Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Курсовая работа по курсу «Компьютерная графика»

Тема: «Поверхность вращения. Образующая — NURBS кривая 4-го порядка»

Студент: М. А. Инютин
Преподаватель: А. В. Морозов
Группа: М8О-307Б-19
Дата:
Оценка:
Подпись:

Москва, 2021

Поверхность вращения. Образующая — NURBS кривая 4-го порядка

Задача: 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 буффер вершин Open GL. Реализовать возможность отображения опорных точек, направляющих и других данных по которым формируется порция поверхности и отключения каркасной визуализации.

1 Описание

Освещение

Для построения фотореалистичного изображения буду использовать простую модель освещения:

$$I = I_a + I_d + I_s$$

где I_a — фоновая составляющая, I_d — рассеянная составляющая, I_a — зеркальная составляющая.

Фоновая составляющая

Даже при отсутсвии света в комнате мы может различать объекты, потому что лучи отражаются от всех поверхностей в комнате. Расчёт всех отражений слишком сложный, поэтому для простоты каждый объект получает часть фонового освещения:

$$I_a = k_a \cdot i_a$$

где k_a — свойство материала воспринимать фоновое освещени, i_a — мощность фонового освещения.

Рассеянная составляющая

Рассеянное отражение света происходит, когда свет как бы проникает под поверхность объекта, поглощается, а затем вновь испускается. При этом положение наблюдателя не имеет значения, так как диффузно отраженный свет рассеивается равномерно по всем направлениям.

Интенсивность света обратно пропорциональна квадрату расстояния от источника, следовательно, объект, лежащий дальше от него, должен быть темнее. Для простоты буду применять модель $d + K$, так как в модели обратных квадратов для $|d| \leqslant 1$ будет осветление объекта, что нужно отдельно обрабатывать. Константа K в этом выражении подбиратеся из соображений эстетики.

Выражение для рассеянного освещения имеет следующий вид:

$$I_d = \frac{k_d \cdot i_l}{d + K} \cdot \cos(\vec{L}, \vec{N})$$

где k_d — свойство материала воспринимать рассеянное освещение, i_l — интенсивность точеченого источника, $\vec{L}$ — направление из точки на источник света, $\vec{N}$ — вектор нормали в точке, K — произвольная постоянная, d — расстояние от источника света до точки.

Зеркальная составляющая

Интенсивность зеркально отраженного света зависит от угла падения, длины волны падающего света и свойств вещества отражающей поверхности. Зеркальное отражение света является направленным. Так как физические свойства зеркального отражения очень сложны, используется коээфициент глянцевости материала:

$$I_s = \frac{k_s \cdot i_l}{d + K} \cdot \cos^p(\vec{R}, \vec{S})$$

где k_s — свойство материала воспринимать зекральное освещение, $\vec{R}$ — вектор отражённого от поверхности луча, $\vec{S}$ — вектор наблюдения, p — глянцевость материала.

OpenGL и шейдеры

OpenGL (Open Graphics Library) — спецификация, определяющая платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

Для отображения на экран необходимо использовать Vertex Buffer. Этот метод хранит инфморацию об объекте непосредственно в видеопамяти. Напрямую обратиться к ней нельзя, OpenGL предосталвяет возможность создания буферов вершин и индексов, куда следует скопировать данные об объекте.

Шейдер (Shader) — программа для процессора графической карты (GPU), управляющая поведением шейдерной стадии графического конвейера и занимающаяся обработкой соответствующих входных данных.

В работе используются вершинный и фрагментный шейдеры. Первый трансформирует вершины из локального пространства в пространство камеры. Второй отвечает за закрасу геометрического объекта, интерполирует параметры и расчитывает цвет отдельно взятого пикселя.

Шейдеры можно писать на GLSL (Graphics Library Shader Language) — высокоуровневом языке используемом для шейдеров OpenGL. Синтаксис языка очень похож на синтаксис C.

Полиномиальная кривая и поверхность

NURBS-кривая — частный случай B-сплайна. Отличие состоит в наличии весов у вершин и в представлении координат кривой в виде дроби. Веса вершин задаются пользователем.

