Spherical Inversion Computer Simulation Write-up

Fedor Aglyamov, Jesse DeJong

The University of Texas at Austin

December 13, 2019 - Final Project for M427L-H

Abstract

Inversion in three-dimensions occurs with respect to a reference sphere, whose origin and radius dictate the shape of the inverted object/shape. While a relatively simple transformation maps points from an original object to the inverted object, conceptualizing this transformation is better understood with the help of visualization. Our MATLAB simulation application helps to visualize the inversion of three-dimensional shapes by creating animations of an object's transformation. Additionally, stereographic projection, an instance of Spherical inversion, can be better understood with the help of simulations created by our application.

1 Introduction/Background

Circular inversion, oftentimes referred to as geometric inversion, is generally attributed to the geometer Jakob Steiner, despite being discovered independently by multiple mathematicians. It is a fundamental operation of the study of inversive geometry, and has been utilized in understanding famous theorems in Geometry. Unlike in numerical inversion, where the inverse of a number x would be the reciprocal $1/x$, geometric inversion involves Euclidean transformations that maintain fundamental properties of shapes/figures. For instance, in two dimensions, circle inversion maintains the angles of a 2D figure in its inverse. Additionally, the most simple curve, the circle, maps to either itself, or a line.

To take a point mapped in Euclidean space to its inverse, the following equation is used:

$$OP * OP' = R^2 \tag{1}$$

where the point P is mapped to the point P', with regards to a reference circle with center O. P' is a point that lies on the ray OP, and R refers to the radius of the reference circle. **Figure 1.a** depicts this transformation. Additionally, because P' is considered the inverse of P, applying this same transformation on the point P' will map the point back to P. The formula also allows for the

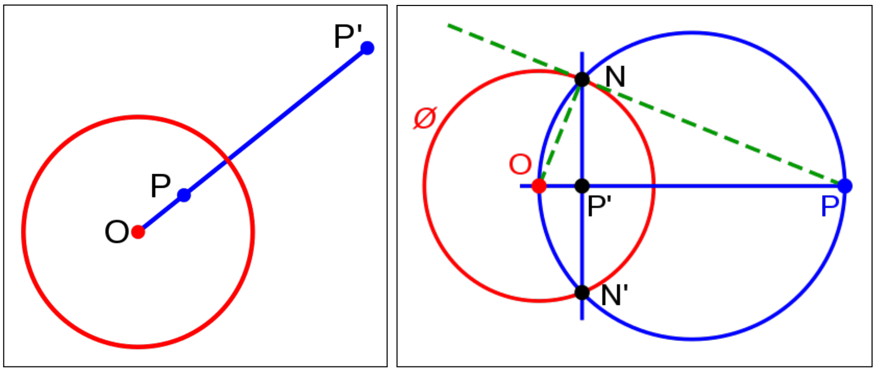

Figure 1: (a) The picture on the left depicts the transformation process applied by $OP * OP' = R^2$. (b) The picture on the right depicts the line segments that must be drawn with a straight edge and compass to manually compute the geometric inverse.

use of a compass and straightedge to determine the points of inversion, which is depicted in **Figure 1.b**.

The formula for circular inversion always maps points such that they obey the following observations:

1. A point P, inside of the reference circle, will be mapped to a point P', outside of the reference circle. Therefore, a point P, outside of the reference circle, will be mapped to a point P', inside of the reference circle.

2. A point P on the circumference/boundary of the reference circle inverts to the same point.

Thus, it can be noted that the closer a point P is to the center of the reference circle O, the farther P' will be mapped from O.

Lastly, in order to visualize this process of inversion, but in three dimensions, we decided to use MATLAB and the App Designer toolkit that MathWorks provides. Our goal was to create an application that models spherical inversion, to allow for its users to conceptualize and understand the process of geometric inversion.

2 Spherical Inversion

Circular Inversion in two dimensions extends to a third dimension in the form of Spherical Inversion. Instead of using a reference circle, a reference sphere is used (as shown in **Figure 2**). The same properties of circular inversion apply to that of spherical inversion, including the fact that spheres invert to either spheres or planes (in two dimensions, circles inverted to circles or lines).

