```
clc; clear;
warning('off');

cube = { [0,0,0,0; 0,0,1,1; 0,1,1,0],  [1,1,1,1; 0,1,1,0; 0,0,1,1], ...
         [0,1,1,0; 0,0,0,0; 0,0,1,1],  [0,0,1,1; 1,1,1,1; 0,1,1,0], ...
         [0,0,1,1; 0,1,1,0; 0,0,0,0],  [0,1,1,0; 0,0,1,1; 1,1,1,1] };

box  = { [0,0,0,0; 0,0,5,5; 0,3,3,0],  [8,8,8,8; 0,5,5,0; 0,0,3,3], ...
         [0,8,8,0; 0,0,0,0; 0,0,3,3],  [0,0,8,8; 5,5,5,5; 0,3,3,0], ...
         [0,0,8,8; 0,5,5,0; 0,0,0,0],  [0,8,8,0; 0,0,5,5; 3,3,3,3]};

octahedron = { [1,0,0; 0,1,0; 0,0,1], [0,1,0; -1,0,0; 0,0,1], ...
               [-1,0,0; 0,-1,0; 0,0,1], [0,-1,0; 1,0,0; 0,0,1], ...
               [0,1,0; 1,0,0; 0,0,-1], [1,0,0; 0,-1,0; 0,0,-1], ...
               [0,-1,0; -1,0,0; 0,0,-1], [-1,0,0; 0,1,0; 0,0,-1] };

pyramid = { [1,0,0; 0,1,0; 0,0,1], [0,1,0; -1,0,0; 0,0,1], ...
            [-1,0,0; 0,-1,0; 0,0,1], [0,-1,0; 1,0,0; 0,0,1], ...
            [1,0,-1,0; 0,-1,0,1; 0,0,0,0]};

frustum = { [2,0,0,1; 0,2,1,0; 0,0,1,1], [0,-2,-1,0; 2,0,0,1; 0,0,1,1], ...
            [-2,0,0,-1; 0,-2,-1,0; 0,0,1,1], [0,2,1,0; -2,0,0,-1; 0,0,1,1] ...
            [2,0,-2,0; 0,-2,-0,2; 0,0,0,0], [1,0,-1,0; 0,1,0,-1; 1,1,1,1]};


% Phase 1.

R = DrawRotatedPolyhedronPhase1(EulerRotation(pi/4, pi/4, 0), cube);

% Phase 2.

R = DrawRotatedPolyhedronPhase2(R);

% Phase 3.

DrawRotatedPolyhedronPhase3(R);

% Phase 4.
% Before Phase3 we obtain o from Phase2's result.
% Return C where each col (3*1) represent 1 center's coordinate (x, y ,z)

C4 = DrawRotatedPolyhedronPhase4(R);

% Phase 5.
% Modify Phase 4 to calculate the radius as well.
% Return C where each col (4*1) represent 1 center's coordinate (x, y, z)
% The last is the radius

C5 = DrawRotatedPolyhedronPhase5(R);

% Phase 6.
% Modity Phase 5 to store 2 vectors.
% Return C where each col (4*1) represent 1 center's coordinate (x, y, z)
% The 4th is the radius
% The 5 ~ 7th is vector 1
```

```
% The 8 ~ 9th is vector 2

C6 = DrawRotatedPolyhedronPhase6(R);

% Phase 7.
% Take the result from Phase6 to generate the points of the circle
% We use N = 99 (which means use 100 points to draw a circle)

G = DrawRotatedPolyhedronPhase7(C6);

% Phase 8.

DrawRotatedPolyhedronPhase8(R, G, 2);

% Final Phase
% We can now organize all subfunction to generate DrawRotatedPolyhedron
% Let us draw a frustum!

