```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <title>3dplot</title>



  <style type='text/css'>

  </style>



<script type='text/javascript'>//<![CDATA[
window.onload=function(){
////////////////// javascript.js file contents begin here /////////////////

function registerNameSpace(ns) {
    var nsParts = ns.split(".");
    var root = window;
    var n = nsParts.length;

    for (var i = 0; i < n; i++) {
        if (typeof root[nsParts[i]] == "undefined") root[nsParts[i]] = new
Object();

        root = root[nsParts[i]];
    }
}

registerNameSpace("greg.ross.visualisation");

/*
 * This is the main class and entry point of the tool
 * and represents the Google viz API.
 * ************************************************
 */
greg.ross.visualisation.SurfacePlot = function (container) {
    this.containerElement = container;
}

greg.ross.visualisation.SurfacePlot.prototype.draw = function (data, options)
{
    var xPos = options.xPos;
    var yPos = options.yPos;
    var w = options.width;
    var h = options.height;
    var colourGradient = options.colourGradient;
    var fillPolygons = options.fillPolygons;
    var tooltips = options.tooltips;
    var xTitle = options.xTitle;
    var yTitle = options.yTitle;
    var zTitle = options.zTitle;
    var restrictXRotation = options.restrictXRotation;
```

```
    if (this.surfacePlot == undefined) this.surfacePlot = new
greg.ross.visualisation.JSSurfacePlot(xPos, yPos, w, h, colourGradient,
this.containerElement, fillPolygons, tooltips, xTitle, yTitle, zTitle,
restrictXRotation);

    this.surfacePlot.redraw(data);
}

/*
 * This class does most of the work.
 * ********************************
 */
greg.ross.visualisation.JSSurfacePlot = function(x, y, width, height,
colourGradient, targetElement, fillRegions, tooltips, xTitle, yTitle, zTitle,
restrictXRotation){
    this.targetDiv;
    var id = allocateId();
    var canvas;
    var canvasContext = null;

    var scale = greg.ross.visualisation.JSSurfacePlot.DEFAULT_SCALE;

    var currentZAngle =
greg.ross.visualisation.JSSurfacePlot.DEFAULT_Z_ANGLE;
    var currentXAngle =
greg.ross.visualisation.JSSurfacePlot.DEFAULT_X_ANGLE;

    this.data = null;
        var canvas_support_checked = false;
        var canvas_supported = true;
    var data3ds = null;
    var displayValues = null;
    var numXPoints;
    var numYPoints;
    var transformation;
    var cameraPosition;
    var colourGradient;
    var colourGradientObject;
    var renderPoints = false;

    var mouseDown1 = false;
    var mouseDown3 = false;
    var mousePosX = null;
    var mousePosY = null;
    var lastMousePos = new greg.ross.visualisation.Point(0, 0);
    var mouseButton1Up = null;
    var mouseButton3Up = null;
    var mouseButton1Down = new greg.ross.visualisation.Point(0, 0);
    var mouseButton3Down = new greg.ross.visualisation.Point(0, 0);
    var closestPointToMouse = null;
    var xAxisHeader = "";
    var yAxisHeader = "";
    var zAxisHeader = "";
    var xAxisTitleLabel = new greg.ross.visualisation.Tooltip(true);
    var yAxisTitleLabel = new greg.ross.visualisation.Tooltip(true);
    var zAxisTitleLabel = new greg.ross.visualisation.Tooltip(true);
    var tTip = new greg.ross.visualisation.Tooltip(false);
```

```
    function init(){
        transformation = new greg.ross.visualisation.Th3dtran();

        createTargetDiv();

        if (!targetDiv)
            return;

        createCanvas();
    }

    function hideTooltip(){
        tTip.hide();
    }

    function displayTooltip(e){
        var position = new greg.ross.visualisation.Point(e.x, e.y);
        tTip.show(tooltips[closestPointToMouse], 200);
    }

    function render(data){
        canvasContext.clearRect(0, 0, canvas.width, canvas.height);
        canvasContext.fillStyle = '#000';
        canvasContext.fillRect(0, 0, canvas.width, canvas.height);
        this.data = data;

        var canvasWidth = width;
        var canvasHeight = height;

        var minMargin = 20;
        var drawingDim = canvasWidth - minMargin * 2;
        var marginX = minMargin;
        var marginY = minMargin;

        transformation.init();
        transformation.rotate(currentXAngle, 0.0, currentZAngle);
        transformation.scale(scale);
        transformation.translate(drawingDim / 2.0 + marginX, drawingDim / 2.0
+ marginY, 0.0);

        cameraPosition = new greg.ross.visualisation.Point3D(drawingDim / 2.0
+ marginX, drawingDim / 2.0 + marginY, -1000.0);

        if (renderPoints) {
            for (i = 0; i < data3ds.length; i++) {
                var point3d = data3ds[i];
                canvasContext.fillStyle = '#ff2222';
                var transformedPoint =
transformation.ChangeObjectPoint(point3d);
                transformedPoint.dist = distance(transformedPoint,
cameraPosition);

                var x = transformedPoint.ax;
                var y = transformedPoint.ay;

                canvasContext.beginPath();
```

```
                var dotSize =
greg.ross.visualisation.JSSurfacePlot.DATA_DOT_SIZE;

                canvasContext.arc((x - (dotSize / 2)), (y - (dotSize / 2)),
1, 0, self.Math.PI * 2, true);
                canvasContext.fill();
            }
        }

        var axes = createAxes();
        var polygons = createPolygons(data3ds);

        for (i = 0; i < axes.length; i++) {
            polygons[polygons.length] = axes[i];
        }

        // Sort the polygons so that the closest ones are rendered last
        // and therefore are not occluded by those behind them.