The same equality, $OP * OP' = R^2$, still applies, and can be extended to a formula to compute the inverse of Cartesian coordinates. If $O = (x_0, y_0, z_0), P = (x, y, z), P' = (x', y', z')$, and R is the radius of the reference sphere, then the following equations can be derived:

$$x' = x_0 + \frac{k^2(x - x_0)}{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} \tag{2}$$

$$y' = y_0 + \frac{k^2(y - y_0)}{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} \tag{3}$$

$$z' = z_0 + \frac{k^2(z - z_0)}{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} \tag{4}$$

These equations can also be expressed as one equation in vector format, where x is the original point, x' is the inverse point, and x_0 is the center of the reference sphere:

$$x' = x_0 + \frac{k^2(x - x_0)}{|x - x_0|^2} \tag{5}$$

Proof for equation (5):

By the definition of geometric inversion in three dimensions, we know that O, P, and P' are co-linear. We also know that the rays OP and OP' are pointing in the same direction. It follows that

$$(x', y', z') = k(x, y, z)$$

for positive real number k. From equation (1), we know that $OP * OP' = R^2$. Because of the above formula, we get that

$$k = \frac{R^2}{OP^2}$$

after solving for k. Because P' lies on the ray OP, we can create an equation for a line that is solves for P' in terms of our other known quantities.

$$P' = O + \frac{R^2}{OP^2}(P - O)$$

Therefore, after replacing P, P', and O with their coordinate definitions, we get equation (5). Note that $OP^2 = (x - x_0)^2$.

$$x' = x_0 + \frac{k^2(x - x_0)}{|x - x_0|^2}$$

This completes the proof.

In our application, we used equation (5) to manually map points of an original shape to its inverse. **Figures 2-8** depict examples of Spherical inversions, produced by our application.

An interesting application of spherical inversion are stereographic projections, which are transformations that map spheres into planes. **Figure 5** depicts such a transformation/mapping. One instance where stereographic projections are useful is in Cartology. A spherical inversion in this case, when centered on a specific city, will transform flight paths to shortest distances from this city to other cities.

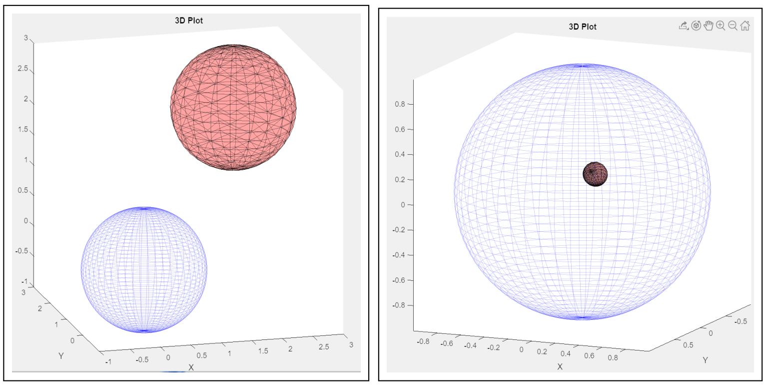

Figure 2: Geometric Inversion of a sphere (centered at $x = 2$, $y = 2$, and $z = 2$ with radius 1) that is located outside of the reference sphere (centered at $x = 0$, $y = 0$, and $z = 0$ with radius 1), to a sphere inside of the reference sphere.

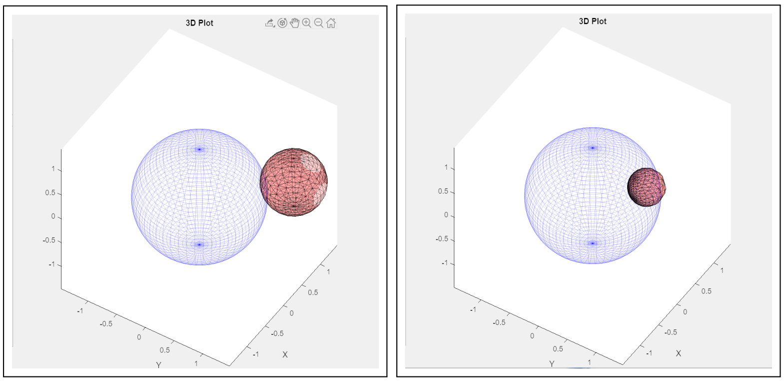

Figure 3: Geometric Inversion of a sphere (centered at $x = 1$, $y = 1$, and $z = 0$ with radius .5) that is located both inside and outside of the reference sphere (centered at $x = 0$, $y = 0$, and $z = 0$ with radius 1), to a sphere both inside and outside of the reference sphere.

