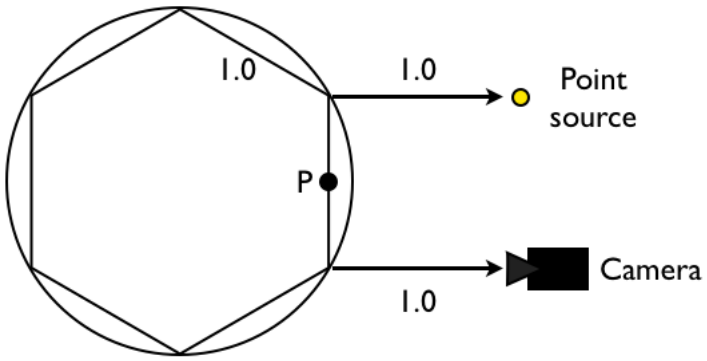
Week 6 Tutorial Solutions

Question 1:

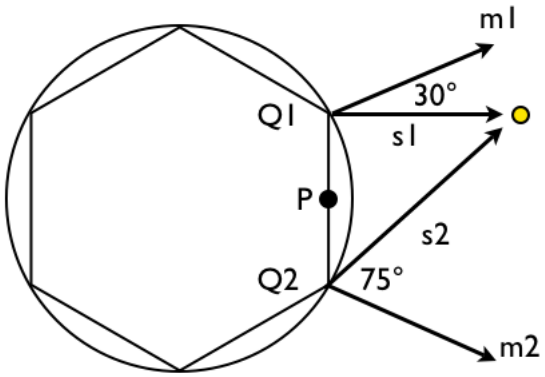
Consider lighting a quad on the surface of a cylindrical mesh with radius 1. You are approximating the cylinder with a six-sided prism, as shown below. The position and distance to the light source and camera are labelled.



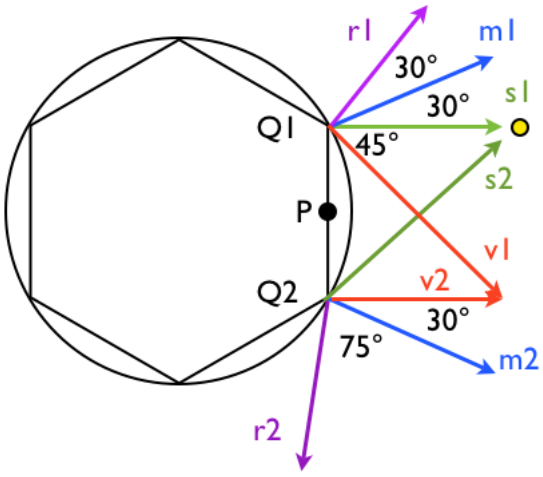
Compute the **diffuse** and **specular** intensity at the point P on the midpoint of the edge, using:

1. Flat shading
2. Gouraud shading
3. Phong shading
4. The actual surface of the cylinder.

Assume the source intensity and reflection coefficients are all 1. The Phong exponent is 1. Assume for the specular calculations you are using actual the reflection vector from the Phong lighting model and not the halfway vector or any approximations.



$$
\begin{aligned}
I_d(Q_1) &= \hat{s_1} \cdot \hat{m_1} \\
&= \cos(30^\circ) \\
&\approx 0.866 \\
I_d(Q_1) &= \hat{s_2} \cdot \hat{m_2} \\
&= \cos(75^\circ) \\
&\approx 0.259
\end{aligned}
$$



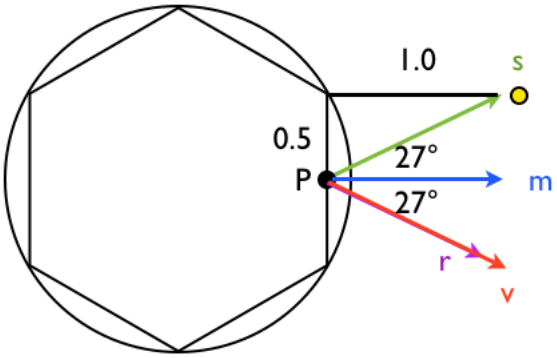
$$
\begin{aligned}
I_s(Q_1) &= \max(0, (\hat{r_1} \cdot \hat{v_1})) \\
&= \max(0, \cos(105^\circ)) \\
&= 0 \\
I_s(Q_2) &= \max(0, \hat{r_2} \cdot \hat{v_2}) \\
&= \max(0, \cos(105^\circ)) \\
&= 0
\end{aligned}
$$