        // This is really Painter's algorithm.
        polygons.sort(greg.ross.visualisation.PolygonComaparator);
        //polygons = sort(polygons);

        canvasContext.lineWidth = 1;
        canvasContext.strokeStyle = '#888';
        canvasContext.lineJoin = "round";

        for (i = 0; i < polygons.length; i++) {
            var polygon = polygons[i];

            if (polygon.isAnAxis()) {
                var p1 = polygon.getPoint(0);
                var p2 = polygon.getPoint(1);

                canvasContext.beginPath();
                canvasContext.moveTo(p1.ax, p1.ay);
                canvasContext.lineTo(p2.ax, p2.ay);
                canvasContext.stroke();
            }
            else {
                var p1 = polygon.getPoint(0);
                var p2 = polygon.getPoint(1);
                var p3 = polygon.getPoint(2);
                var p4 = polygon.getPoint(3);

                var colourValue = (p1.lz * 1.0 + p2.lz * 1.0 + p3.lz * 1.0 +
p4.lz * 1.0) / 4.0;

                // if (colourValue < 0)
                    // colourValue *= -1;

                var rgbColour = colourGradientObject.getColour(colourValue);
                var colr = "rgb(" + rgbColour.red + "," + rgbColour.green +
"," + rgbColour.blue + ")";
                canvasContext.fillStyle = colr;

                canvasContext.beginPath();
                canvasContext.moveTo(p1.ax, p1.ay);
```

```
                canvasContext.lineTo(p2.ax, p2.ay);
                canvasContext.lineTo(p3.ax, p3.ay);
                canvasContext.lineTo(p4.ax, p4.ay);
                canvasContext.lineTo(p1.ax, p1.ay);

                if (fillRegions)
                    canvasContext.fill();
                else
                    canvasContext.stroke();
            }
        }

        canvasContext.stroke();

        if (supports_canvas())
            renderAxisText(axes);
    }

    function renderAxisText(axes){
        var xLabelPoint = new greg.ross.visualisation.Point3D(0.0, 0.5, 0.0);
        var yLabelPoint = new greg.ross.visualisation.Point3D(-0.5, 0.0,
0.0);
        var zLabelPoint = new greg.ross.visualisation.Point3D(-0.5, 0.5,
0.5);

        var transformedxLabelPoint =
transformation.ChangeObjectPoint(xLabelPoint);
        var transformedyLabelPoint =
transformation.ChangeObjectPoint(yLabelPoint);
        var transformedzLabelPoint =
transformation.ChangeObjectPoint(zLabelPoint);

        var xAxis = axes[0];
        var yAxis = axes[1];
        var zAxis = axes[2];

        canvasContext.fillStyle = '#fff';

        if (xAxis.distanceFromCamera > yAxis.distanceFromCamera) {
            var xAxisLabelPosX = transformedxLabelPoint.ax;
            var xAxisLabelPosY = transformedxLabelPoint.ay;
            canvasContext.fillText(xTitle, xAxisLabelPosX, xAxisLabelPosY);
        }

        if (xAxis.distanceFromCamera < yAxis.distanceFromCamera) {
            var yAxisLabelPosX = transformedyLabelPoint.ax;
            var yAxisLabelPosY = transformedyLabelPoint.ay;
            canvasContext.fillText(yTitle, yAxisLabelPosX, yAxisLabelPosY);
        }

        if (xAxis.distanceFromCamera < zAxis.distanceFromCamera) {
            var zAxisLabelPosX = transformedzLabelPoint.ax;
            var zAxisLabelPosY = transformedzLabelPoint.ay;
            canvasContext.fillText(zTitle, zAxisLabelPosX, zAxisLabelPosY);
        }
    }
```

```
    var sort = function(array){
        var len = array.length;

        if (len < 2) {
            return array;
        }

        var pivot = Math.ceil(len / 2);
        return merge(sort(array.slice(0, pivot)), sort(array.slice(pivot)));
    }

    var merge = function(left, right){
        var result = [];
        while ((left.length > 0) && (right.length > 0)) {
            if (left[0].distanceFromCamera < right[0].distanceFromCamera) {
                result.push(left.shift());
            }
            else {
                result.push(right.shift());
            }
        }

        result = result.concat(left, right);
        return result;
    }


    function createAxes(){
        var axisOrigin = new greg.ross.visualisation.Point3D(-0.5, 0.5, 0);
        var xAxisEndPoint = new greg.ross.visualisation.Point3D(0.5, 0.5, 0);
        var yAxisEndPoint = new greg.ross.visualisation.Point3D(-0.5, -0.5,
0);
        var zAxisEndPoint = new greg.ross.visualisation.Point3D(-0.5, 0.5,
1);

        var transformedAxisOrigin =
transformation.ChangeObjectPoint(axisOrigin);
        var transformedXAxisEndPoint =
transformation.ChangeObjectPoint(xAxisEndPoint);
        var transformedYAxisEndPoint =
transformation.ChangeObjectPoint(yAxisEndPoint);
        var transformedZAxisEndPoint =
transformation.ChangeObjectPoint(zAxisEndPoint);

        var axes = new Array();

        var xAxis = new greg.ross.visualisation.Polygon(cameraPosition,
true);
        xAxis.addPoint(transformedAxisOrigin);
        xAxis.addPoint(transformedXAxisEndPoint);
        xAxis.calculateCentroid();
        xAxis.calculateDistance();
        axes[axes.length] = xAxis;

        var yAxis = new greg.ross.visualisation.Polygon(cameraPosition,
true);
        yAxis.addPoint(transformedAxisOrigin);
```

```
        yAxis.addPoint(transformedYAxisEndPoint);