Координаты кривой степени k вычисляются по формуле: $r(t) = \frac{\sum_{i=0}^{n} N_{i,k} \cdot P_i \cdot w_i}{\sum_{i=0}^{n} N_{i,k} \cdot w_i}$

$N_{i,p}$ определяется рекуррентными соотношениями:

$$N_{i,0}(t) = \begin{cases} 1, t \in [u_i, u_{i+1}), \\ 1, t \notin [u_i, u_{i+1}) \end{cases}$$

$N_{i,p}(t) = \frac{t-u_i}{u_{i+p}-u_i} \cdot N_{i,p-1}(t) + \frac{u_{i+p+1}-t}{u_{i+p+1}-u_{i+1}} \cdot N_{i+1,p-1}(t)$

Поверхность вращения — поверхность, образуемая вращения кривой вокруг оси. В курсовой работе реализован поворот кривой относительно прямой, параллельной оси абцисс.

2 Исходный код

Файл *Curve.cs* содержит код для генерации NURBS-кривой 4 порядка. В ходе табулирования функциия, задающей кривую, вычисляются коэффициенты $N_{i,p}$ и координаты точки для какого-то t.

```
1  private double u(int i)
2  {
3      return segments[i];
4  }
5
6  private double Nip(int i, int p, double t)
7  {
8      if (p == 0)
9      {
10         return ((u(i) <= t) && (t < u(i + 1))) ? 1.0 : 0.0;
11     }
12     else
13     {
14         return (t - u(i)) / (u(i + p) - u(i)) * Nip(i, p - 1, t) + (u(i + p + 1) - t) /
               (u(i + p + 1) - u(i + 1)) * Nip(i + 1, p - 1, t);
15     }
16 }
17
18 public void CalcCurve()
19 {
20     GenSegments();
21     CountData = steps * (CountPoints - DEGREE + 2);
22     Data = new List<Vector4D>(CountData);
23     double t = DEGREE - 1;
24     for (int i = 0; i < CountPoints - DEGREE + 2; ++i)
25     {
26         for (int j = 0; j < steps; ++j)
27         {
28             Vector4D tt = new Vector4D();
29             double qq = 0;
30             for (int ii = 0; ii < CountPoints; ++ii)
31             {
32                 double curNip = Nip(ii, DEGREE, t);
33                 tt = tt + curNip * W[ii] * Points[ii];
34                 qq = qq + curNip * W[ii];
35             }
36             Data.Add(tt / qq);
37             t += step;
38         }
39     }
40 }
```

В *Figure.cs* задаётся сама фигура. Сначала вычисляется кривая с заданными параметрами, по этим точкам генерируется поверхность вращения. После генерации точек, образующих поверхность, задаются полигоны, вычисляются вектора нормали к вершинам.

```
41 public Figure(int paramPhi, int paramCurve, List<Vector4D> controlPoints, List<double>
       weights, Misc.Colour col)
42 {
43     ParamPhi = paramPhi;
44     ParamCurve = paramCurve - 1;
45     DeltaPhi = Misc.ToRadians(Misc.MAX_DEG / (double)(ParamPhi));
46     ControlPoints = controlPoints;
47     Weights = weights;
48     FigureColour = col;
49     GenVertices();
50     GenFigure();
51     GenVertexNormals();
52 }
53