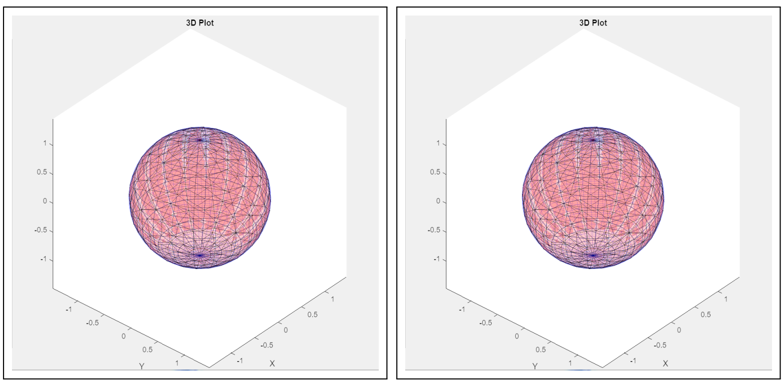

Figure 4: Geometric Inversion of a sphere (centered at $x = 0$, $y = 0$, and $z = 0$ with radius 1) that is located on top of the reference sphere (centered at $x = 0$, $y = 0$, and $z = 0$ with radius 1), to a sphere on top of the reference sphere. Points on the boundary of the reference sphere don't change.

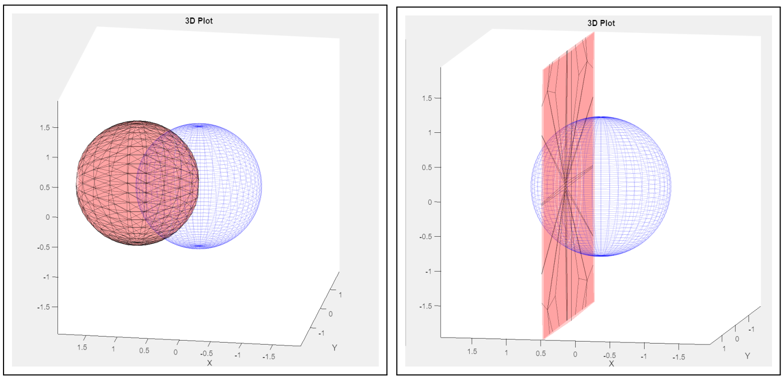

Figure 5: Geometric Inversion of a sphere (centered at $x = 1$, $y = 0$, and $z = 0$ with radius 1) that is touching the center of the reference sphere (centered at $x = 0$, $y = 0$, and $z = 0$ with radius 1), to a plane.

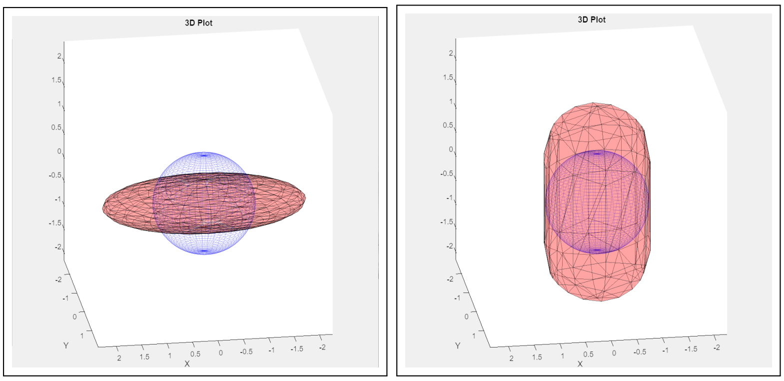

Figure 6: Geometric Inversion of an ellipse (centered at $x = 0$, $y = 0$, and $z = 0$ with semi-radius of $A = 2$, $B = 1$, and $C = 0.5$) centered at the reference sphere (centered at $x = 0$, $y = 0$, and $z = 0$ with radius 1) to its inverse.