        yAxis.calculateCentroid();
        yAxis.calculateDistance();
        axes[axes.length] = yAxis;

        var zAxis = new greg.ross.visualisation.Polygon(cameraPosition,
true);
        zAxis.addPoint(transformedAxisOrigin);
        zAxis.addPoint(transformedZAxisEndPoint);
        zAxis.calculateCentroid();
        zAxis.calculateDistance();
        axes[axes.length] = zAxis;

        return axes;
    }

    function createPolygons(data3D){
        var i;
        var j;
        var polygons = new Array();
        var index = 0;

        for (i = 0; i < numXPoints - 1; i++) {
            for (j = 0; j < numYPoints - 1; j++) {
                var polygon = new
greg.ross.visualisation.Polygon(cameraPosition, false);

                var p1 = transformation.ChangeObjectPoint(data3D[j + (i *
numYPoints)]);
                var p2 = transformation.ChangeObjectPoint(data3D[j + (i *
numYPoints) + numYPoints]);
                var p3 = transformation.ChangeObjectPoint(data3D[j + (i *
numYPoints) + numYPoints + 1]);
                var p4 = transformation.ChangeObjectPoint(data3D[j + (i *
numYPoints) + 1]);

                polygon.addPoint(p1);
                polygon.addPoint(p2);
                polygon.addPoint(p3);
                polygon.addPoint(p4);
                polygon.calculateCentroid();
                polygon.calculateDistance();

                polygons[index] = polygon;
                index++;
            }
        }

        return polygons;
    }

    function getDefaultColourRamp(){
        var colour1 = {
            red: 0,
            green: 0,
            blue: 255
        };
```

```
        var colour2 = {
            red: 0,
            green: 255,
            blue: 255
        };
        var colour3 = {
            red: 0,
            green: 255,
            blue: 0
        };
        var colour4 = {
            red: 255,
            green: 255,
            blue: 0
        };
        var colour5 = {
            red: 255,
            green: 0,
            blue: 0
        };
        return [colour1, colour2, colour3, colour4, colour5];
    }

    this.redraw = function(data){
        numXPoints = data.getNumberOfRows() * 1.0;
        numYPoints = data.getNumberOfColumns() * 1.0;

        var minZValue = Number.MAX_VALUE;
        var maxZValue = Number.MIN_VALUE;

        for (var i = 0; i < numXPoints; i++) {
            for (var j = 0; j < numYPoints; j++) {
                var value = data.getFormattedValue(i, j) * 1.0;

                if (value < minZValue)
                    minZValue = value;

                if (value > maxZValue)
                    maxZValue = value;
            }
        }

        var cGradient;

        if (colourGradient)
            cGradient = colourGradient;
        else
            cGradient = getDefaultColourRamp();

        // if (minZValue < 0 && (minZValue*-1) > maxZValue)
          // maxZValue = minZValue*-1;

        colourGradientObject = new
greg.ross.visualisation.ColourGradient(minZValue, maxZValue, cGradient);

        var canvasWidth = width;
        var canvasHeight = height;
```

```
        var minMargin = 20;
        var drawingDim = canvasWidth - minMargin * 2;
        var marginX = minMargin;
        var marginY = minMargin;

        if (canvasWidth > canvasHeight) {
            drawingDim = canvasHeight - minMargin * 2;
            marginX = (canvasWidth - drawingDim) / 2;
        }
        else
            if (canvasWidth < canvasHeight) {
                drawingDim = canvasWidth - minMargin * 2;
                marginY = (canvasHeight - drawingDim) / 2;
            }

        var xDivision = 1 / (numXPoints - 1);
        var yDivision = 1 / (numYPoints - 1);
        var xPos, yPos;
        var i, j;
        var numPoints = numXPoints * numYPoints;
        data3ds = new Array();
        var index = 0;

        // Calculate 3D points.
        for (i = 0, xPos = -0.5; i < numXPoints; i++, xPos += xDivision) {
            for (j = 0, yPos = 0.5; j < numYPoints; j++, yPos -= yDivision) {
                var x = xPos;
                var y = yPos;

                data3ds[index] = new greg.ross.visualisation.Point3D(x, y,
data.getFormattedValue(i, j));
                index++;
            }
        }

        render(data);
    }

    function allocateId(){
        var count = 0;
        var name = "surfacePlot";

        do {
            count++;
        }
        while (document.getElementById(name + count))

        return name + count;
    }

    function createTargetDiv(){
        this.targetDiv = document.createElement("div");
        this.targetDiv.id = id;
        this.targetDiv.className = "surfaceplot";
        this.targetDiv.style.background = '#ffffff'
        this.targetDiv.style.position = 'absolute';
```

```
        if (!targetElement)
            document.body.appendChild(this.targetDiv);
        else {
            this.targetDiv.style.position = 'relative';
            targetElement.appendChild(this.targetDiv);
        }

        this.targetDiv.style.left = x + "px";
        this.targetDiv.style.top = y + "px";
    }

    function getInternetExplorerVersion()    // Returns the version of
Internet Explorer or a -1
    // (indicating the use of another browser).
