
                sizeof(uint)));
122     }
123     gl.DrawElements(OpenGL.GL_TRIANGLE_FAN, approx0 + 2, OpenGL.GL_UNSIGNED_INT, (
            IntPtr)((approx0 + 2 + (approx1 - 1) * 2 * (approx0 + 1)) * sizeof(uint)));
124
125     if (CurVisual == Visualization.PhongShading)
126     {
127         gl.DisableVertexAttribArray((uint)attribute_normale);
128         gl.DisableVertexAttribArray((uint)attribute_coord);
129         gl.UseProgram(0);
130     }
131     else
132     {
133         gl.DisableClientState(OpenGL.GL_VERTEX_ARRAY);
134     }
135
136     gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
137     gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, 0);
138
139     if (CurVisual == Visualization.RandomColor)
140     {
141         gl.DisableClientState(OpenGL.GL_COLOR_ARRAY);
142     }
143
144     if (isLightActive)
145     {
146         gl.Color(0.99, 0, 0);
147         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, LightVertexBuffer[0]);
148         gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, LightIndexBuffer[0]);
149
150         gl.EnableClientState(OpenGL.GL_VERTEX_ARRAY);
151         unsafe
152         {
153             gl.VertexPointer(4, OpenGL.GL_DOUBLE, sizeof(DVector4), (IntPtr)0);
154         }
155
156         gl.PointSize(10);
157         gl.DrawElements(OpenGL.GL_POINTS, 1, OpenGL.GL_UNSIGNED_INT, (IntPtr)0);
158
159         gl.DisableClientState(OpenGL.GL_VERTEX_ARRAY);
160         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
161         gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, 0);
162     }
163
164     if (isNormalActive)
165     {
166         gl.Color(0, 0, 0.99);
```

