CASTING SHADOWS:
Shading Digital Elevation Models Using Ray Tracing

Kevin M. Gill*
Fidelity Investments, Merrimack NH

Keywords: *Ray Tracing, Digital Elevation Models, Computer Graphics*

Abstract

This article describes the algorithms and methods used in generating shadows on digital elevation models. These include formulas that are very common in computer graphics applications and are often provided by specific frameworks (i.e. OpenGL). The basics of model rendering are covered from the structure of the source data to the interpolation of hypsometric/bathymetric tint colors. Hillshading is a common component seen on elevation models and is covered with the formulas provided either in the text or in the appendix. The primary algorithm presented is the calculation of shadows using ray tracing. The basic components of the method are presented leaving the reader free to provide any number of enhancements that may be needed in their own software. The methods are based on code from the jDem846 open source project managed by the author. Code listings are provided below and available in full online.

1 Overview

1.1 Digital Elevation Models

Digital Elevation Models (alternately "Digital Terrain Model") are computer generated two or three dimensional models of an area of terrain found on a planet, asteroid, or other terrestrial body. They are generated via complex algorithms using elevation data generated via remote satellite sensing or direct manual surveying. The models have a number of uses including terrain and surface analysis, engineering and infrastructure design, geographic information systems, line-of-site analysis, etc [1].

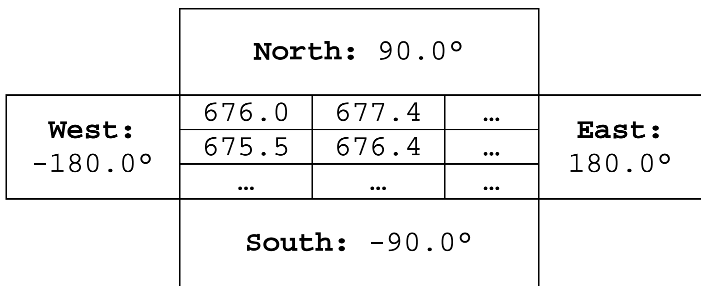
1.2 Source Data

Data can represent surface elevation, such as the top of trees or buildings, or bare surface terrain elevation. Raw data is generally organized into a gridded raster format, or height map, with each data point containing the elevation of the geographic location it represents. There is any number of resolutions used in the preparation of the data which will need to be accounted for when developing software that will consume this data to make sure that it is projected correctly.

The raster is commonly represented as cylindrical (as opposed to spherical) with the north/south latitudinal direction making up rows and east/west longitudinal direction making up the individual samples, or columns, in each row. Simple formats like a GridFloat are simply rasters of 32 bit floating point values which are relatively simple to read. Check the format specifications for instructions or attempt to find an existing code library that supports it.

Copyright © 2012 by Kevin M. Gill. Published by Rivier College, with permission.

ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).



		North: 90.0°		
West: -180.0°	676.0	677.4	…	East: 180.0°
	675.5	676.4	…	
	…	…	…	
		South: -90.0°		

Table 1 Example raster data layout

1.3 Variables

There are a number of variables that will keep coming up that will be derived from your source data or from the parameters controlling the rendering of your model.

- **Latitude**: The vertical, or North/South, measurement of a geographic point on a globe. Usually signified with ϕ (Greek letter phi).
- **Longitude**: The horizontal, or East/West, measurement of a geographic point on a globe. Usually signified with λ (Greek letter lambda).
- **North**: The upper latitudinal limit of the data
- **South**: The lower latitudinal limit of the data
- **East**: The right longitudinal limit of the data
- **West**: The left longitudinal limit of the data
- **Rows**: The number of rows making up the raster
- **Columns**: The number of columns per row in the raster
- **Latitude Resolution**: The height, in degrees, of each data point.
  - Should be equal to $\frac{(NORTH-SOUTH)}{ROWS}$
- **Longitude Resolution**: The width, in degrees, of each data point.
  - Should be equal to $\frac{(EAST-WEST)}{COLUMNS}$
- **Latitude Resolution (meters):** The height, in meters, of a particular data point. Will often need to be recalculated for each latitude/longitude pair for enhanced accuracy.
- **Longitude Resolution (meters):** The width, in meters, of a particular data point. Like the latitude resolution, it will need to be recalculated for each latitude/longitude pair for enhanced accuracy.
- **Minimum Elevation:** The lowest elevation in the data. Especially useful if using a calculation based on a points elevation percentile.
- **Maximum Elevation:** The highest elevation in the data.
- **No Data**: A value in the raster that indicates that the point should not be considered as part of the terrain.
- **Earth Mean Radius**: The mean radius of the Earth (6,371 km). This is assuming a perfectly spherical planet, which it is not. For enhanced accuracy you'll need to consider such things as the equatorial radius, polar radius, flattening, etc., to represent the oblate spheroid shape of the globe.