    {
        var rv = -1; // Return value assumes failure.
        if (navigator.appName == 'Microsoft Internet Explorer') {
            var ua = navigator.userAgent;
            var re = new RegExp("MSIE ([0-9]{1,}[\.0-9]{0,})");
            if (re.exec(ua) != null)
                rv = parseFloat(RegExp.$1);
        }
        return rv;
    }

    function supports_canvas(){
               if (canvas_support_checked) return canvas_supported;

                canvas_support_checked = true;
         canvas_supported = !!document.createElement('canvas').getContext;
                return canvas_supported;
    }

    function createCanvas(){
        canvas = document.createElement("canvas");

        if (!supports_canvas()) {
            G_vmlCanvasManager.initElement(canvas);
            canvas.style.width = width;
            canvas.style.height = height;
        }

        canvas.className = "surfacePlotCanvas";
        canvas.setAttribute("width", width);
        canvas.setAttribute("height", height);
        canvas.style.left = '0px';
        canvas.style.top = '0px';

        targetDiv.appendChild(canvas);

        canvasContext = canvas.getContext("2d");
        canvasContext.font = "bold 18px sans-serif";
        canvasContext.clearRect(0, 0, canvas.width, canvas.height);

        canvasContext.fillStyle = '#000';
```

```
    canvasContext.fillRect(0, 0, canvas.width, canvas.height);

    canvasContext.beginPath();
    canvasContext.rect(0, 0, canvas.width, canvas.height);
    canvasContext.strokeStyle = '#888';
    canvasContext.stroke();

    canvas.onmousemove = mouseIsMoving;
    canvas.onmouseout = hideTooltip;
    canvas.onmousedown = mouseDownd;
    canvas.onmouseup = mouseUpd;

    //added by edupont
    canvas.addEventListener("touchstart", mouseDownd, false);
           canvas.addEventListener("touchmove", mouseIsMoving, false);
           canvas.addEventListener("touchend", mouseUpd, false);
           canvas.addEventListener("touchcancel", hideTooltip, false);
}

function mouseDownd(e){
    if (isShiftPressed(e)) {
        mouseDown3 = true;
        mouseButton3Down = getMousePositionFromEvent(e);
    }
    else {
        mouseDown1 = true;
        mouseButton1Down = getMousePositionFromEvent(e);
    }
}

function mouseUpd(e){
    if (mouseDown1) {
        mouseButton1Up = lastMousePos;
    }
    else
        if (mouseDown3) {
            mouseButton3Up = lastMousePos;
        }

    mouseDown1 = false;
    mouseDown3 = false;
}

function mouseIsMoving(e){
    var currentPos = getMousePositionFromEvent(e);

    if (mouseDown1) {
        hideTooltip();
        calculateRotation(currentPos);
    }
    else
        if (mouseDown3) {
            hideTooltip();
            calculateScale(currentPos);
        }
        else {
            closestPointToMouse = null;
```

```
                var closestDist = Number.MAX_VALUE;

                for (var i = 0; i < data3ds.length; i++) {
                    var point = data3ds[i];
                    var dist = distance({
                        x: point.ax,
                        y: point.ay
                    }, currentPos);

                    if (dist < closestDist) {
                        closestDist = dist;
                        closestPointToMouse = i;
                    }
                }

                if (closestDist > 16) {
                    hideTooltip();
                    return;
                }

                displayTooltip(currentPos);
            }

                    return false;
    }

    function isShiftPressed(e){
        var shiftPressed = 0;

        if (parseInt(navigator.appVersion) > 3) {
            var evt = navigator.appName == "Netscape" ? e : event;

            if (navigator.appName == "Netscape" &&
parseInt(navigator.appVersion) == 4) {
                // NETSCAPE 4 CODE
                var mString = (e.modifiers + 32).toString(2).substring(3, 6);
                shiftPressed = (mString.charAt(0) == "1");
            }
            else {
                // NEWER BROWSERS [CROSS-PLATFORM]
                shiftPressed = evt.shiftKey;
            }

            if (shiftPressed)
                return true;
        }

        return false;
    }

    function getMousePositionFromEvent(e){
        if (getInternetExplorerVersion() > -1) {
            var e = window.event;

            if (e.srcElement.getAttribute('Stroked') == true) {
                if (mousePosX == null || mousePosY == null)
                    return;
```

```
            }
            else {
                mousePosX = e.offsetX;
                mousePosY = e.offsetY;
            }
        }
        else
            if (e.layerX || e.layerX == 0) // Firefox
            {
                mousePosX = e.layerX;
                mousePosY = e.layerY;
            }
            else if (e.offsetX || e.offsetX == 0) // Opera
            {
                mousePosX = e.offsetX;
                mousePosY = e.offsetY;
            }
                    else if (e.touches[0].pageX || e.touches[0].pageX == 0)
//touch events
            {
                    mousePosX = e.touches[0].pageX;
                    mousePosY = e.touches[0].pageY;
            }

        var currentPos = new greg.ross.visualisation.Point(mousePosX,
mousePosY);

        return currentPos;
    }

    function calculateRotation(e){
        lastMousePos = new
greg.ross.visualisation.Point(greg.ross.visualisation.JSSurfacePlot.DEFAULT_Z
_ANGLE, greg.ross.visualisation.JSSurfacePlot.DEFAULT_X_ANGLE);

        if (mouseButton1Up == null) {
            mouseButton1Up = new
greg.ross.visualisation.Point(greg.ross.visualisation.JSSurfacePlot.DEFAULT_Z