DrawRotatedPolyhedron(EulerRotation(0, pi, 0), frustum);




function R=EulerRotation(A,B,C)
   RA = [ cos(A), -sin(A), 0 ; sin(A), cos(A), 0; 0, 0, 1];
   RB = [ 1, 0, 0; 0, cos(B), sin(B); 0, -sin(B), cos(B)];
   RC = [ cos(C), -sin(C), 0 ; sin(C), cos(C), 0; 0, 0, 1];
   R = RC * RB * RA;
end

function R = DrawRotatedPolyhedronPhase1(M,P)
    R = P;
    n = size(R, 2);

    for i = 1:n
        R{1, i} = M * R{1, i};
    end
end

function R = DrawRotatedPolyhedronPhase2(P)
    R = P;
    n = size(R, 2);

    for i = 1:n
        face = R{1, i};

        a = face(:, 1);
        b = face(:, 2);
        c = face(:, 3);

        on = cross(c - b, a - b);

        if on(3, 1) > 0
            R{1, i} = [];
        end
    end
end
```

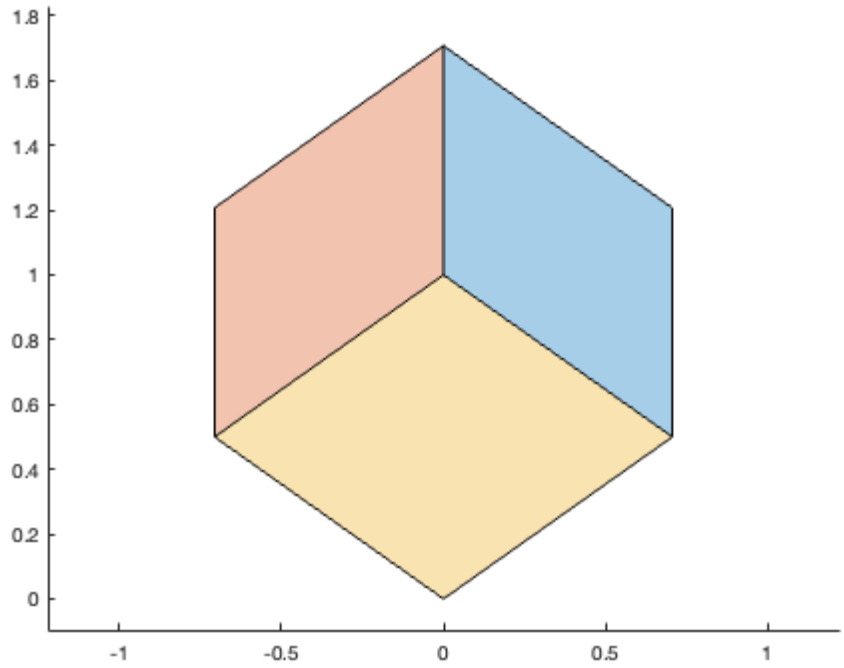
```
function DrawRotatedPolyhedronPhase3(P)
    n = size(P, 2);

    figure(1);
    axis equal;
    hold on;

    for i = 1:n
        face = P{1, i};

        if isempty(face)
            continue;
        end

        px = face(1, :);
        py = face(2, :);

        pgon = polyshape(px, py);
        plot(pgon);
    end

    hold off;
end

function C = DrawRotatedPolyhedronPhase4(P)
    n = size(P, 2);
    C = zeros(3, n);
    idx = 1;

    for i = 1:n
        face = P{1, i};

        if isempty(face)
            continue;
        end

        px = face(1, :);
        py = face(2, :);
        pz = face(3, :);
        c = [mean(px);
             mean(py);
             mean(pz)];

        C(:, idx) = c;
        idx = idx + 1;
    end

    for i = idx:n
        C(:, idx) = [];
    end

end

function C = DrawRotatedPolyhedronPhase5(P)
    n = size(P, 2);
    C = zeros(4, n);
```

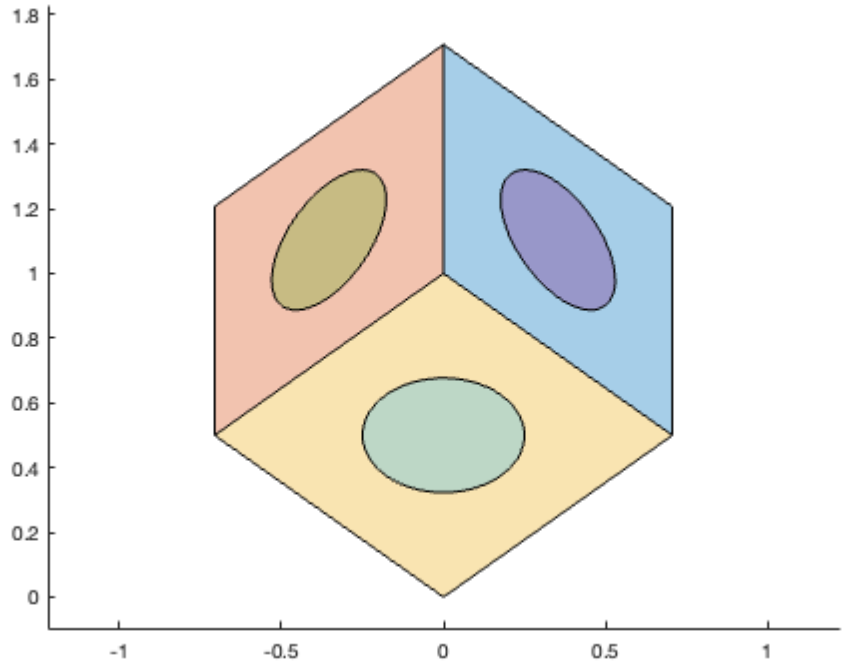
```
    idx = 1;

    for i = 1:n
        face = P{1, i};

        if isempty(face)
            continue;
        end

        px = face(1, :);
        py = face(2, :);
        pz = face(3, :);
        c = [mean(px);
             mean(py);
             mean(pz)];

        k = size(face,2);
        r = realmax;