- **Solar Azimuth Angle**: The horizontal angle of light source being cast onto the model surface. Will often need to be recalculated for each latitude/longitude pair for higher accuracy. Ranges between 0° and 360° clockwise. Generally signified by α (Greek letter alpha).
- **Solar Elevation Angle**: The vertical angle of the light source being cast onto the model surface. Light the azimuth, this will need to be recalculated for each latitude/longitude pair for higher accuracy. Ranged between 0° and 90°. Generally signified by θ_s (Greek letter theta).

2 Elevation Tinting (Hypsometric/Bathymetric Tinting)

The most basic means of visualizing the levels of elevation is with a gradient tinting, called Hypsometric tinting for above sea-level and bathymetric tinting for below sea-level coloring. The primary concept is to end up with a specific color for each elevation represented in the data.

For a percentage based tint levels, finding the proper color would follow take three steps:

1. Take the ratio of an elevation to the minimum and maximum values. This would allow you to select the color stops to be used in the tint that fall directly above and below this ratio.

$$a = \frac{(elevation - minimum)}{(maximum - minimum)}$$

Equation 1

2. Take a second ratio of the above ratio to the upper and lower bounds as defined by the color stops.

$$r = \frac{a - lower}{(upper - lower)}$$

Equation 2

3. Interpolate the color from the stops using the ratio from step 2.

$$red_{elevation} = red_{lower} * (1 - r) + red_{upper} * r$$
$$green_{elevation} = green_{lower} * (1 - r) + green_{upper} * r$$
$$blue_{elevation} = blue_{lower} * (1 - r) + blue_{upper} * r$$

Equation 3

	RGB	Level
	E5E5E5	100%
	B2B2B2	68%
	797979	52%
	7C2123	39%
	9A4600	18%
	E4CF7E	5%
	248234	3%
	006246	0%

