Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №3 по курсу «Компьютерная графика»

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

Москва, 2021

Основы построения фотореалистичных изображений

Задача: Используя результаты предыдущей лабораторной работы, 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 задания: Шаровой сектор.

1 Описание

Параметризация

Параметризация шара производится по двум углам — ϕ и θ в сферической системе координат. Табулируя уравнение сферы по углам можно получить глобус, который при достаточной степени аппроксимации неотличим от шара.

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.

2 Исходный код

Файл *Figure.cs* содержит код для генерации фигуры. Для расчёта освещения в точке требуется предподсчитать нормали в вершинам и сопоставить каждой вершине все полигоны, в которые она входит.

```
public void GenVertN()
{
    List<Vector4D> polygonNormals = new List<Vector4D>();
    foreach (Polygon poly in _polygons)
    {
        polygonNormals.Add(poly.NormalVector());
        for (int i = 0; i < poly.Count; ++i)
        {
            poly._normals.Add(new Vector4D());
        }
    }
    _normals = new List<Vector4D>();
    foreach (VertexPolygonsPair item in _vertices)
    {
        Vector4D vec = new Vector4D();
        foreach (int polyId in item.Second)
        {
            vec = vec + _polygons[polyId].NormalVector();
        }
        vec = vec / item.Second.Count;
        _normals.Add(vec);
        foreach (int polyId in item.Second)
        {
            Polygon poly = _polygons[polyId];
            for (int i = 0; i < poly.Count; ++i)
            {
                if (item.First == poly[i])
                {
                    poly._normals[i] = vec;
                }
            }
        }
    }
}

private void GenVertexPolygons()
{
    for (int i = 0; i < _polygons.Count; ++i)
    {
        foreach (Vector4D vertex in _polygons[i]._data)
        {
            for (int j = 0; j < _vertices.Count; ++j)
            {
```

```
44                  if (_vertices[j].First == vertex)
45                  {
46                      _vertices[j].Second.Add(i);
47                  }
48              }
49          }
50      }
51  }
```

Geometry.cs содержит классы векторов и матрицы, методы для работы с ними. В *Misc.cs* заданы константы, используемые во всей программе.

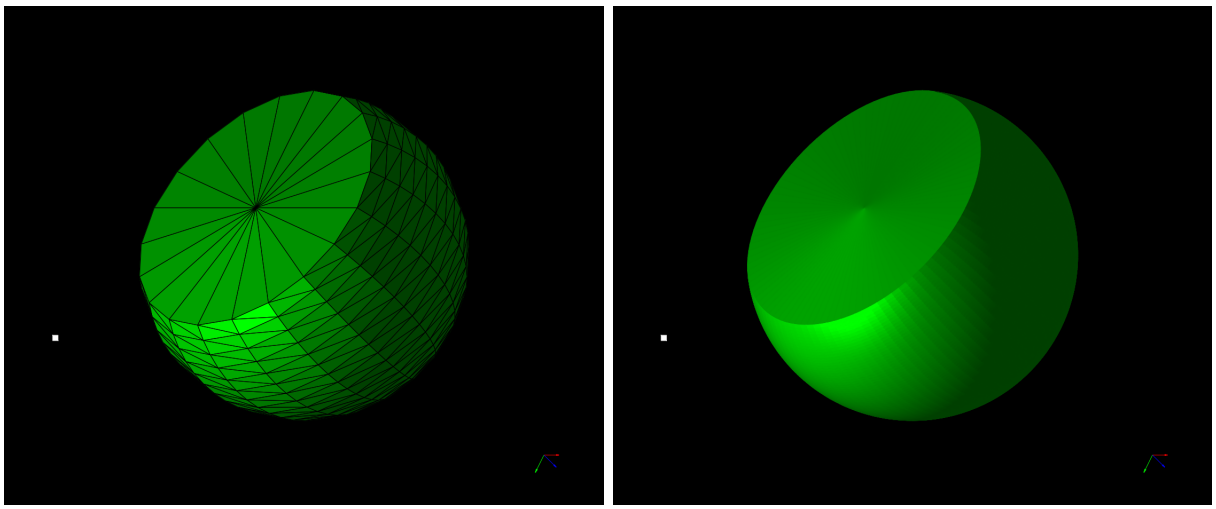
Файл *MainWindow.cs* содержит интерфейс и методы отрисовки фигуры. Функция расчёта отсвещения:

```
52  private void GenShade()
53  {
54      GetMaterialData();
55      GetLightSourceData();
56      polygonShade = new List<Misc.Colour>();
57      if (_radioButtonNoShading.Active)
58      {
59          foreach (Misc.Colour col in polygonColors)
60          {
61              polygonShade.Add(col);
62          }
63          return;
64      }
65      if (_radioButtonFlat.Active)
66      {
67          for (int i = 0; i < polygonColors.Count; ++i)
68          {
69              Vector4D N = fig._polygons[i].NormalVector();
70              Vector4D curPolyCenter = fig._polygons[i].GetCenter();
71              polygonShade.Add(GenShadePoint(polygonColors[i], curPolyCenter, N));
72          }
73      }
74  }
75
76  private Vector4D GetVectorL(Vector4D p)
77  {
78      return lightSource - p;
79  }
80
81  private Vector4D GetVectorR(Vector4D p, Vector4D n, Vector4D l)
82  {
83      return 2 * n * Vector4D.Dot(n, l) - l;
84  }
85
86  private Vector4D GetVectorS(Vector4D p)
```