        for j = 1:k
            v1 = [face(1, j);
                  face(2, j);
                  face(3, j)];

            v2 = [face(1, mod(j, k) + 1);
                  face(2, mod(j, k) + 1);
                  face(3, mod(j, k) + 1)];

            edge_vector = v2 - v1;
            edge_length = norm(edge_vector);

            cv1 = v1 - c;

            projection = (dot(cv1, edge_vector) / edge_length^2) * edge_vector;

            distance = norm(cv1 - projection);

            if distance < r
                r = distance;
            end
        end

        r = r / 2;

        c = [c;
             r];

        C(:, idx) = c;
        idx = idx + 1;
    end

    for i = idx:n
        C(:, idx) = [];
    end

end
```

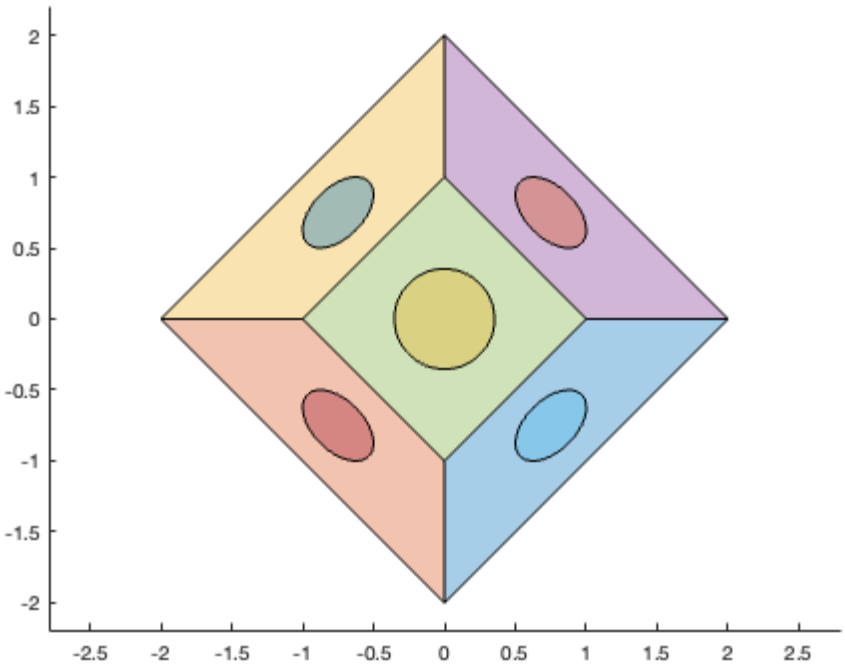
```
function C = DrawRotatedPolyhedronPhase6(P)
    n = size(P, 2);
    C = zeros(10, n);
    idx = 1;

    for i = 1:n
        face = P{1, i};

        if isempty(face)
            continue;
        end

        px = face(1, :);
        py = face(2, :);
        pz = face(3, :);
        c = [mean(px);
             mean(py);
             mean(pz)];

        k = size(face,2);
        r = realmax;

        for j = 1:k
            v1 = [face(1, j);
                  face(2, j);
                  face(3, j)];

            v2 = [face(1, mod(j, k) + 1);
                  face(2, mod(j, k) + 1);
                  face(3, mod(j, k) + 1)];

            edge_vector = v2 - v1;
            edge_length = norm(edge_vector);

            cv1 = v1 - c;

            projection = (dot(cv1, edge_vector) / edge_length^2) * edge_vector;

            distance = norm(cv1 - projection);

            if distance < r
                r = distance;
            end
        end

        r = r / 2;

        vec1 = [face(1, 1) - face(1, 2);
                face(2, 1) - face(2, 2);
                face(3, 1) - face(3, 2);];
        vec1 = vec1./norm(vec1);

        vec2 = [face(1, 1) - face(1, 3);
                face(2, 1) - face(2, 3);
                face(3, 1) - face(3, 3);];