Table 2 Example hypsometric tint levels

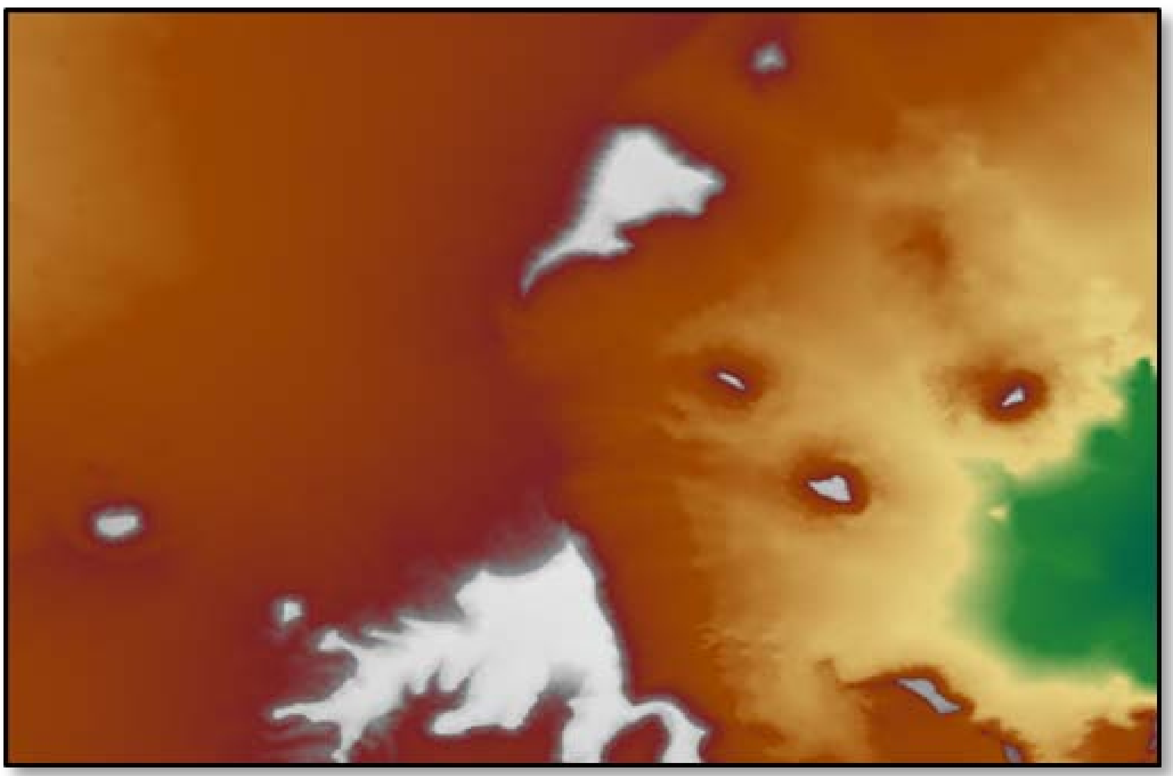


Figure 1 Model with hypsometric tinting

3 Shaded Relief / Hill Shading

Hypsometric or bathymetric tinting are great for visualizing the basic relationships of varying levels of elevation over a model, however they are weak for displaying the true roughness of a terrain. To add the concept of illumination over a model, a series of algorithms are applied to indicate specular highlighting and shadows. By general convention, hill-shading is applied to appear to have a top-left lighting (sunlight from the Northwest). However it is often useful to make the light source configurable so that shading can represent specific times of day or to highlight specific features in the terrain.

The concept is to visualize the data as a surface and how that surface interacts with the light being projected onto it. To do this, we need to tease this interaction out of the data. To do this for a particular latitude/longitude pair, we need the elevation for that point, and the elevation directly north, south, east, and west of it. With this information we can calculate the normal of that plain.

Initially, the light needs to be assigned a source vector which can be used to determine the surface highlights of each latitude/longitude pair being rendered. To make is light source easier to configure, you would specify a solar elevation and azimuth (defined above) either manually or via formulas using date and time. A basic vector pointing straight back can be rotated by the solar azimuth and elevation angles to provide the illumination source.

$$Sun_x = 0$$
$$Sun_y = 0$$
$$Sun_z = -1$$
$$Source = VectorRotate(\theta_s, -\alpha, Sun)$$

Equation 4

To calculate the normal for a vertex, we take the normal of the four surrounding triangles that make up a square with the target latitude/longitude pair in the middle. For each triangle, the following calculation is used to get the normal.

$$A = P_0 - P_1$$
$$B = P_1 - P_2$$
$$C = CrossProduct\,(A, B)$$
$$N = Normalize(C)$$

Equation 5

Then we get the mean normal for the vertex by averaging the four triangles using the following calculation.

$$N_x = \frac{NW_x + SW_x + SE_x + NE_x}{4}$$
$$N_y = \frac{NW_y + SW_y + SE_y + NE_y}{4}$$
$$N_z = \frac{NW_z + SW_z + SE_z + NE_z}{4}$$

Equation 6

And, finally we need to know which way the normal is pointing in relation to the light source. The value this calculates using the dot-product formula can be thought of as a relationship of the position of the normal to the light source on a sphere.

$$A = Normalize(P_0)$$
$$B = Normalize(P_1)$$
$$dot = A_x * B_x + A_y * B_y + A_z * B_z$$

Equation 7

Given a position on that sphere for the light source, the normal will decrease from 1.0 to -1.0 as it gets further away. If the normal is pointing directly towards the light source, the dot product will yield the maximum of 1.0. Inversely if the normal if pointed directly away from the light source, the dot product will yield the minimum of -1.0. In terms of specular highlighting, the maximum dot product will result in the maximum illumination of that point and the minimum dot product will result in the minimum illumination (or maximum shadow) of the point.