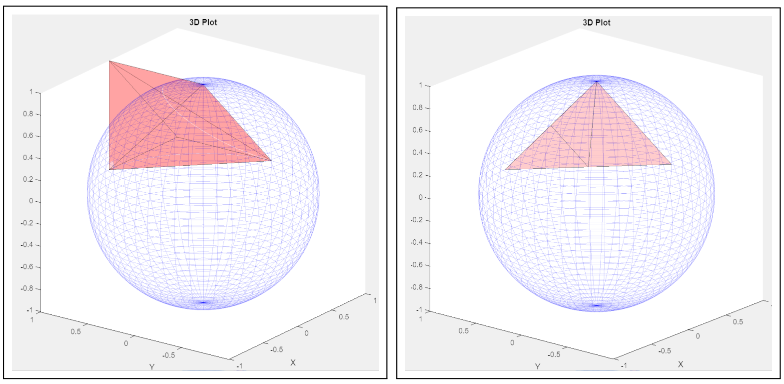

Figure 7: Geometric Inversion of custom points with respect to a reference sphere (centered at $x = 0$, $y = 0$, and $z = 0$ with radius 1). Points were entered by a user using the Spherical Inversion application.

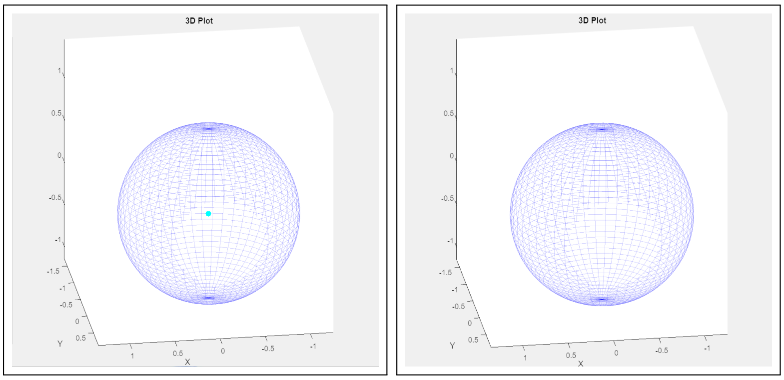

Figure 8: Geometric Inversion of a point located at $x = 0$, $y = 0$, and $z = 0$ that is at the center of the reference sphere. Because the inverse is undefined at this point, the inverse cannot be computed.

3 MATLAB Application Code

```
classdef SphereInverter < matlab.apps.AppBase
    methods (Access = private)
        % generate sphere or cylinder
        function genSphCyl(app)
            % gen coords based on sphere or cylinder
            if (app.ShapeSelect == "Sphere")
                [x, y, z] = sphere(app.Res);
                app.Coords = [x(:) * app.ShapeRad, y(:) *
                    app.ShapeRad, z(:) * app.ShapeRad];
            else
                [x, y, z] = cylinder(app.ShapeRad, app.
                   Res);
                app.Coords = [x(:), y(:), z(:) * app.
                   Height];
            end
            dimensions = size(app.Coords);
            numOfPoints = dimensions(1);
            app.NumOfPoints = numOfPoints;
            % iterate through coordinates and set center
                vals
            for num = 1:app.NumOfPoints
                app.Coords(num, 1) = app.Coords(num, 1) +
                    app.ShapeX;
                app.Coords(num, 2) = app.Coords(num, 2) +
                    app.ShapeY;
                app.Coords(num, 3) = app.Coords(num, 3) +
                    app.ShapeZ;
            end
        end

