COSC6372 HW4 – Shading

Jiechang Guo UHID: 2084258

April 2, 2023

1 Problem

The problem for this assignment is to render the teapot with lights. To do so, Gouraud shading and Phong shading will be implemented.

2 Method

2.1 Phong Lighting Model

Calculating the light effect can bring us more reality of the rendered image. A simple lighting model is called the Phong lighting model which models the physics of light. The Phong lighting model contains the following three parts[1]:

- ambient lighting: simulate when the scene is dark which always gives the object some color. Because, in the real world, it's hard to be pure dark. There will be indirect light in the scene.
- diffuse lighting: simulate the amount of light reflected on an object. The amount of light reflected is determined by the normal of the surface.
- specular lighting: simulate the bright spot effect on shiny objects.

The picture 1 shows the three components of the Phong lighting model.

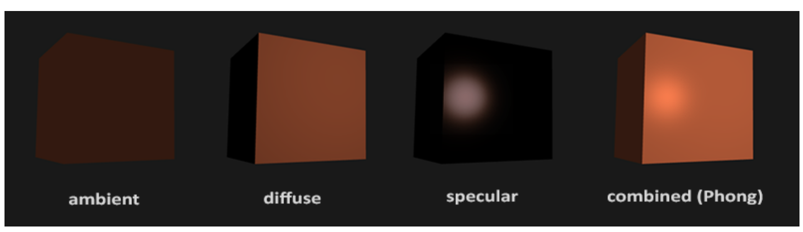

Figure 1: Phong lighting model effect[1]

To calculate the final color for an object, we will need the material of the object including ambient coefficient, diffuse coefficient, specular coefficient, and power spec and defined as four real numbers: kA, kD, kS, and s respectively. The equation for calculating the light with object color are followed:

$$
\begin{aligned}
&ambient = kA * objectColor * lightColor\\
&diffuse = (max(dot(normal, lightDir), 0) * kD * objectColor * lightColor\\
&viewDir = normalize(viewPos - fragPos)\\
&reflectDir = -lightDir - 2.0 * dot(normal, -lightDir) * normal\\
&specular = (pow(dot(viewDir, reflectDir), 0), s) * objectColor * lightColor
\end{aligned}
$$

The light shading can be calculated in either world space or view space, for convenience, we will calculate it in view space. Also, there are two shading modes, Gouraud shading which calculates the light for every vertex, and Phong shading which calculates the light for every fragment. In the following section, we will go into detail about each of the methods.

2.2 Calculate Light in View Space

In this assignment, we will calculate the light in the view space, because in the view space, the eye position is always at the origin. Thus when we calculate the viewDir in the above equation, we only need the fragPos. We need to transform the vertex, normal, and light direction to view space using the transform matrix.

Transform the vertex to view space This process is similar to converting the vertex from object space to screen space which we did in the last assignment except we do not apply projection and viewport transform.

$$fragPos = viewMat * modelMat * vertexPos$$

Transform the vector to view space For vector, we mean the normal vector and the light direction vector. Unlike transforming a vertex as a position we do not use homogeneous coordinates.

$$normalInView = mat3(transpose(inverse(view * model))) * normInObject$$
$$lightDirInView = mat3(transpose(inverse(view * model))) * lightDirInObject$$

2.3 Gouraud Shading

Gouraud shading is implemented the Phong lighting model in the vertex shader which calculates the vertex color for every vertex. And the vertex color will be used for interpolating the fragment color. This method is fast because the number of vertices is less than the number of fragments. However, it will cause artifacts.

2.4 Phong Shading

On the other hand, the Phong shading is implemented in the Phong lighting model in the fragment shader which calculates the fragment color. To do so, except to interpolate the vertex position, and vertex color, the normal for each fragment will also be interpolated.

3 Implementation

Transform on normal and light direction From object space to view space. The transform of normal and light direction are similar, thus we only list the code for transforming the normal.

```
GzVector Gz::fromObjectToView(GzVector &n)
{
  //Normal = mat3(transpose(inverse(view * model))) * aNormal;
  GzMatrix transMat3 = Zeros(3);
  for(int i = 0; i < 3; i++)
  {
    for(int j = 0; j < 3; j++)
    {
      transMat3[i][j] = transMatrix[i][j];
    }
  }
  GzMatrix res = Zeros(3);
  res = transMat3.inverse3x3();
  res = res.transpose();
  GzMatrix normMat;
  normMat.resize(3,1);
  normMat[0][0] = n[0];
  normMat[1][0] = n[1];
  normMat[2][0] = n[2];
  GzMatrix resMat = res * normMat;

  GzVector nRes;
  nRes[0] = resMat[0][0];
  nRes[1] = resMat[1][0];
  nRes[2] = resMat[2][0];
    return nRes;
}
```