```
167         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, normalDataBuffer[0]);
168         gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, normalIndexBuffer[0]);
169 
170         gl.EnableClientState(OpenGL.GL_VERTEX_ARRAY);
171         unsafe
172         {
173             gl.VertexPointer(4, OpenGL.GL_DOUBLE, sizeof(DVector4), (IntPtr)0);
174         }
175 
176         gl.DrawElements(OpenGL.GL_LINES, normalPoints.Length, OpenGL.GL_UNSIGNED_INT, (
                IntPtr)0);
177 
178         gl.DisableClientState(OpenGL.GL_VERTEX_ARRAY);
179         gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
180         gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, 0);
181     }
182 
183 }
```

4 Демонстрация работы





5 Выводы

В результате выполнения данной лабораторной работы я познакомился с возможностями OpenGL по отрисовки 3D фигур, в частности, использованием буфера вершин, шейдеров, подгружаемых глобальных преобразующих матриц и многого другого. Полученные знания я применил для отрисовки прямого эллиптического цилиндра с затенением по Фонгу.

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу «Компьютерная графика»
Тема: Создание шейдерных анимационных эффектов в OpenGL 2.1

Студент: Д. С. Ляшун
Преподаватель: А. В. Морозов
Группа: М8О-307Б
Дата:
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Цель: Приобретение практических данных по созданию простых шейдерных анимаций в OpenGL.

Задание: Для поверхности, созданной в л.р. №5, обеспечить выполнение следующего шейдерного эффекта.

Вариант тела: Анимация. Координата Y изменяется по закону $Y = Y \cdot cos(t + Y)$

2 Теоретические сведения

Для создания требуемого шейдерного эффекта достаточно немного изменить вершинный шейдер - менять указанным способом координаты Y загружаемых вершин:

$$Y = Y \cdot cos(t + Y)$$

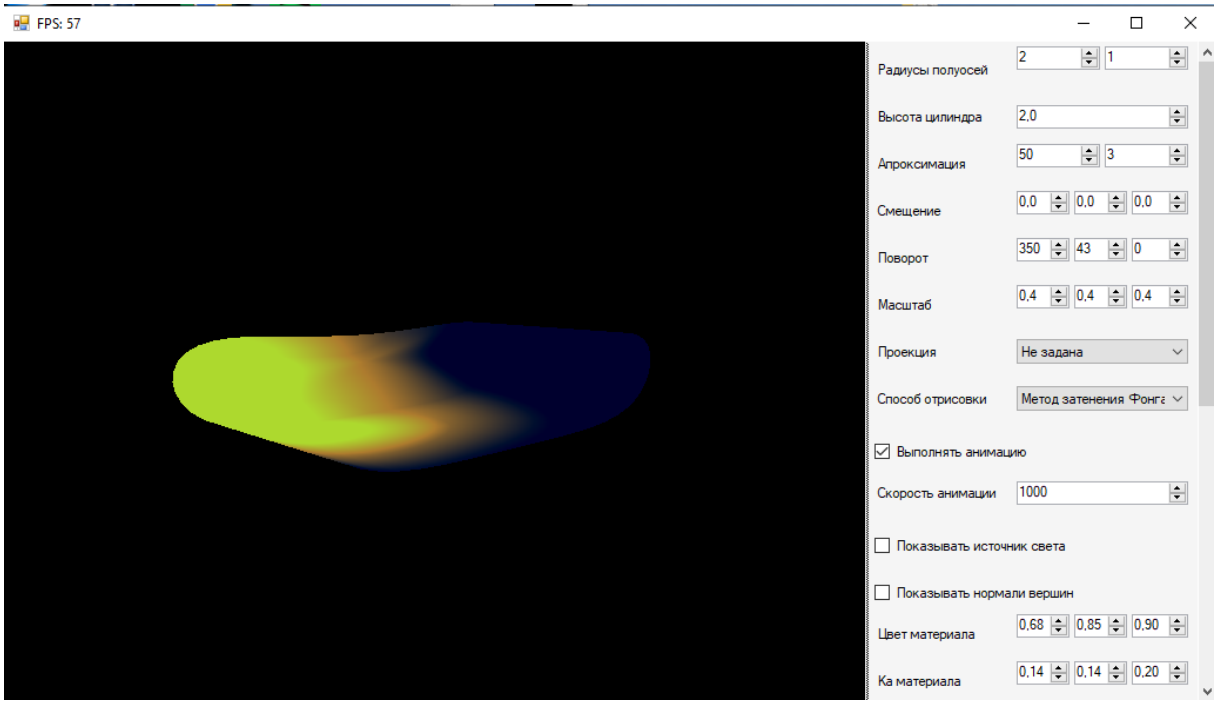
Обновленная величина времени t будет подгружаться в шейдер каждый раз при выполнении основного цикла работы программы. Стоит отметить, что при изменении координат вершин не требуется пересчитывать все ранее вычисленные нормали фигуры - изменения при анимации незначительные и в принципе этого не требуют.

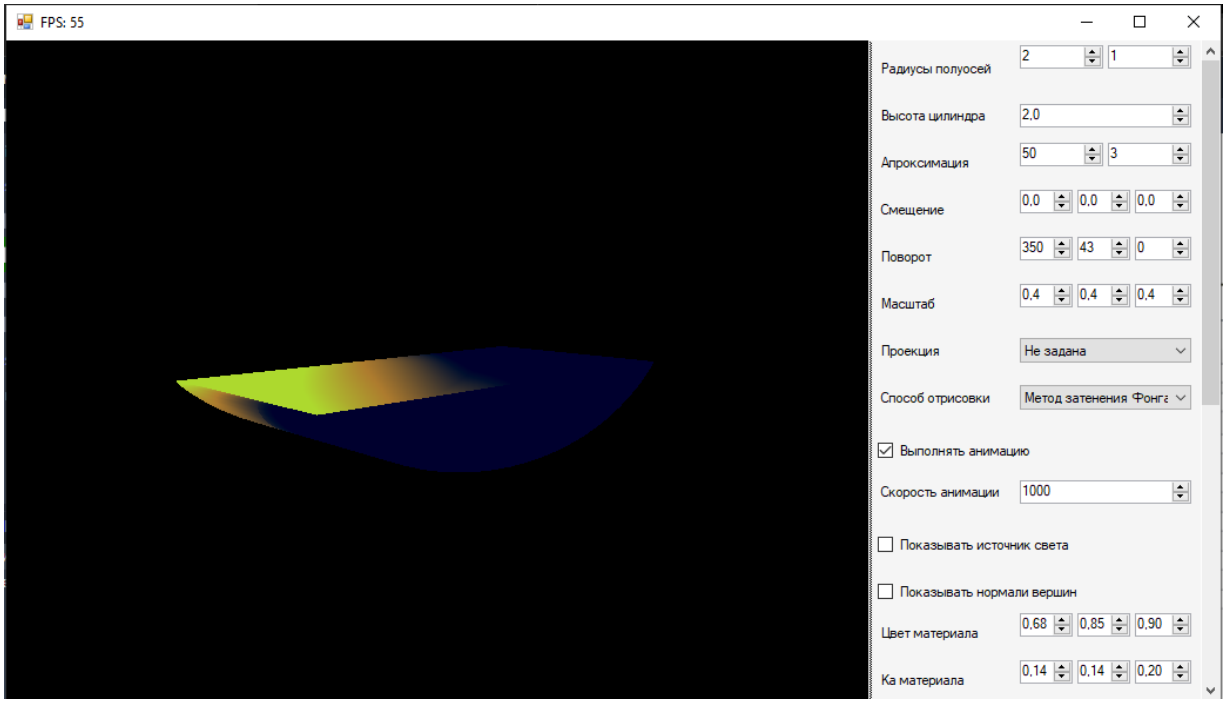
3 Листинг программы

Обновленный вид вершинного шейдера:

```
#version 150 core

attribute vec4 Normal;
attribute vec4 Coord;

out vec3 FragNormale;
out vec3 FragVertex;

uniform mat4 Projection;
uniform mat4 ModelView;
uniform mat4 NormalMatrix;
uniform mat4 ModelMatrix;
uniform mat4 PointMatrix;

uniform float Time;

void main(void)
{
    vec4 Coord_now = Coord;
    if (Time > 0)
        Coord_now.y = Coord.y * cos(Time + Coord.y);
    FragVertex = vec3(PointMatrix * Coord_now);
    FragNormale = vec3(NormalMatrix * Normal);
    FragNormale = normalize(FragNormale);
    gl_Position = (Projection * ModelView) * Coord_now;
}
```

4 Демонстрация работы





5 Выводы

В результате выполнения данной лабораторной работы я познакомился с возможностью создания простых шейдерных анимаций в OpenGL.

Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Компьютерная графика»
Тема: Построение плоских полиномиальных кривых

Студент: Д. С. Ляшун
Преподаватель: А. В. Морозов
Группа: М8О-307Б

Дата:
Оценка:
Подпись:

Москва, 2021

1 Постановка задачи

Цель: Приобретение практических навыков по работе с графиками сложных кривых, используя библиотеку OpenGL.

Задание: Написать программу, строящую полиномиальную кривую по заданным точкам. Обеспечить возможность изменения позиции точек и, при необходимости, значений касательных векторов и натяжения

Вариант тела: Интерполяционный многочлен Лагранжа по шести точкам.

2 Теоретические сведения

В общем виде интерполяционный многочлен в форме Лагранжа записывается в следующем виде:

$$L(x) = \sum_{i=0}^{n}(f(x_i) \cdot \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j})$$

Данный вариант предполагает всего 6 точек, которые будут задавать многочлен - их значения можно будет явно настраивать в графическом интерфейсе приложения.

Непосредственную отрисовку графика кривой будем осуществлять в OpenGL, в частности, используя буфер вершин VBO.

3 Листинг программы

Вычисление многочлена Лагранжа:

```
double LagrangePolynomial(double x, Point[] args)
{
    double Result = 0;
    for (int i = 0; i < args.Length; ++i)
    {
        double product_res = 1;
        for (int j = 0; j < args.Length; ++j)
        {
            if (i == j) continue;

            product_res *= (x - args[j].x) / (args[i].x - args[j].x);

        }
        Result += args[i].y * product_res;
    }
    return Result;
}
```

Основной ход работы программы:

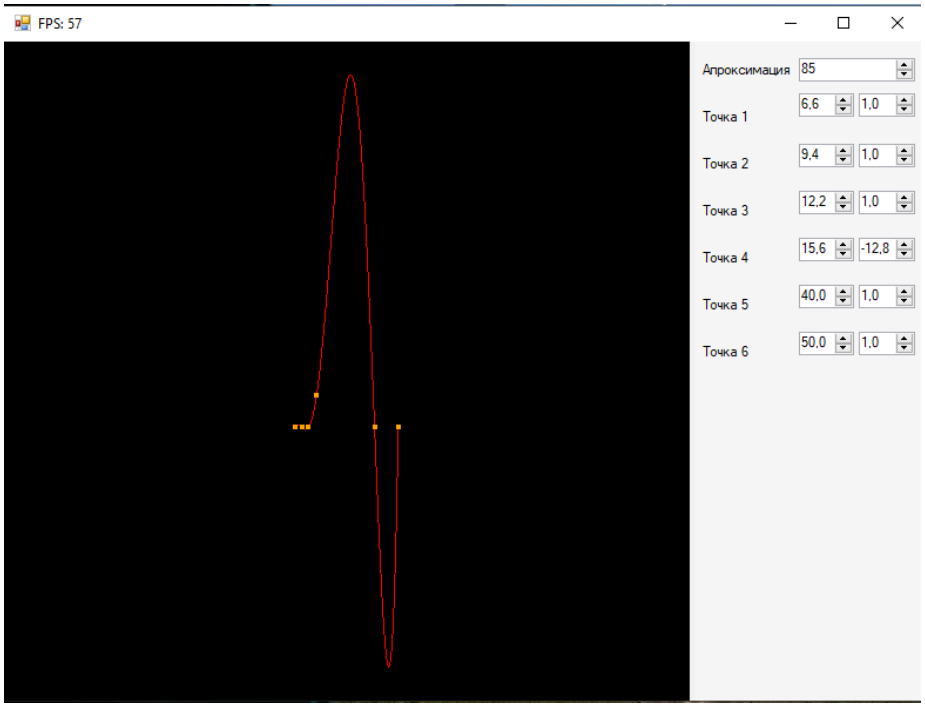
```
protected void WorkWithPoints(DeviceArgs e, Point[] points, Point[] pointsInWindow,
    uint[] indices, uint ver_id, uint ind_id)
{
    var gl = e.gl;