```

```
        % generate ellipsoid
        function genEllipsoid(app)
            [x, y, z] = ellipsoid(app.ShapeX, app.ShapeY, app.ShapeZ, app.ARad, app.BRad, app.CRad, app.Res);
            app.Coords = [x(:), y(:), z(:)];
            dimensions = size(app.Coords);
            numOfPoints = dimensions(1);
            app.NumOfPoints = numOfPoints;
        end

        % generate inverses
        function getInverses(app)
            numOfPoints = app.NumOfPoints;
            app.Inverses = zeros(numOfPoints, 3);
            pointNum = 1;
            % iterate through coordinates and find inverses
            while pointNum <= numOfPoints
                p = app.Coords(pointNum, :);
                o = [app.InvX, app.InvY, app.InvZ];
                inverse = getInv(app, p, o, app.InvRad);
                % if point is not at origin
                if ~isequal(inverse, 'd')
                    app.Inverses(pointNum, :) = inverse(:);
                    pointNum = pointNum + 1;
                % if point is at origin, remove it
                else
                    app.Coords(pointNum, :) = [];
                    app.Inverses(pointNum, :) = [];
                    numOfPoints = numOfPoints - 1;
                    if app.ShowPlotPoints == 1
                        p = app.Points{pointNum};
                        delete(p);
                        app.Points(pointNum) = [];
                    end
                    continue;
                end
            end
            app.NumOfPoints = numOfPoints;
        end

        % individual inversion on point's coordinates
        function i = getInv(~, p, o, r)
            i = zeros(1, 3);
            % find spherical inverse of passed point based on passed origin and radius
            if isequal(p, o)
                i = 'd';
```

```
            else
                for d = 1:3
                    i(d) = o(d) + ((r^2) * ((p(d) - o(d)))) / ((p(1) - o(1))^2 + (p(2) - o(2))^2 + (p(3) - o(3))^2);
                end
            end
        end

        % animate transition to inverse locations
        function animate(app)
            numOfPoints = app.NumOfPoints;
            steps = zeros(numOfPoints, 3);
            % get step info for each point
            for num = 1:numOfPoints
                inverse = app.Inverses(num, :);
                p = app.Coords(num, :);
                change = [(inverse(1) - p(1)) / app.Step, (inverse(2) - p(2)) / app.Step, (inverse(3) - p(3)) / app.Step];
                steps(num, :) = change;
            end
            % animate movement of each point to its inverse
            for t = 1:app.Step
                app.Coords = app.Coords + steps;
                if app.ShowPlotPoints == 1
                    for num = 1:app.NumOfPoints
                        p = app.Points{num};
                        p.XData = app.Coords(num, 1);
                        p.YData = app.Coords(num, 2);
                        p.ZData = app.Coords(num, 3);
                        drawnow limitrate;
                    end
                end
                % update surface reprsenting shape
                drawShape(app);
                pause(1 / app.Step);
            end
        end

        % show points
        function showPoints(app)
            colors = ["b", "c", "g", "m", "k", "r", "y"];
            colorDimensions = size(colors);
            numOfColors = colorDimensions(2);
            % delete current points
            hidePoints(app);
            % plot new points from coords
            app.Points = cell(1, app.NumOfPoints);
```

```
            for num = 1:app.NumOfPoints
                p = plot3(app.UIAxes, app.Coords(num, 1), app.Coords(num, 2), app.Coords(num, 3), "r.");
                p.MarkerSize = 20;
                p.Color = colors(rem(num, numOfColors) + 1);
                app.Points{num} = p;
            end
        end

        % update custom "Current Point" dropdown menu
        function updateCurPointDropdown(app)
            num = app.CurPoint;
            items = strings(1, app.NumOfPoints);
            formatStr = "%d: (%d, %d, %d)";
            newStr = sprintf(formatStr, num, app.Coords(num, 1), app.Coords(num, 2), app.Coords(num, 3));
            % generate array of items with changed option
            for i = 1:app.NumOfPoints
                if i == num
                    items(i) = newStr;
                else
                    items(i) = app.CurPointBox.Items{i};
                end
            end
            app.CurPointBox.Items = items;
            % show points if applicable
            if app.ShowPlotPoints == 1
                showPoints(app);
            end
        end
    end

        % Value changed function: CurPointBox
        function updateCurPoint(app, event)
            num = app.CurPointBox.Value;
            app.CurPoint = num;
            % update location coords based on current point
            app.CustomX = app.Coords(num, 1);
            app.CustomY = app.Coords(num, 2);
            app.CustomZ = app.Coords(num, 3);
            app.CustomXBox.Value = app.CustomX;
            app.CustomYBox.Value = app.CustomY;
            app.CustomZBox.Value = app.CustomZ;
            % show points if applicable
            if app.ShowPlotPoints == 1
                showPoints(app);
```

```
            end
        end

        % Value changed function: CustomXBox
        function updateCustomX(app, event)
            app.CustomX = app.CustomXBox.Value;
            num = app.CurPoint;
            app.Coords(num, 1) = app.CustomX;
            updateCurPointDropdown(app);
        end

        % Value changed function: CustomYBox
        function updateCustomY(app, event)
            app.CustomY = app.CustomYBox.Value;
            num = app.CurPoint;
            app.Coords(num, 2) = app.CustomY;
            updateCurPointDropdown(app);
        end

        % Value changed function: CustomZBox
        function updateCustomZ(app, event)
            app.CustomZ = app.CustomZBox.Value;
            num = app.CurPoint;
            app.Coords(num, 3) = app.CustomZ;
            updateCurPointDropdown(app);
        end
    end
end
```