Transform on vertex From object space to view space. We use the homogeneous coordinates to transform the vertex to view space.

```
GzVertex Gz::fromObjectToView(GzVertex &v)
{
    GzMatrix res;
  res.fromVertex(v);
  res=transMatrix*res;
  GzVertex vRes=res.toVertex();
  return vRes;
}
```

Direct light color calculating based on Phong model We accumulated the light colors from multiple light sources.

```
void GzFrameBuffer::loadLightTrans(GzMatrix mat)
{
  //clear previous transform on light directions
```

```
  lights.clear();
  lights = orignLights;

  GzMatrix transMat3 = Zeros(3);
  for(int i = 0; i < 3; i++)
  {
    for(int j = 0; j < 3; j++)
    {
      transMat3[i][j] = mat[i][j];
    }
  }
  //transform light direction to view space
  for(int i = 0; i < lights.size(); i++)
  {
    GzVector dir = lights[i].first;
    dir.normalize();
    GzMatrix res = Zeros(3);
    res = transMat3.inverse3x3();
    res = res.transpose();
    GzMatrix dirMat;
    dirMat.resize(3,1);
    dirMat[0][0] = dir[0];
    dirMat[1][0] = dir[1];
    dirMat[2][0] = dir[2];
    GzMatrix resMat = res * dirMat;

    GzVector nRes;
    nRes[0] = resMat[0][0];
    nRes[1] = resMat[1][0];
    nRes[2] = resMat[2][0];

    lights[i].first = nRes;
  }
}

void GzFrameBuffer::addLight(const GzVector& v, const GzColor& c) {
  orignLights.push_back(pair<GzVector,GzColor>(v,c));
}

GzVector GzFrameBuffer::calDirLight(GzVector p, GzVector n, GzVector dir, GzVector ltc)
{
  //ambient
  GzVector ambient;
  ambient = kA*ltc;

  GzVector zeroV;
  // diffuse
  GzVector norm = n;
  norm.normalize();
  GzVector lightDir = zeroV - dir;
  lightDir.normalize();
  float diff = max(dot(norm, lightDir), 0.0);
  GzVector diffuse = kD * diff * ltc;

  // specular
  GzVector viewDir;//we calculate the light in view space, so the viewPos-fragPos is equal to -
    fragPos
  viewDir = zeroV - p;
  viewDir.normalize();

  GzVector reflectDir = reflect(zeroV-lightDir, norm);
  reflectDir.normalize();

  float spec = pow(max(dot(viewDir, reflectDir), 0.0), s);
  GzVector specular = kS * spec * ltc;

  return ambient + diffuse + specular;
}

GzVector GzFrameBuffer::calAllLight(GzVector p, GzVector n)
{
  GzVector lightColor;

  //for each light source
  for(int j = 0; j < lights.size(); j++)
```

```
  {
    GzVector ltc;
    ltc[0] = lights[j].second[0];
    ltc[1] = lights[j].second[1];
    ltc[2] = lights[j].second[2];

    lightColor = lightColor + calDirLight(p,n,lights[j].first ,ltc);
  }
    return lightColor;
}
```