    for (int i = 0; i < pointsInWindow.Length; ++i)
    {
        pointsInWindow[i] = new Point();
        pointsInWindow[i] = FromViewToPhysicalSpace(points[i]);
        pointsInWindow[i].x /= Width;
        pointsInWindow[i].y /= Heigh;
        pointsInWindow[i].x -= 0.5d;
        pointsInWindow[i].y -= 0.5d;
        pointsInWindow[i].y *= -1;
        pointsInWindow[i].x *= 2;
        pointsInWindow[i].y *= 2;
        Console.WriteLine(i + " " + pointsInWindow[i].x + " " + pointsInWindow[i].y);
    }

    unsafe
    {
        gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, vertexBuffer[ver_id]);

        fixed (Point* ptr = &pointsInWindow[0])
        {
            gl.BufferData(OpenGL.GL_ARRAY_BUFFER, pointsInWindow.Length * sizeof(Point)
```

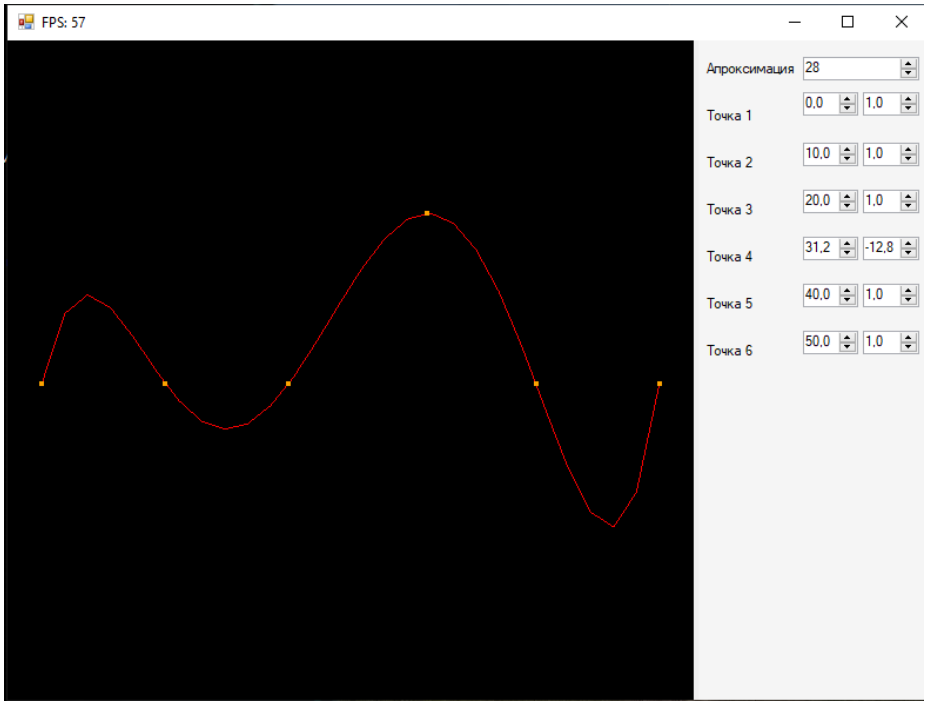
```
                , (IntPtr)ptr, OpenGL.GL_STATIC_DRAW);
        }

        gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);

        gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, indexBuffer[ind_id]);

        fixed (uint* ptr = &indices[0])
        {
            gl.BufferData(OpenGL.GL_ELEMENT_ARRAY_BUFFER, indices.Length * sizeof(uint)
                , (IntPtr)ptr, OpenGL.GL_STATIC_DRAW);
        }
        gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, 0);
    }
}