_ANGLE, greg.ross.visualisation.JSSurfacePlot.DEFAULT_X_ANGLE);
        }

        if (mouseButton1Down != null) {
            lastMousePos = new greg.ross.visualisation.Point(mouseButton1Up.x
+ (mouseButton1Down.x - e.x),//
 mouseButton1Up.y + (mouseButton1Down.y - e.y));
        }

        currentZAngle = lastMousePos.x % 360;
        currentXAngle = lastMousePos.y % 360;

               if (restrictXRotation) {

                       if (currentXAngle < 0)
                               currentXAngle = 0;
                       else
                               if (currentXAngle > 90)
                                       currentXAngle = 90;
```

```
                }

        closestPointToMouse = null;
        render(data);
    }

    function calculateScale(e){
        lastMousePos = new greg.ross.visualisation.Point(0,
greg.ross.visualisation.JSSurfacePlot.DEFAULT_SCALE /
greg.ross.visualisation.JSSurfacePlot.SCALE_FACTOR);

        if (mouseButton3Up == null) {
            mouseButton3Up = new greg.ross.visualisation.Point(0,
greg.ross.visualisation.JSSurfacePlot.DEFAULT_SCALE /
greg.ross.visualisation.JSSurfacePlot.SCALE_FACTOR);
        }

        if (mouseButton3Down != null) {
            lastMousePos = new greg.ross.visualisation.Point(mouseButton3Up.x
+ (mouseButton3Down.x - e.x),//
 mouseButton3Up.y + (mouseButton3Down.y - e.y));
        }

        scale = lastMousePos.y *
greg.ross.visualisation.JSSurfacePlot.SCALE_FACTOR;

        if (scale < greg.ross.visualisation.JSSurfacePlot.MIN_SCALE)
            scale = greg.ross.visualisation.JSSurfacePlot.MIN_SCALE + 1;
        else
            if (scale > greg.ross.visualisation.JSSurfacePlot.MAX_SCALE)
                scale = greg.ross.visualisation.JSSurfacePlot.MAX_SCALE - 1;

        lastMousePos.y = scale /
greg.ross.visualisation.JSSurfacePlot.SCALE_FACTOR;

        closestPointToMouse = null;
        render(data);
    }

    init();
}

/**
 * Given two coordinates, return the Euclidean distance
 * between them
 */
function distance(p1, p2){
    return Math.sqrt(((p1.x - p2.x) *
    (p1.x -
    p2.x)) +
    ((p1.y - p2.y) * (p1.y - p2.y)));
}

/*
 * Matrix3d: This class represents a 3D matrix.
 * ****************************************
```

```
 */
greg.ross.visualisation.Matrix3d = function(){
    this.matrix = new Array();
    this.numRows = 4;
    this.numCols = 4;

    this.init = function(){
        this.matrix = new Array();

        for (var i = 0; i < this.numRows; i++) {
            this.matrix[i] = new Array();
        }
    }

    this.getMatrix = function(){
        return this.matrix;
    }

    this.matrixReset = function(){
        for (var i = 0; i < this.numRows; i++) {
            for (var j = 0; j < this.numCols; j++) {
                this.matrix[i][j] = 0;
            }
        }
    }

    this.matrixIdentity = function(){
        this.matrixReset();
        this.matrix[0][0] = this.matrix[1][1] = this.matrix[2][2] =
this.matrix[3][3] = 1;
    }

    this.matrixCopy = function(newM){
        var temp = new greg.ross.visualisation.Matrix3d();
        var i, j;

        for (i = 0; i < this.numRows; i++) {
            for (j = 0; j < this.numCols; j++) {
                temp.getMatrix()[i][j] = (this.matrix[i][0] *
newM.getMatrix()[0][j]) + (this.matrix[i][1] * newM.getMatrix()[1][j]) +
(this.matrix[i][2] * newM.getMatrix()[2][j]) + (this.matrix[i][3] *
newM.getMatrix()[3][j]);
            }
        }

        for (i = 0; i < this.numRows; i++) {
            this.matrix[i][0] = temp.getMatrix()[i][0];
            this.matrix[i][1] = temp.getMatrix()[i][1];
            this.matrix[i][2] = temp.getMatrix()[i][2];
            this.matrix[i][3] = temp.getMatrix()[i][3];
        }
    }

    this.matrixMult = function(m1, m2){
        var temp = new greg.ross.visualisation.Matrix3d();
        var i, j;
```

```
        for (i = 0; i < this.numRows; i++) {
            for (j = 0; j < this.numCols; j++) {
                temp.getMatrix()[i][j] = (m2.getMatrix()[i][0] *
m1.getMatrix()[0][j]) + (m2.getMatrix()[i][1] * m1.getMatrix()[1][j]) +
(m2.getMatrix()[i][2] * m1.getMatrix()[2][j]) + (m2.getMatrix()[i][3] *
m1.getMatrix()[3][j]);
            }
        }

        for (i = 0; i < this.numRows; i++) {
            m1.getMatrix()[i][0] = temp.getMatrix()[i][0];
            m1.getMatrix()[i][1] = temp.getMatrix()[i][1];
            m1.getMatrix()[i][2] = temp.getMatrix()[i][2];
            m1.getMatrix()[i][3] = temp.getMatrix()[i][3];
        }
    }

    this.init();
}

/*
 * Point3D: This class represents a 3D point.
 * ****************************************
 */
greg.ross.visualisation.Point3D = function(x, y, z){
    this.displayValue = "";

    this.lx;
    this.ly;
    this.lz;
    this.lt;

    this.wx;
    this.wy;
    this.wz;
    this.wt;

    this.ax;
    this.ay;
    this.az;
    this.at;

    this.dist;

    this.initPoint = function(){
        this.lx = this.ly = this.lz = this.ax = this.ay = this.az = this.at =
this.wx = this.wy = this.wz = 0;
        this.lt = this.wt = 1;
    }

    this.init = function(x, y, z){
        this.initPoint();
        this.lx = x;
        this.ly = y;
        this.lz = z;

        this.ax = this.lx;
```

```
        this.ay = this.ly;
        this.az = this.lz;
    }

    function multiply(p){
        var Temp = new Point3D();
        Temp.lx = this.lx * p.lx;
        Temp.ly = this.ly * p.ly;
        Temp.lz = this.lz * p.lz;
        return Temp;
    }

    function getDisplayValue(){
        return displayValue;
    }

    function setDisplayValue(displayValue){
        this.displayValue = displayValue;
    }

    this.init(x, y, z);
}

/*
 * Polygon: This class represents a polygon on the surface plot.