Gouraud Shading and Phong Shading Calculate light on each vertex or each fragment. For the detail of the rasterization and interpolating of normal please see the source code.

```
void GzFrameBuffer::drawTriangle(vector<GzVertex> &v, vector<GzVector>& p, vector<GzVector> &n,
    vector<GzColor> &c, GzFunctional status)
{
  //Do light color shading on vertex
  if(curShadeModel == GZ_GOURAUD && n.size() == v.size() && (status&GZ_LIGHTING))
  {
    //for each vertex, calculate the light color according to the normal
    for(int i = 0; i < n.size(); i++)
    {
      GzVector lightColor = calAllLight(p[i], n[i]);

      c[i][0] = c[i][0] * lightColor[0];
      c[i][1] = c[i][1] * lightColor[1];
      c[i][2] = c[i][2] * lightColor[2];
    }

    drawTriangle(v,c,status);
  }//Do light color shading on fragment
  else if(curShadeModel == GZ_PHONG && n.size() == v.size() && (status&GZ_LIGHTING))
  {
    GzInt yMin, yMax;
    GzReal xMin, xMax, zMin, zMax;
    GzColor cMin, cMax;
    GzVector nMin, nMax;
    GzVector pMin, pMax;

    v.push_back(v[0]);
    c.push_back(c[0]);
    n.push_back(n[0]);
    p.push_back(p[0]);

    yMin=INT_MAX;
    yMax=-INT_MAX;

    for (GzInt i=0; i<3; i++) {
      yMin=min((GzInt)floor(v[i][Y]), yMin);
      yMax=max((GzInt)floor(v[i][Y]-1e-3), yMax);
    }

    for (GzInt y=yMin; y<=yMax; y++) {
      xMin=INT_MAX;
      xMax=-INT_MAX;
      for (GzInt i=0; i<3; i++) {
        if ((GzInt)floor(v[i][Y])==y) {
          if (v[i][X]<xMin) {
            xMin=v[i][X];
            zMin=v[i][Z];
            nMin=n[i];
            cMin=c[i];
            pMin=p[i];
          }
          if (v[i][X]>xMax) {
            xMax=v[i][X];
            zMax=v[i][Z];
            nMax=n[i];
            cMax=c[i];
            pMax=p[i];
          }
        }
        if ((y-v[i][Y])*(y-v[i+1][Y])<0) {
```

```
          GzReal x;
          realInterpolate(v[i][Y], v[i][X], v[i+1][Y], v[i+1][X], y, x);
          if (x<xMin) {
            xMin=x;
            realInterpolate(v[i][Y], v[i][Z], v[i+1][Y], v[i+1][Z], y, zMin);
            colorInterpolate(v[i][Y], c[i], v[i+1][Y], c[i+1], y, cMin);
            vectorInterpolate(v[i][Y], n[i], v[i+1][Y], n[i+1], y, nMin);
            vectorInterpolate(v[i][Y], p[i], v[i+1][Y], p[i+1], y, pMin);
          }
          if (x>xMax) {
            xMax=x;
            realInterpolate(v[i][Y], v[i][Z], v[i+1][Y], v[i+1][Z], y, zMax);
            colorInterpolate(v[i][Y], c[i], v[i+1][Y], c[i+1], y, cMax);
            vectorInterpolate(v[i][Y], n[i], v[i+1][Y], n[i+1], y, nMax);
            vectorInterpolate(v[i][Y], p[i], v[i+1][Y], p[i+1], y, pMax);
          }
        }
      }
      drawRasLine(y, xMin, zMin, nMin, pMin, cMin, xMax-1e-3, zMax, nMax, pMax, cMax, status);
    }

  }

}
```

4 Result

In this section, the final result of the rendered teapots will be illustrated.

The final result of Gouraud shading is shown in figure 2, and the final result of Phong shading is shown in figure 3. We can tell that the result of the Phong shading is smoother.

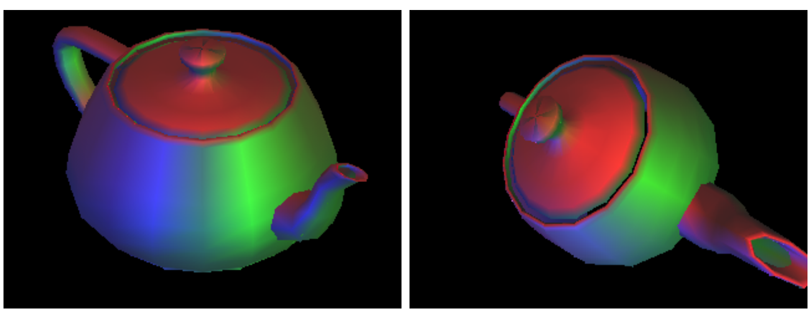

Figure 2: GouraudTeaPot1, GouraudTeaPot2

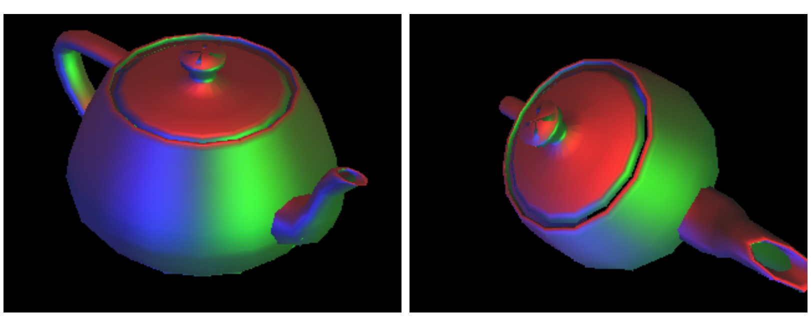

Figure 3: PhongTeaPot1, PhongTeaPot2

References

[1] Joey de Vries. *Basic Lighting*. https://learnopengl.com/Lighting/Basic-Lighting.