        proj = dot(vec2, vec1) * vec1;
```

```
        vec2 = vec2 - proj;
        vec2 = vec2./norm(vec2);

        c = [c;
             r;
             vec1;
             vec2];

        C(:, idx) = c;
        idx = idx + 1;

    end

    for i = idx:n
        C(:, idx) = [];
    end

end

function G = DrawRotatedPolyhedronPhase7(C)
    n = size(C, 2);
    G = zeros(3, 100 * n);
    N = 99;
    idx = 1;

    for i = 1:n

        x = C(1, i);
        y = C(2, i);
        z = C(3, i);
        r = C(4, i);
        ux = C(5, i);
        uy = C(6, i);
        uz = C(7, i);
        vx = C(8, i);
        vy = C(9, i);
        vz = C(10, i);

        for j = 0:N

            a = 2 * pi * j / N;

            xx = x + r * cos(a) * ux + r * sin(a) * vx;
            yy = y + r * cos(a) * uy + r * sin(a) * vy;
            zz = z + r * cos(a) * uz + r * sin(a) * vz;

            c = [xx;
                 yy;
                 zz];

            G(:, idx) = c;

            idx  = idx + 1;
        end
    end
end
```

```
function DrawRotatedPolyhedronPhase8(P, G, i)
    n = size(P, 2);

    figure(i);
    axis equal;
    hold on;

    for i = 1:n
        face = P{1, i};

        if isempty(face)
            continue;
        end

        px = face(1, :);
        py = face(2, :);

        pgon = polyshape(px, py);
        plot(pgon);
    end

    k = size(G,2) / 100;

    for i = 0 : k-1
        subG = G(:, 100 * i + 1: 100* (i + 1));
        px = subG(1, :);
        py = subG(2, :);

        pgon = polyshape(px, py);
        plot(pgon);
    end

    hold off;
end

function DrawRotatedPolyhedron(M, P)

    R = DrawRotatedPolyhedronPhase1(M, P);

    R = DrawRotatedPolyhedronPhase2(R);

    C = DrawRotatedPolyhedronPhase6(R);

    G = DrawRotatedPolyhedronPhase7(C);

    DrawRotatedPolyhedronPhase8(R, G, 3);

end
```


1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-1
-0.5
0
0.5
1


1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-1
-0.5
0
0.5
1



Published with MATLAB® R2023b