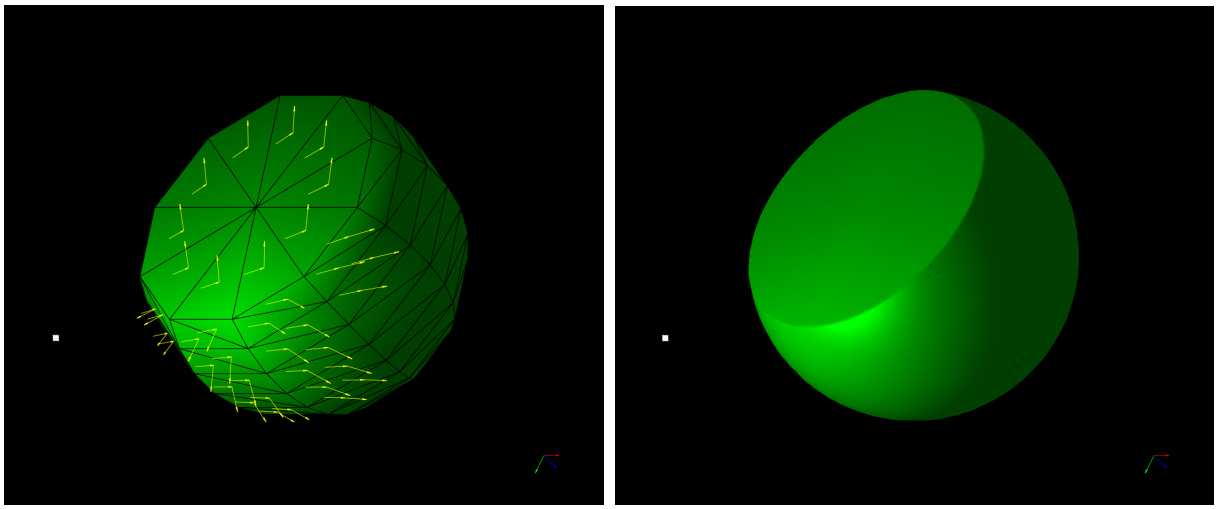
```
87  {
88      return INF_VEC - p;
89  }
90
91  private Misc.Colour GenShadePoint(Misc.Colour col, Vector4D point, Vector4D N)
92  {
93      N.Normalize();
94      Vector4D L = GetVectorL(point);
95      double dist = 0.001 * L.Len();
96      L.Normalize();
97      Vector4D R = GetVectorR(point, N, L);
98      R.Normalize();
99      Vector4D S = GetVectorS(point);
100     S.Normalize();
101     Misc.Colour ambientIntensity = ka * ia;
102     Misc.Colour diffuseIntensity = kd * il * Vector4D.Dot(L, N);
103     Misc.Colour specularIntensity = ks * il * Math.Pow(Vector4D.Dot(R, S),
            SHADING_COEF_P);
104     if (Vector4D.Dot(L, N) < Misc.EPS)
105     {
106         diffuseIntensity = new Misc.Colour();
107         specularIntensity = new Misc.Colour();
108     }
109     if (Vector4D.Dot(R, S) < Misc.EPS)
110     {
111         specularIntensity = new Misc.Colour();
112     }
113     ambientIntensity.Clamp();
114     diffuseIntensity.Clamp();
115     specularIntensity.Clamp();
116     Misc.Colour res = col * (ambientIntensity + (diffuseIntensity + specularIntensity)
            / (dist + SHADING_COEF_K));
117     res.Clamp();
118     return res;
119 }
```

3 Демонстрация работы программы

Плоское затенение. Даже при большой степени аппроксимации видны полосы.



Затенение Гуро. Предусмотрена возможность отрисовки падающего и отраженно луча для полигона.



Рассеянная и зеркальная состовляющая освещения. Слева только рассенянная, справа обе состовляющие.