protected void DrawGraphic(DeviceArgs e, uint ver_id, uint ind_id, uint sz, uint
    draw_type, double r, double g, double b)
{
    var gl = e.gl;

    gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, vertexBuffer[ver_id]);
    gl.EnableClientState(OpenGL.GL_VERTEX_ARRAY);
    unsafe
    {
        gl.VertexPointer(4, OpenGL.GL_DOUBLE, sizeof(Point), (IntPtr)0);
    }

    gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, indexBuffer[ind_id]);

    gl.Color(r, g, b);
    gl.PointSize(sz);
    gl.DrawElements(draw_type, indices.Length, OpenGL.GL_UNSIGNED_INT, (IntPtr)0);

    gl.DisableClientState(OpenGL.GL_VERTEX_ARRAY);
    gl.BindBuffer(OpenGL.GL_ARRAY_BUFFER, 0);
    gl.BindBuffer(OpenGL.GL_ELEMENT_ARRAY_BUFFER, 0);
}

protected override void OnDeviceUpdate(object s, DeviceArgs e)
{
    var gl = e.gl;
    gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT | OpenGL.
        GL_STENCIL_BUFFER_BIT);

    if (0 != ((int)_Commands & (int)Commands.Create))
    {
        _Commands ^= Commands.Create;
        points = new Point[PointsCount];
```

```
71          pointsInWindow = new Point[PointsCount];
72          indices = new uint[PointsCount];
73          _Commands |= Commands.ChangeFigure;
74      }
75
76      if (0 != ((int)_Commands & (int)Commands.ChangeFigure))
77      {
78          _Commands ^= Commands.ChangeFigure;
79
80          generators = new Point[] {new Point(point1), new Point(point2), new Point(
                point3),
81                                                                  new Point(point4),
                                                                        new Point(
                                                                    point5), new
                                                                    Point(point6)
                                                                    };
82          x_min = double.MaxValue;
83          x_max = double.MinValue;
84          y_min = double.MaxValue;
85          y_max = double.MinValue;
86
87          for (int i = 0; i < generators.Length; ++i)
88          {
89              x_min = Math.Min(x_min, generators[i].x);
90              x_max = Math.Max(x_max, generators[i].x);
91          }
92
93          double x_cur = x_min;
94          double step = (x_max - x_min) / (PointsCount - 1);
95
96          for (int i = 0; i < points.Length; ++i)
97          {
98
99              points[i] = new Point(x_cur, LagrangePolynomial(x_cur, generators));
100             indices[i] = (uint)i;
101
102             x_cur += step;
103
104             y_min = Math.Min(y_min, points[i].y);
105             y_max = Math.Max(y_max, points[i].y);
106         }
107
108         _Commands |= Commands.ChangeWindow;
109     }
110
111     if (0 != ((int)_Commands & (int)Commands.ChangeWindow))
112     {
113         _Commands ^= Commands.ChangeWindow;
114
```

```
115         gl.MatrixMode(OpenGL.GL_PROJECTION);
116         gl.LoadIdentity();
117 
118         gl.MatrixMode(OpenGL.GL_MODELVIEW);
119         gl.LoadIdentity();
120 
121         Width = e.Width;
122         Heigh = e.Heigh;
123         ViewSize.X = x_max - x_min;
124         ViewSize.Y = y_max - y_min;
125         AutoScale.X = .9 * Width / ViewSize.X;
126         AutoScale.Y = .9 * Heigh / ViewSize.Y;
127         AutoScale.X = AutoScale.Y = Math.Min(AutoScale.X, AutoScale.Y);
128         Automove.X = Width / 2 - (x_min + x_max) / 2 * AutoScale.X;
129         Automove.Y = Heigh / 2 - (y_min + y_max) / 2 * AutoScale.Y;
130         gl.GenBuffers(2, vertexBuffer);
131         gl.GenBuffers(2, indexBuffer);
132         WorkWithPoints(e, points, pointsInWindow, indices, 0, 0);
133         WorkWithPoints(e, generators, generatorsInWindow, gen_indices, 1, 1);
134     }
135 
136     DrawGraphic(e, 0, 0, 2, OpenGL.GL_LINE_STRIP, 0.99, 0, 0);
137     DrawGraphic(e, 1, 1, 4, OpenGL.GL_POINTS, 0.99, 0.64, 0);
138 
139 }
```

4 Демонстрация работы





5 Выводы

В результате выполнения данной лабораторной работы я познакомился построением плоских полиномиальных кривых с использованием технологий OpenGL.