54 private void GenVertices()
55 {
56     Vertices = new List<Vertex>();
57     Curve nurbs = new Curve(ParamCurve, ControlPoints, Weights);
58     LayerVertices = new List< List<Vertex> >();
59     for (int i = 0; i < nurbs.CountData; ++i)
60     {
61         double curZ = nurbs.Data[i].Y;
62         double curXY = (1 + nurbs.Data[i].X) / 2;
63         double stepPhi = 0;
64         List<Vertex> curLayerVertices = new List<Vertex>();
65         for (int j = 0; j < ParamPhi; ++j)
66         {
67             Vertex curVertex = new Vertex(curXY * Math.Cos(stepPhi), curXY * Math.Sin(
                   stepPhi), curZ, FigureColour);
68             curLayerVertices.Add(curVertex);
69             Vertices.Add(curVertex);
70             stepPhi = stepPhi + DeltaPhi;
71         }
72         LayerVertices.Add(curLayerVertices);
73     }
74     CenterLow = new Vertex(0, 0, Vertices[Vertices.Count - 1].Data.Z, FigureColour);
75     CenterHigh = new Vertex(0, 0, Vertices[0].Data.Z, FigureColour);
76     Vertices.Add(CenterLow);
77     Vertices.Add(CenterHigh);
78     for (int i = 0; i < Vertices.Count; ++i)
79     {
80         Vertices[i].Id = i;
81     }
82 }
```

```
83
84  private void GenFigure()
85  {
86      Polygons = new List<Polygon>();
87      GenCenter();
88  }
89
90  private void GenCenter()
91  {
92      for (int i = 0; i < LayerVertices.Count - 1; ++i)
93      {
94          for (int j = 0; j < ParamPhi; ++j)
95          {
96              Vertex a = LayerVertices[i][j];
97              Vertex b = LayerVertices[i][(j + 1) % ParamPhi];
98              Vertex c = LayerVertices[i + 1][(j + 1) % ParamPhi];
99              Vertex d = LayerVertices[i + 1][j];
100             Polygons.Add(new Polygon(a, c, b));
101             Polygons.Add(new Polygon(c, a, d));
102         }
103     }
104 }
105
106 public void GenVertexNormals()
107 {
108     foreach (Vertex vert in Vertices)
109     {
110         vert.Normal = new Vector4D();
111         vert.Normal.W = 0;
112     }
113     foreach (Polygon poly in Polygons)
114     {
115         Vector4D polyN = poly.NormalVector();
116         polyN.Normalize();
117         foreach (Vertex vert in poly.Data)
118         {
119             vert.Normal = vert.Normal + polyN;
120             vert.Normal.W += 1;
121         }
122     }
123     foreach (Vertex vert in Vertices)
124     {
125         vert.Normal.X = vert.Normal.X / vert.Normal.W;
126         vert.Normal.Y = vert.Normal.Y / vert.Normal.W;
127         vert.Normal.Z = vert.Normal.Z / vert.Normal.W;
128         vert.Normal.W = 0;
129         vert.Normal.Normalize();
130     }
131 }
```