Flat :

$$\begin{aligned} I_d(P) &= I_d(Q_1) \text{ or } I_d(Q_2) \\ &\approx 0.866 \text{ or } 0.259 \\ I_s(P) &= I_s(Q_1) \\ &= 0 \end{aligned}$$

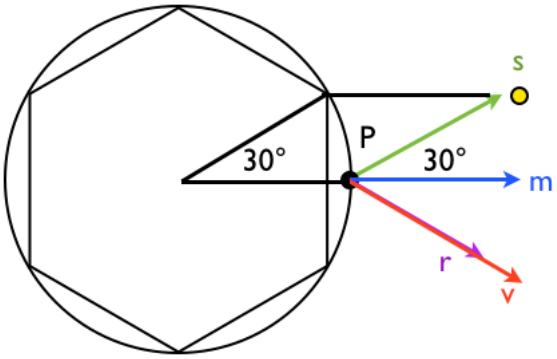
Gouraud :

$$\begin{aligned} I_d(P) &= \tfrac{1}{2} I_d(Q_1) + \tfrac{1}{2} I_d(Q_2) \\ &\approx 0.563 \\ I_s(P) &= \tfrac{1}{2} I_s(Q_1) + \tfrac{1}{2} I_s(Q_2) \\ &= 0 \end{aligned}$$



Phong :

$$\begin{aligned} \tan(\theta) &= 0.5 \\ \theta &\approx 27^\circ \\ I_d(P) &= \hat{s} \cdot \hat{m} \\ &= \cos(\theta) \\ &\approx 0.894 \\ I_s(P) &= \hat{r} \cdot \hat{v} \\ &= \cos(0^\circ) \\ &= 1 \end{aligned}$$



Actual :

$$\begin{aligned} I_d(P) &= \hat{s} \cdot \hat{m} \\ &= \cos(30^\circ) \\ &\approx 0.866 \\ I_s(P) &= \hat{r} \cdot \hat{v} \\ &= \cos(0^\circ) \\ &= 1 \end{aligned}$$

Question 2: 3d Modeling

a. Look at the TriangleMesh class in UNSWgraph. How does it generate normals? What do we have to do if we have a vertex on mesh that is shared amongst many faces but we don't want that vertex to appear smooth under lighting?
The TriangleMesh class will generate vertex normals or face normals depending on what constructors are called. If you supply indices then the mesh will be drawn with indexed drawing and vertex normals will be calculated by averaging the face normals of all contributing faces. If no indices are supplied, then face normals will be calculated by the cross product method.

b. Write code to generate a cylinder as one or more triangle meshes. The front face of the cylinder should be a circle of radius 1 at (0,0,-1), lying on the z=-1 plane with a back face at z=-3.

```
private void makeCylinder(GL3 gl) {
    //An front circle
    List frontcircle = new ArrayList<>();
    List frontIndices = new ArrayList<>();
    float angleIncrement = (float) (2*Math.PI/NUM_SLICES);
    for (int i = 0; i < NUM_SLICES; i++) {
        float angle = i * angleIncrement;
        float x = (float) Math.cos(angle);
        float y = (float) Math.sin(angle);
        frontcircle.add(new Point3D(x, y, -1));

        //Generate indices that effectively create a triangle fan.
        frontIndices.add(i);
        if (i > 2) {
            frontIndices.add(0);
            frontIndices.add(i-1);
```

```
            }
        }

        TriangleMesh front = new TriangleMesh(frontcircle, frontIndices, true);

        // The back circle
        List backCircle = new ArrayList<>();
        for (Point3D p : frontcircle)
            backCircle.add(p.translate(0,0,-2));
        List backIndices = new ArrayList<>(frontIndices);
        Collections.reverse(backIndices);

        TriangleMesh back = new TriangleMesh(backCircle, backIndices, true);

        // We want the sides to be smooth, so make sure the vertices are shared.
        List sides = new ArrayList<>();
        List sideIndices = new ArrayList<>();
        for (int i = 0; i < NUM_SLICES; i++) {
            //The corners of the quad we will draw as triangles
            Point3D f = frontcircle.get(i);
            Point3D b = backCircle.get(i);

            sides.add(f);
            sides.add(b);

            //Indices
            int j = (i + 1) % NUM_SLICES;
            sideIndices.add(2*i);
            sideIndices.add(2*i + 1);
            sideIndices.add(2*j + 1);

            sideIndices.add(2*i);
            sideIndices.add(2*j + 1);
            sideIndices.add(2*j);
        }

        TriangleMesh sidesMesh = new TriangleMesh(sides, sideIndices, true);

        front.init(gl);
        back.init(gl);
        sidesMesh.init(gl);
        meshes.add(front);
        meshes.add(back);
        meshes.add(sidesMesh);
    }
```

c. Suppose we have a profile of a curve that is a quarter of a circle as defined by the following function

```
C(u) = (radius*cos(u),radius*sin(u)) For u [0..90]
```