 * ************************************************************
 */
greg.ross.visualisation.Polygon = function(cameraPosition, isAxis){
    this.points = new Array();
    this.cameraPosition = cameraPosition;
    this.isAxis = isAxis;
    this.centroid = null;
    this.distanceFromCamera = null;

    this.isAnAxis = function(){
        return this.isAxis;
    }

    this.addPoint = function(point){
        this.points[this.points.length] = point;
    }

    this.distance = function(){
        return this.distance2(this.cameraPosition, this.centroid);
    }

    this.calculateDistance = function(){
        this.distanceFromCamera = this.distance();
    }

    this.calculateCentroid = function(){
        var xCentre = 0;
        var yCentre = 0;
        var zCentre = 0;

        var numPoints = this.points.length * 1.0;
```

```
        for (var i = 0; i < numPoints; i++) {
            xCentre += this.points[i].ax;
            yCentre += this.points[i].ay;
            zCentre += this.points[i].az;
        }

        xCentre /= numPoints;
        yCentre /= numPoints;
        zCentre /= numPoints;

        this.centroid = new greg.ross.visualisation.Point3D(xCentre, yCentre,
zCentre);
    }

    this.distance2 = function(p1, p2){
        return ((p1.ax - p2.ax) * (p1.ax - p2.ax)) + ((p1.ay - p2.ay) *
(p1.ay - p2.ay)) + ((p1.az - p2.az) * (p1.az - p2.az));
    }

    this.getPoint = function(i){
        return this.points[i];
    }
}

/*
 * PolygonComaparator: Class used to sort arrays of polygons.
 * ************************************************************
 */
greg.ross.visualisation.PolygonComaparator = function(p1, p2){
    var diff = p1.distanceFromCamera - p2.distanceFromCamera;

    if (diff == 0)
        return 0;
    else
        if (diff < 0)
            return -1;
        else
            if (diff > 0)
                return 1;

    return 0;
}

/*
 * Th3dtran: Class for matrix manipuation.
 * ************************************************************
 */
greg.ross.visualisation.Th3dtran = function(){
    this.matrix;
    this.rMat;
    this.rMatrix;
    this.objectMatrix;
    this.local = true;

    this.init = function(){
        this.matrix = new greg.ross.visualisation.Matrix3d();
        this.rMat = new greg.ross.visualisation.Matrix3d();
```

```
    this.rMatrix = new greg.ross.visualisation.Matrix3d();
    this.objectMatrix = new greg.ross.visualisation.Matrix3d();

    this.initMatrix();
}

this.initMatrix = function(){
    this.matrix.matrixIdentity();
    this.objectMatrix.matrixIdentity();
}

this.translate = function(x, y, z){
    this.rMat.matrixIdentity();
    this.rMat.getMatrix()[3][0] = x;
    this.rMat.getMatrix()[3][1] = y;
    this.rMat.getMatrix()[3][2] = z;

    if (this.local) {
        this.objectMatrix.matrixCopy(this.rMat);
    }
    else {
        this.matrix.matrixCopy(this.rMat);
    }
}

this.rotate = function(x, y, z){
    var rx = x * (Math.PI / 180.0);
    var ry = y * (Math.PI / 180.0);
    var rz = z * (Math.PI / 180.0);

    this.rMatrix.matrixIdentity();
    this.rMat.matrixIdentity();
    this.rMat.getMatrix()[1][1] = Math.cos(rx);
    this.rMat.getMatrix()[1][2] = Math.sin(rx);
    this.rMat.getMatrix()[2][1] = -(Math.sin(rx));
    this.rMat.getMatrix()[2][2] = Math.cos(rx);
    this.rMatrix.matrixMult(this.rMatrix, this.rMat);

    this.rMat.matrixIdentity();
    this.rMat.getMatrix()[0][0] = Math.cos(ry);
    this.rMat.getMatrix()[0][2] = -(Math.sin(ry));
    this.rMat.getMatrix()[2][0] = Math.sin(ry);
    this.rMat.getMatrix()[2][2] = Math.cos(ry);
    this.rMat.matrixMult(this.rMatrix, this.rMat);

    this.rMat.matrixIdentity();
    this.rMat.getMatrix()[0][0] = Math.cos(rz);
    this.rMat.getMatrix()[0][1] = Math.sin(rz);
    this.rMat.getMatrix()[1][0] = -(Math.sin(rz));
    this.rMat.getMatrix()[1][1] = Math.cos(rz);
    this.rMat.matrixMult(this.rMatrix, this.rMat);

    if (this.local) {
        this.objectMatrix.matrixCopy(this.rMatrix);
    }
    else {
        this.matrix.matrixCopy(this.rMatrix);
```

```
        }
    }

    this.scale = function(scale){
        this.rMat.matrixIdentity();
        this.rMat.getMatrix()[0][0] = scale;
        this.rMat.getMatrix()[1][1] = scale;
        this.rMat.getMatrix()[2][2] = scale;

        if (this.local) {
            this.objectMatrix.matrixCopy(this.rMat);
        }
        else {
            this.matrix.matrixCopy(this.rMat);
        }
    }

    this.changeLocalObject = function(p){
        p.wx = (p.ax * this.matrix.getMatrix()[0][0] + p.ay *
this.matrix.getMatrix()[1][0] + p.az * this.matrix.getMatrix()[2][0] +
this.matrix.getMatrix()[3][0]);
        p.wy = (p.ax * this.matrix.getMatrix()[0][1] + p.ay *
this.matrix.getMatrix()[1][1] + p.az * this.matrix.getMatrix()[2][1] +
this.matrix.getMatrix()[3][1]);
        p.wz = (p.ax * this.matrix.getMatrix()[0][2] + p.ay *
this.matrix.getMatrix()[1][2] + p.az * this.matrix.getMatrix()[2][2] +
this.matrix.getMatrix()[3][2]);

        return p;
    }

    this.ChangeObjectPoint = function(p){
        p.ax = (p.lx * this.objectMatrix.getMatrix()[0][0] + p.ly *
this.objectMatrix.getMatrix()[1][0] + p.lz *
this.objectMatrix.getMatrix()[2][0] + this.objectMatrix.getMatrix()[3][0]);
        p.ay = (p.lx * this.objectMatrix.getMatrix()[0][1] + p.ly *
this.objectMatrix.getMatrix()[1][1] + p.lz *
this.objectMatrix.getMatrix()[2][1] + this.objectMatrix.getMatrix()[3][1]);
        p.az = (p.lx * this.objectMatrix.getMatrix()[0][2] + p.ly *
this.objectMatrix.getMatrix()[1][2] + p.lz *
this.objectMatrix.getMatrix()[2][2] + this.objectMatrix.getMatrix()[3][2]);

        return p;
    }

    this.init();
}

/*
 * Point: A simple 2D point.
