Manvi Goel
2019472

Computer Graphics
Assignment 1

Question 1

Changes in Code

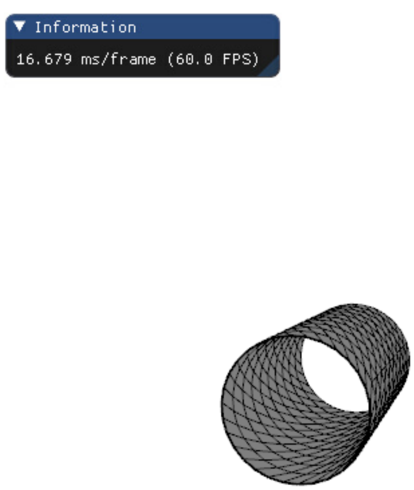
Note:- All changes are done in the main.py file. The changes are wrapped by *"Start"* and "*End*" comments to denote changes. Search "Start" for the changes.

1. Change the number of triangles drawn in *glDrawArrays.* Increased the number of triangles to support the newly drawn surface.
2. Create a function to supply the coordinate points of the surface using its parametric equation (hardcoded) and parameters.
3. Added a nested loop in *createParametricObject.* Specify the number of horizontal lines and vertical lines into which the surface is divided. Using the number of lines, divide the range of parameters and create a constant step size to create rectangles. Divide the rectangles into two triangles.

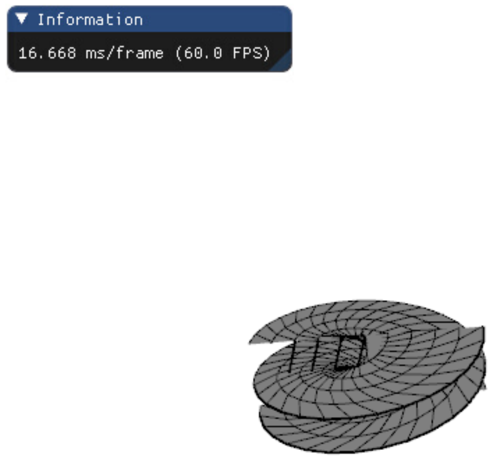
Output

Note:- The code has the equation for cylinder hard coded, but it was changed to create multiple surfaces given below.

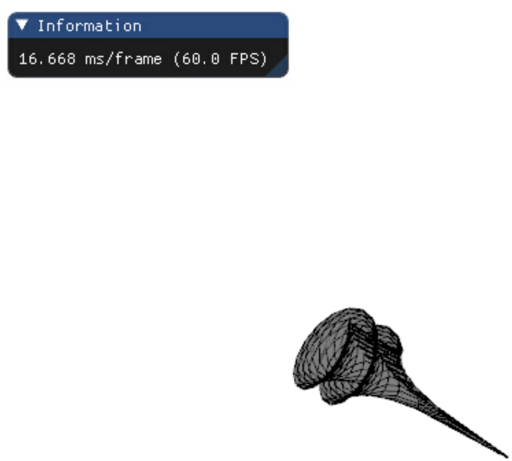
1. Cylinder. (Code in main.cpp)
   Equation. $x = R*\cos(u)$; $y = R*\sin(u)$; $z = v$ in (u, v)
   Parameter Range: $0 <= u <= 2\pi$; $0 <= v <= 20$.



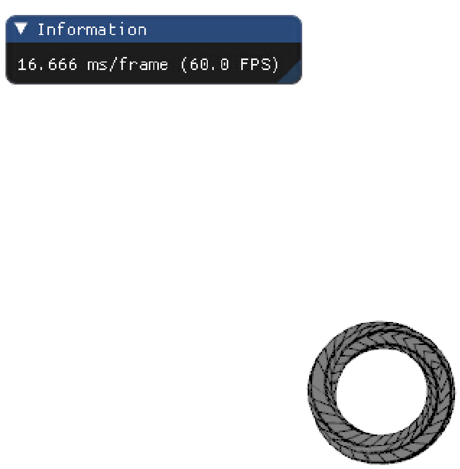
2. Helicoid
   Equation. $x = v*\cos(u)$; $y = v*\sin(u)$; $z = u$;
   Parameter Range: $-10 <= v <= 10$; $-\pi <= u <= \pi$.



3. Deni’s Surface
   Parametric Equation: $(5\cos(u)\sin(v), 5\sin(u)\cos(v), 5(\cos(v) + \ln(\tan(v/2))) + 0.5u;$
   Parameter Range: $0 <= u <= 4\pi i$; $0.01 <= v <= 1$.



4. Klien Bottle

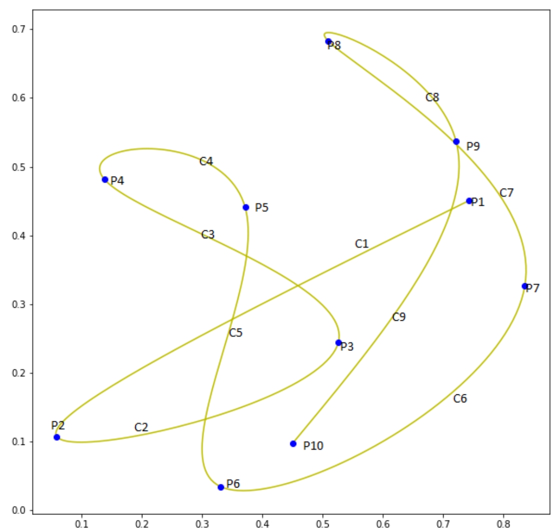


Question 2

Algorithm and Explanation

We aim to interpolate n given points using cubic Bezier curves.