What would the surface of revolution around the y-axis be defined as in terms of u and the angle v for v[0..360]

```
P(u,v) = ?
```

```
First, split the function that defines the original curve into an X and Y function.
X(u) = radius*cos(u)
Y(u) = radius*sin(u)
We're rotating the curve around the y-axis, which means the y-component of the function that defines the surface is the same as the Y function.
P(u,v) = ( X(u)cos(v),Y(u), X(u)*sin(v))
```

Write a piece of code to generate a list of vertices and normals for the object.

```
private void createVertices() {
    //Angle used in semi-circle profile. Semi-circle so PI/2
    float angleSize = (float) Math.PI/2.0/(NUM_STACKS);
    int i;
    int j = 0;
    int k=0;
    vertices = new ArrayList();
    normals = new ArrayList();
    for(i = 0; i < NUM_STACKS ; i++){
        float currentAngle = angleSize*i;
        float x = radius * Math.sin(currentAngle);
        float y = radius * Math.cos(currentAngle);
        for(j = 0; j <= NUM_SLICES ;j++){
            float x2 = (float) (x * Math.cos(Math.PI * 2.0 * j/(NUM_SLICES-1) ));
            float z  = (float) (x * Math.sin(Math.PI * 2.0 * j/(NUM_SLICES-1) ));

            Point3D p = new Point3D(x2, y, z);

            vertices.add(p);

            //In this situation the normals are just the x,y,z co-ordinates.

            normals.add(new Vector3(x2,y,z));
        }
    }
}
```

d. Calculate the frenet frame for the helix spine at t = PI for b = 1

```
C(t) = (cos(t), sin(t), bt)
```

```
First find the origin
C(PI) = (-1,0,PI)

Then find the tangent. One approach is to use derivative with respect to t:
C'(t) = (-sin(t), cos(t), b)
At t = PI this is (0,-1,1) or, once normalised, (0,-1/sqrt(2),1/sqrt(2));
```

```
Or you could evaluate C(t) with t = PI-Epsilon and t = PI+Epsilon, for some small epsilon and find the difference.

For example, tangent = C(PI+0.1) - C(PI-0.1) will yield the same result (approximately).

However we find the tangent, once we have it we can assign it to k.
k = (0,-1/sqrt(2),1/sqrt(2))

For the i-axis, we use the perpendicular vector corresponding to (-k_y,k_x,0).
This gives us (1/sqrt(2),0,0) which we need to normalise

i = (1,0,0)

To calculate the j-axis we need something perpendicular to both k and i. We can find that by taking the cross product.
j = k x i
  = (0,-1/sqrt(2),1/sqrt(2)) x (1,0 0);

  = (0,1/sqrt(2),-1/sqrt(2))

                i   j          k           o
Frenet frame = [ 1   0          0           -1  ]
               [ 0   1/sqrt(2) -1/sqrt(2)   0   ]
               [ 0  -1/sqrt(2)  1/sqrt(2)   PI  ]
               [ 0   0          0            1  ]
```

Suppose you had the triangle with points P0, P1 and P2 and you wanted to extrude it along the given helical spine. How would you generate the points at the cross section where t = PI on the spine? **You would simply multiply each point by the Frenet Frame Question 3:**

The model of lighting we are using (diffuse + specular + ambient) is very limited. Discuss what kinds of lighting effects are being neglected (there are many). How important are they?

We will discuss some extensions in later lectures.

These algorithms ignore:

- **all incoming light which does not come directly from a light source, I.e. light reflected off one surface onto another (this is approximated by the ambient value)**
- **any cases where one object casts shadows onto another**
- **the size and shape of light sources**
- **transparent materials and translucent materials with refractive properties**
- **(NOT IN EXAM) sub-surface scattering which is an important aspect for rendering skin realistically**

and many more effects.
We will look at ways to create shadows and reflections of the environment by using textures, however solving these problems properly requires more computationally expensive techniques such as radiosity or ray-tracing which we will look at later in the course. .

With any time remaining, catch up on questions you may have missed in previous weeks