Geometry.cs содержит классы векторов и матрицы, методы для работы с ними. В *Misc.cs* заданы константы, используемые во всей программе. Вся работа с интерфейсом, компиляция шейдеров, создание буфера вершин описаны в *MainWindow.cs*.

Вершинный шейдер *Phong.vert* преобразует локальные координаты в видовые, затем передаёт их фрагментному шейдеру.

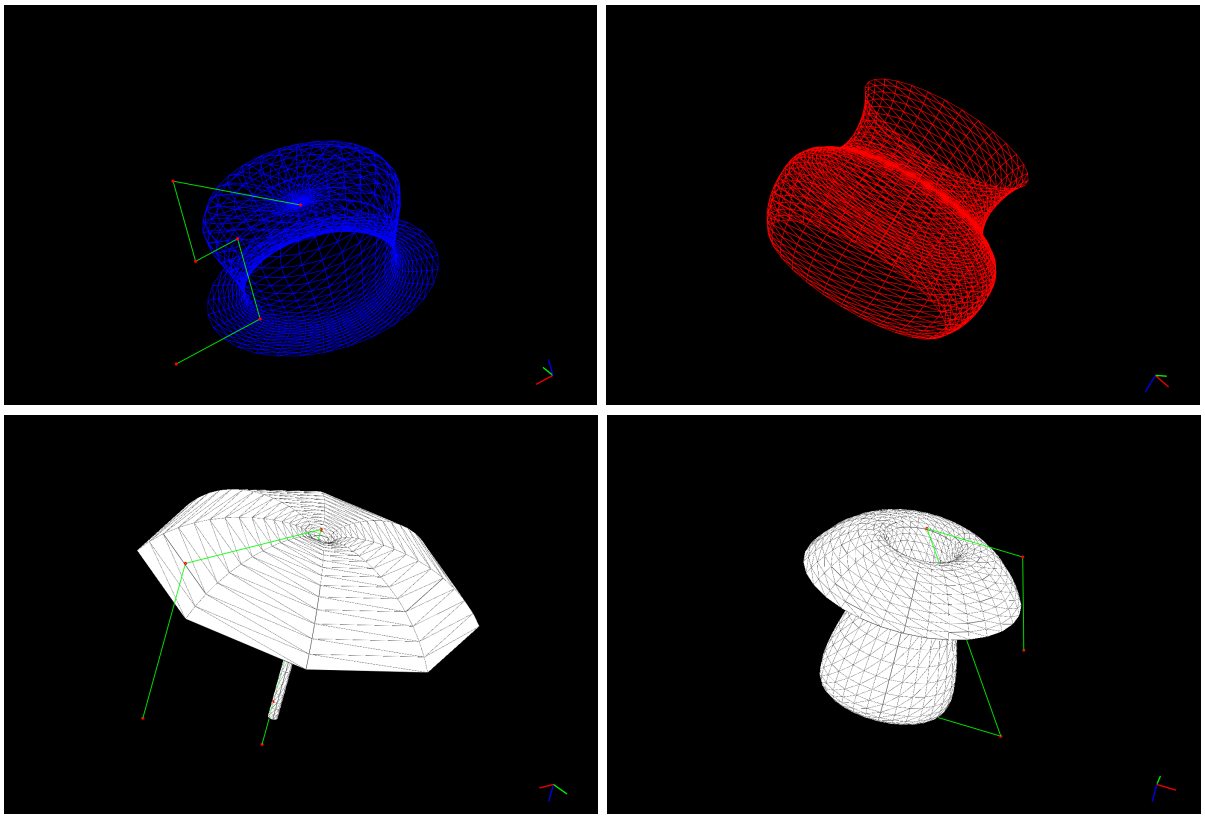
```
#version 150 core

in vec3 cord3f;
in vec3 col3f;
in vec3 norm3f;
out vec3 position;
out vec3 normal;
uniform mat4 proj4f;
uniform mat4 view4f;
uniform mat4 model4f;
uniform bool useSingleColor;
uniform vec3 singleColor3f;
uniform bool moveToCorner;

const vec4 cornerVec = vec4(0.85f, -0.85f, 0, 0);

void main(void) {
    vec4 vertexPos = vec4(cord3f, 1.0f);
    vertexPos = (proj4f * view4f * model4f) * vertexPos;
    vec4 vertexNormal = vec4(norm3f, 0.0f);
    vertexNormal = (view4f * model4f) * vertexNormal;
    position = vertexPos.xyz;
    normal = normalize(vertexNormal.xyz);
    gl_Position = vertexPos;
    if (moveToCorner) {
        gl_Position += cornerVec;
    }
}
```

Фрагментный шейдер *Phong.frag* реализует затенение Фонга, интерполирующее нормаль к точке.