4 Explanation of Code

Approximately 700 lines of code were removed from the above code segment (for brevity), but the code with relevant functionality remains. In addition to responding to changes in user input, the application supports changes in the position and radius of the inversion sphere, as well as specified objects. Custom points are also supported. The main functionality of the application is split between mapping points from their original Cartesian coordinate to their inverse coordinate, and animating the transition designated by the mapping. Equation (5) is used to calculate the location of the inverse point, and a step based system (based on the distance between the original point and the inverse point) is used to animate the transition. The MATLAB provided function *trisurf* was used to graph a boundary around individual points.

5 Screenshot of Application

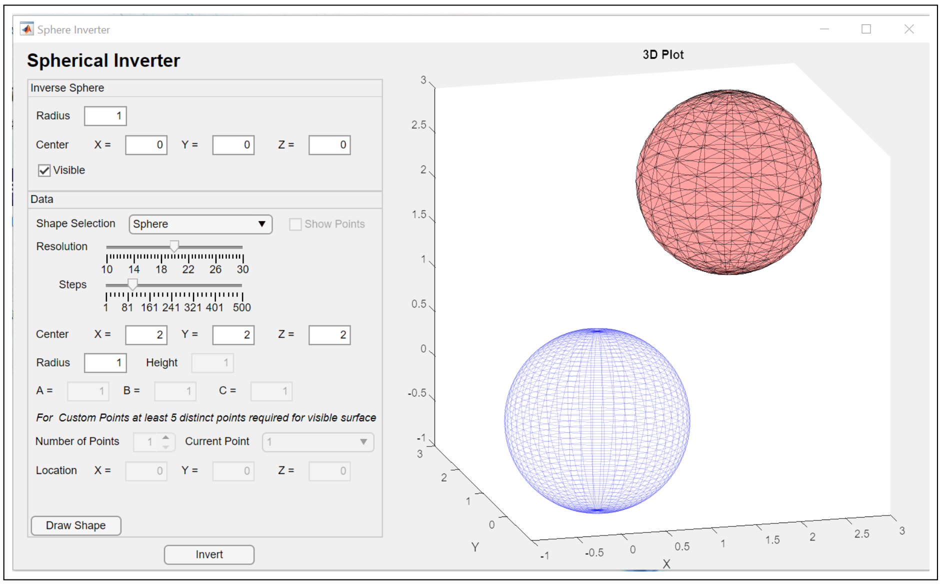

Figure 9: The Spherical Inverter Application. Input options includes the location and radius of the reference sphere (in blue), and the option to invert a sphere (in red), ellipse, cylinder, or custom data points. At the press of the "Invert" button, an animation occurs that animates the transition from the original shape to its inverse.