 * ************************************************************
 */
greg.ross.visualisation.Point = function(x, y){
    this.x = x;
    this.y = y;
}
```

```
/*
 * This function displays tooltips and was adapted from original code by
Michael Leigeber.
 * See http://www.leigeber.com/
 */
greg.ross.visualisation.Tooltip = function(useExplicitPositions){
    var top = 3;
    var left = 3;
    var maxw = 300;
    var speed = 10;
    var timer = 20;
    var endalpha = 95;
    var alpha = 0;
    var tt, t, c, b, h;
    var ie = document.all ? true : false;

    this.show = function(v, w){
        if (tt == null) {
            tt = document.createElement('div');
            tt.style.color = "#fff";

            tt.style.position = 'absolute';
            tt.style.display = 'block';

            t = document.createElement('div');

            t.style.display = 'block';
            t.style.height = '5px';
            t.style.marginleft = '5px';
            t.style.overflow = 'hidden';

            c = document.createElement('div');

            b = document.createElement('div');

            tt.appendChild(t);
            tt.appendChild(c);
            tt.appendChild(b);
            document.body.appendChild(tt);

            if (!ie) {
                tt.style.opacity = 0;
                tt.style.filter = 'alpha(opacity=0)';
            }
            else
                tt.style.opacity = 1;


        }

        if (!useExplicitPositions)
            document.onmousemove = this.pos;

        tt.style.display = 'block';
        c.innerHTML = '<span style="font-weight:bold; font-family: arial;">'
+ v + '</span>';
        tt.style.width = w ? w + 'px' : 'auto';
```

```
        if (!w && ie) {
            t.style.display = 'none';
            b.style.display = 'none';
            tt.style.width = tt.offsetWidth;
            t.style.display = 'block';
            b.style.display = 'block';
        }

        if (tt.offsetWidth > maxw) {
            tt.style.width = maxw + 'px';
        }

        h = parseInt(tt.offsetHeight) + top;

        if (!ie) {
            clearInterval(tt.timer);
            tt.timer = setInterval(function(){
                fade(1)
            }, timer);
        }
    }

    this.setPos = function(e){
        tt.style.top = e.y + 'px';
        tt.style.left = e.x + 'px';
    }

    this.pos = function(e){
        var u = ie ? event.clientY + document.documentElement.scrollTop :
e.pageY;
        var l = ie ? event.clientX + document.documentElement.scrollLeft :
e.pageX;
        tt.style.top = (u - h) + 'px';
        tt.style.left = (l + left) + 'px';
    }

    function fade(d){
        var a = alpha;

        if ((a != endalpha && d == 1) || (a != 0 && d == -1)) {
            var i = speed;

            if (endalpha - a < speed && d == 1) {
                i = endalpha - a;
            }
            else
                if (alpha < speed && d == -1) {
                    i = a;
                }

            alpha = a + (i * d);
            tt.style.opacity = alpha * .01;
            tt.style.filter = 'alpha(opacity=' + alpha + ')';
        }
        else {
            clearInterval(tt.timer);
```

```
            if (d == -1) {
                tt.style.display = 'none';
            }
        }
    }

    this.hide = function(){
        if (tt == null)
            return;

        if (!ie) {
            clearInterval(tt.timer);
            tt.timer = setInterval(function(){
                fade(-1)
            }, timer);
        }
        else {
            tt.style.display = 'none';
        }
    }
}

greg.ross.visualisation.JSSurfacePlot.DEFAULT_X_ANGLE = 47;
greg.ross.visualisation.JSSurfacePlot.DEFAULT_Z_ANGLE = 47;
greg.ross.visualisation.JSSurfacePlot.DATA_DOT_SIZE = 3;
greg.ross.visualisation.JSSurfacePlot.DEFAULT_SCALE = 350;
greg.ross.visualisation.JSSurfacePlot.MIN_SCALE = 50;
greg.ross.visualisation.JSSurfacePlot.MAX_SCALE = 1100;
greg.ross.visualisation.JSSurfacePlot.SCALE_FACTOR = 1.4;


////////////////////// javascript.js file contents end here //////////////////




////////////////////// ColourGradient.js file contents begin here
//////////////////


/*
 * ColourGradient.js
 *
 * Written by Greg Ross
 *
 * Copyright 2012 ngmoco, LLC.  Licensed under the Apache License, Version
2.0 (the "License");
 * you may not use this file except in compliance with the License.  You may
obtain a copy of
 * the License at http://www.apache.org/licenses/LICENSE-2.0.  Unless
required by applicable
 * law or agreed to in writing, software distributed under the License is
distributed on an
```

```
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied.