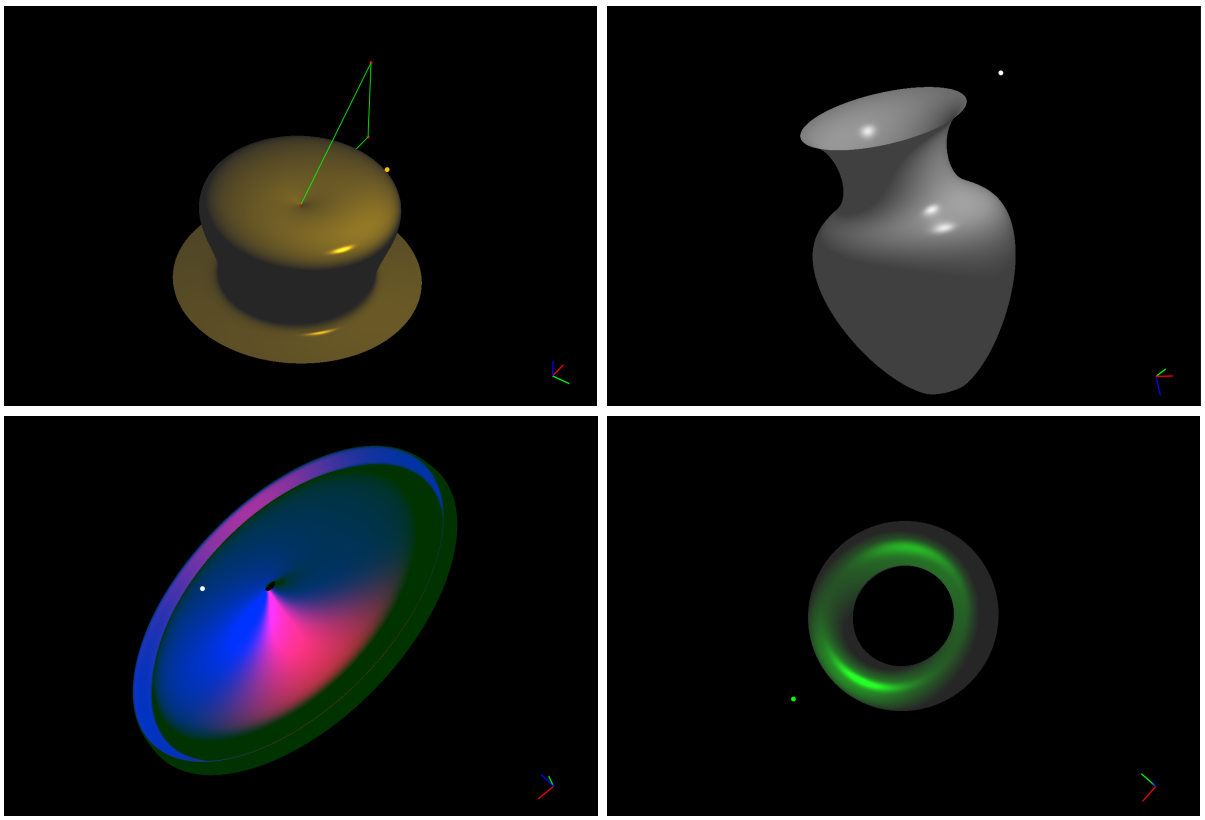
```
29 #version 150 core
30
31 in vec3 position;
32 in vec3 normal;
33 out vec4 color;
34 uniform bool useSingleColor;
35 uniform vec3 singleColor3f;
36 uniform vec3 ka3f;
37 uniform vec3 kd3f;
38 uniform vec3 ks3f;
39 uniform vec3 light3f;
40 uniform vec3 ia3f;
41 uniform vec3 il3f;
42 uniform float p;
43 uniform vec3 camera = vec3(0, 0, -1e9f);
44 uniform float k = 0.5;
45
46 void main(void) {
47     if (useSingleColor) {
48         color = vec4(singleColor3f, 1.0f);
49         return;
50     }
51     vec3 n = normal;
52     if (!gl_FrontFacing) {
53         n = -n;
54     }
55     vec3 res = singleColor3f;
56     vec3 l = light3f - position;
57     float d = length(l);
58     l = normalize(l);
59     vec3 s = normalize(camera - position);
60     vec3 r = normalize(2 * n * dot(n, l) - l);
61     float diffusal = dot(l, n);
62     float specular = dot(r, s);
63     if (diffusal < 1e-3) {
64         diffusal = 0;
65         specular = 0;
66     }
67     if (specular < 1e-3) {
68         specular = 0;
69     }
70     for (int i = 0; i < 3; ++i) {
71         res[i] = res[i] * (ia3f[i] * ka3f[i] + il3f[i] * (kd3f[i] * diffusal + ks3f[i]
               * pow(specular, p)) / (d + k));
72     }
73     color = vec4(res, 1.0f);
74 }
```

3 Демонстрация работы программы

Каркасная визуализация поверхности. В программе пользователь может задать цвет каркаса, выбрать, отрисовывать ли контрольные точки.



Освещение поверхности. Предусмотрено изменение всех парамертов материала (цвет, свойства материала воспринимать фоновое, рассеянное и зеркальное освещение, глянцевость), мощность фонового освещения и интенсивность точечного источника света.



4 Выводы

В ходе выполнения курсовой работы я познакомился с полиномиальными кривыми и поверхностями, реализовал вычисление NURBS-кривой.

Сложнее всего было правильно вычислять коэффициенты при табулирования кривой, для отображения поверхности на экран я использовал 3D-движок из ЛР с небольшими изменениями.

Список литературы

[1] Шикин Е. В., Плис А. И. *Кривые и поверхности на экране компьютера.* — Издательский дом «ДИАЛОГ-МИФИ». — 240 с. (ISBN 5-86404-080-0 ("ДИАЛОГ-МИФИ"))

[2] *B Spline curves and Beizer curves — GameDev.ru*
URL: `https://gamedev.ru/code/articles/bsplines` (дата обращения: 08.12.2021).

[3] *Кривые и поверхности NURBS*
URL: `https://studfile.net/preview/7069999/page:8/` (дата обращения: 13.12.2021).