To connect 2 consecutive points P_i and P_{i+1} we will create a cubic Bezier curve with P_i and P_{i+1} as the initial and final control points, respectively. Apart from the two points, we need 2 more points to create a cubic Bezier curve that requires 4 control points.



Like in the given image with points P_1 to P_{10} has 9 cubic Bezier curves, including C_1 to C_9. Each curve C_i is a cubic Bezier curve starting from P_i and ending at P_{i+1}. For example, C_1 starts from P_1 and ends at P_2.

We know to create a cubic Bezier curve, we need 4 control points with the equation

$$P(t) = K0 * (1-t)^3 + K1 * 3t(1-t)^2 + K2 * 3t^2(1-t) + K3 * t^3$$

where K0, K1, K2, and K3 are the 4 control points. Here, K0 = P_i and K3 = P_{i+1}.

We need to select points K1 and K2 such the C_i is continuous, and the curve between C_i and C_{i+1} is C^1.

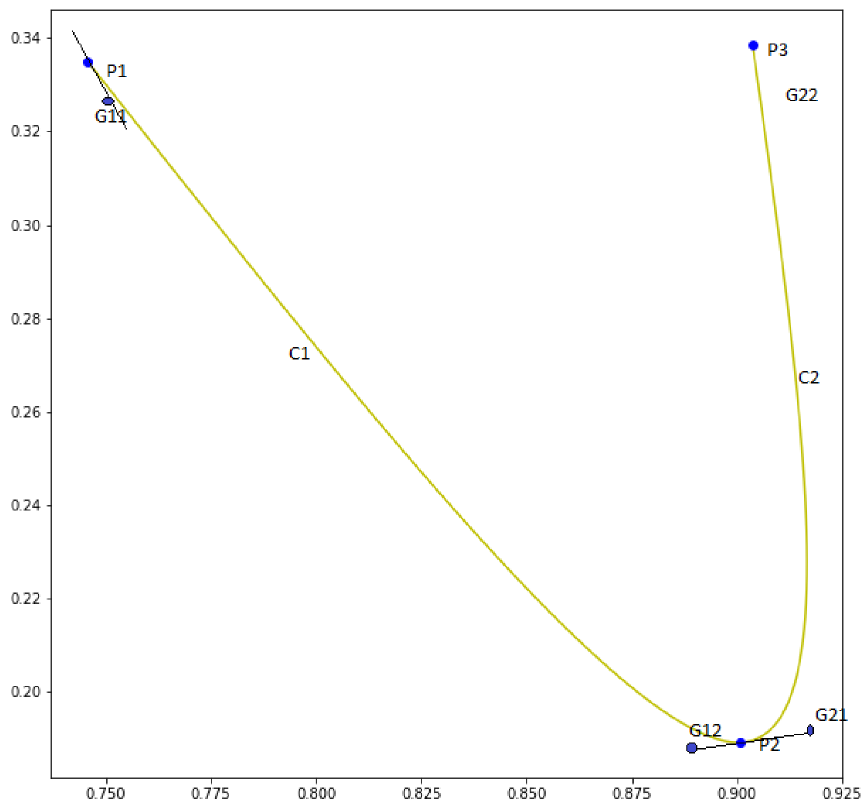
To ensure that C_i and C_{i+1} are continuous and differentiable at P_{i+1}, we need to ensure that the 3rd control point for C_i and 2nd control point for C_{i+1} and P_{i+1} are collianer.

In other words,

If we create 2 ghost control points for each curve C_i namely, G_{i1} and G_{i2} then,

G_{i2}, P_{i+1}, and $G_{(i+1)1}$ should be collinear.

This will ensure that C_i and C_{i+1} are differentiable as $G_{i2}P_{i+1}$ is a tangent to C_i (property of Bezier curve) and $P_{i+1}G_{(i+1)1}$ is a tangent to C_{i+1} (property of Bezier curve) and if the three points are collieaner then the line $G_{i2}G_{(i+1)1}$ is a tangent to both C_i and C_{i+1} at P_{i+1}. Hence, the final curve will be differentiable at P_{i+1}.



Thus, we can create G_{11} and G_{12} randomly, G_{21} will be found using G_{12} and P_2, and G_{22} can be chosen again randomly.
We can continue with the process until we create a complete curve.

<u>Pseudocode</u>

Function PieceWiseCubicBezierCurve(Points):
    For P_k where k = 1 to n-1
        If k is 1
            Initialize G_{k1} randomly. // RandomInitialization(P_k)
            Initialize G_{k2} randomly. // RandomInitialization(P_{k+1})

        Else
            Initialize G_{k1} as a point on $G_{(k-1)2}P_k$. // CoPointInitialization($G_{(k-1)2}$, P_k)
            Initialize G_{k2} randomly. // RandomInitialization(P_{k+1})

        Calculate C_k = CubicBezierCurve(P_k, G_{k1}, G_{k2}, P_{k+1})

    Return {C_k where k = 1 to n-1}

Function RandomInitialization(P_k):
    Initialize ε {Small value to keep G near P}
    $G_k = P_k$ +(/-) ε
    Return G_k

Function CoPointInitialization($G_{(k-1)2}$, P_k) {Using vector equations}
    Initialize ε
    $v = P_k - G_{(k-1)2}$
    $u = v/ ||v||$
    $G_{k1} = P_k + \varepsilon * u$
    Return G_{k1}

Function CubicBezierCurve(P_k, G_{k1}, G_{k2}, P_{k+1})
    $C_k = P_k * (1-t)^3 + G_{k1} * t * (1-t)^2 + G_{k2} * t^2 * (1-t) + P_{k+1} * t^3$
    Return C_k