 * See the License for the specific language governing permissions and
limitations under the
 * License.
 *
 */

/**
 * Class that is used to define a path through RGB space.
 * @author Greg Ross
 * @constructor
 * @param minValue the value that will return the first colour on the path in
RGB space
 * @param maxValue the value that will return the last colour on the path in
RGB space
 * @param rgbColourArray the set of colours that defines the dirctional path
through RGB space.
 * The length of the array must be greater than two.
 */
greg.ross.visualisation.ColourGradient = function (minValue, maxValue,
rgbColourArray) {
    /**
     * Return a colour from a position on the path in RGB space that is
proportioal to
     * the number specified in relation to the minimum and maximum values
from which the
     * bounds of the path are derived.
     * @member ColourGradient
     * @param value
     */
    this.getColour = function (value) {
        if (isNaN(value)) return rgbColourArray[0];

        if (value < minValue || rgbColourArray.length == 1) return
rgbColourArray[0];
        else if (value < minValue || value > maxValue ||
rgbColourArray.length == 1) {
            return rgbColourArray[rgbColourArray.length - 1];
        }

        var scaledValue = mapValueToZeroOneInterval(value, minValue,
maxValue);

        return getPointOnColourRamp(scaledValue);
    };

    function getPointOnColourRamp(value) {
        var numberOfColours = rgbColourArray.length;
        var scaleWidth = 1 / (numberOfColours - 1);
        var index = (value / scaleWidth);
        var index = parseInt(index + "");

        index = index == (numberOfColours - 1) ? index - 1 : index;

        var rgb1 = rgbColourArray[index];
        var rgb2 = rgbColourArray[index + 1];
```

```
        if (rgb1 == void 0 || rgb2 == void 0) return rgbColourArray[0];

        var closestToOrigin, furthestFromOrigin;

        if (distanceFromRgbOrigin(rgb1) > distanceFromRgbOrigin(rgb2)) {
            closestToOrigin = rgb2;
            furthestFromOrigin = rgb1;
        } else {
            closestToOrigin = rgb1;
            furthestFromOrigin = rgb2;
        }

        var t;

        if (closestToOrigin == rgb2) t = 1 - mapValueToZeroOneInterval(value,
index * scaleWidth, (index + 1) * scaleWidth);
        else t = mapValueToZeroOneInterval(value, index * scaleWidth, (index
+ 1) * scaleWidth);

        var diff = [
        t * (furthestFromOrigin.red - closestToOrigin.red),
        t * (furthestFromOrigin.green - closestToOrigin.green),
        t * (furthestFromOrigin.blue - closestToOrigin.blue)];

        var r = closestToOrigin.red + diff[0];
        var g = closestToOrigin.green + diff[1];
        var b = closestToOrigin.blue + diff[2];

        r = parseInt(r);
        g = parseInt(g);
        b = parseInt(b);

        var colr = {
            red: r,
            green: g,
            blue: b
        };

        return colr;
    }

    function distanceFromRgbOrigin(rgb) {
        return (rgb.red * rgb.red) + (rgb.green * rgb.green) + (rgb.blue *
rgb.blue);
    }

    function mapValueToZeroOneInterval(value, minValue, maxValue) {
        if (minValue == maxValue) return 0;

        var factor = (value - minValue) / (maxValue - minValue);
        return factor;
    }
};
```

```
////////////////// ColourGradient.js file contents end here
//////////////////


function setUp() {
    var numRows = 100;
    var numCols = 100;

    var tooltipStrings = new Array();

    var data = {
        values: [],
        getNumberOfRows: function () {
            return numRows;
        },
        getNumberOfColumns: function () {
            return numCols;
        },
        getFormattedValue: function (i, j) {
            return this.values[i][j];
        }
    };

    var d = 360 / numRows;
    var idx = 0;

    for (var i = 0; i < numRows; i++) {

        data.values.push([]);

        for (var j = 0; j < numCols; j++) {
            var value = (Math.cos(i * d * Math.PI / 180.0) * Math.cos(j * d *
Math.PI / 180.0));

            data.values[i].push(value / 4.0);

            tooltipStrings[idx] = "x:" + i + ", y:" + j + " = " + value;
            idx++;
        }
    }

    var surfacePlot = new
greg.ross.visualisation.SurfacePlot(document.getElementById("surfacePlotDiv")
);

    // Don't fill polygons in IE. It's too slow.
    var fillPly = true;

    // Define a colour gradient.
    var colour1 = {
        red: 0,
        green: 0,
        blue: 255
    };
    var colour2 = {
        red: 0,
        green: 255,
```

```
        blue: 255
    };
    var colour3 = {
        red: 0,
        green: 255,
        blue: 0
    };
    var colour4 = {
        red: 255,
        green: 255,
        blue: 0
    };
    var colour5 = {
        red: 255,
        green: 0,
        blue: 0
    };
    var colours = [colour1, colour2, colour3, colour4, colour5];

    // Axis labels.
    var xAxisHeader = "X";
    var yAxisHeader = "Y";
    var zAxisHeader = "Z";

    var options = {
        xPos: 50,
        yPos: 50,
        width: 500,
        height: 500,
        colourGradient: colours,
        fillPolygons: fillPly,
        tooltips: tooltipStrings,
        xTitle: xAxisHeader,
        yTitle: yAxisHeader,
        zTitle: zAxisHeader,
        restrictXRotation: false
    };

    surfacePlot.draw(data, options);
}

setUp();
}//]]>

</script>


</head>
<body>
  <div id='surfacePlotDiv'>
    <!-- SurfacePlot goes here... -->
</div>

</body>


</html>
```