With the dot product in hand, we can extend the hypsometric coloring technique by adjusting the illumination. On a basic level, this is done by adjusting the point color brightness using the following formula.

$$Color = \begin{cases} Color * (1 + dot)\ if\ dot < 0 \\ Color + (1 - Color) * dot\ if\ dot \geq 0 \end{cases}$$

Equation 8

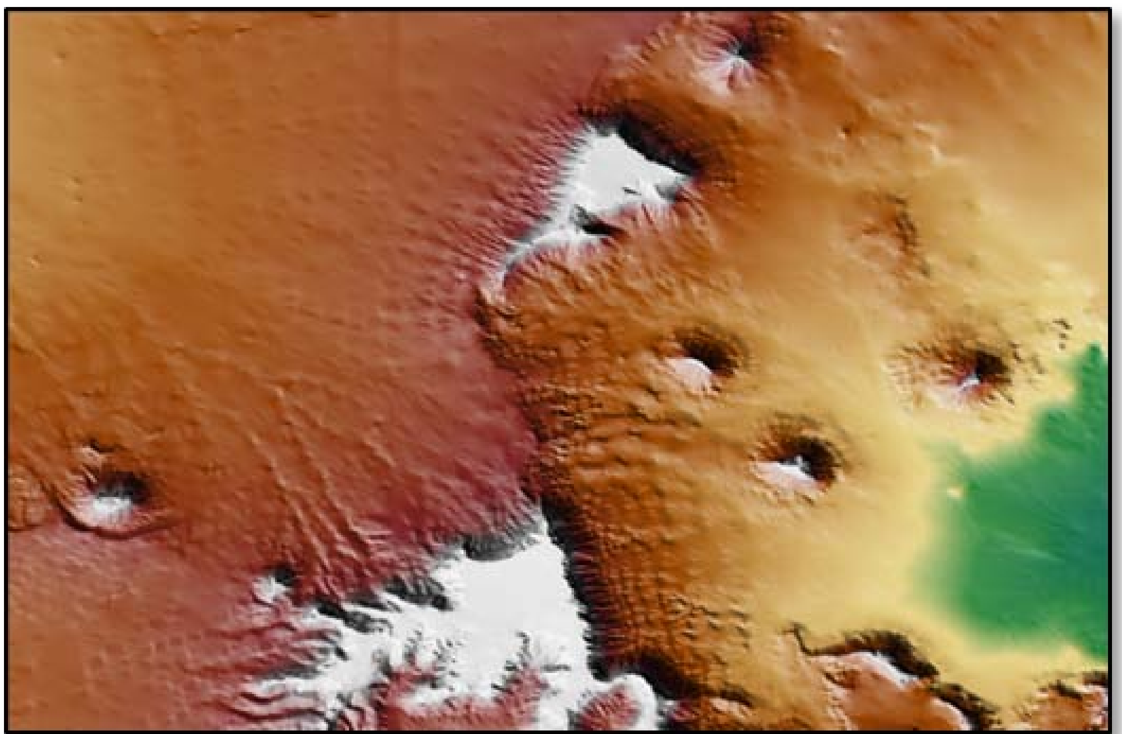


Figure 2 Model with hillshading

4 Ray Tracing

4.1 Overview

The previous methods are very common methods in computer graphics for shading textures. Depending on your configuration, some of them will be accelerated in hardware making them very fast. The method that isn't as common, and certainly not as fast is applying ray tracing to project shadows.

The concept is pretty simple: Determining if a point is in a shadow by following a straight line from that point to the light source. Should the line be blocked, then the point is within a shadow. For most 3D computer graphics, the implementation is pretty simple given the size of objects and the relatively small memory requirements to contain them. In elevation modeling, you are often dealing with gigabytes of raster data making simple equations and collision detection difficult.

For elevation models, the ray tracing will follow this basic concept: For each point checked, we move from the point out and up towards the light source. We do this by calculating spherical coordinates

for each iteration at an increasing radius. Those coordinates can then be transformed into the latitude, longitude, and ray path elevation. The test is that if the elevation at that latitude/longitude pair is greater than the elevation of the ray path, then the ray is blocked. If no terrain elevation along the path exceeds the ray elevation, then it is determined to be unblocked.

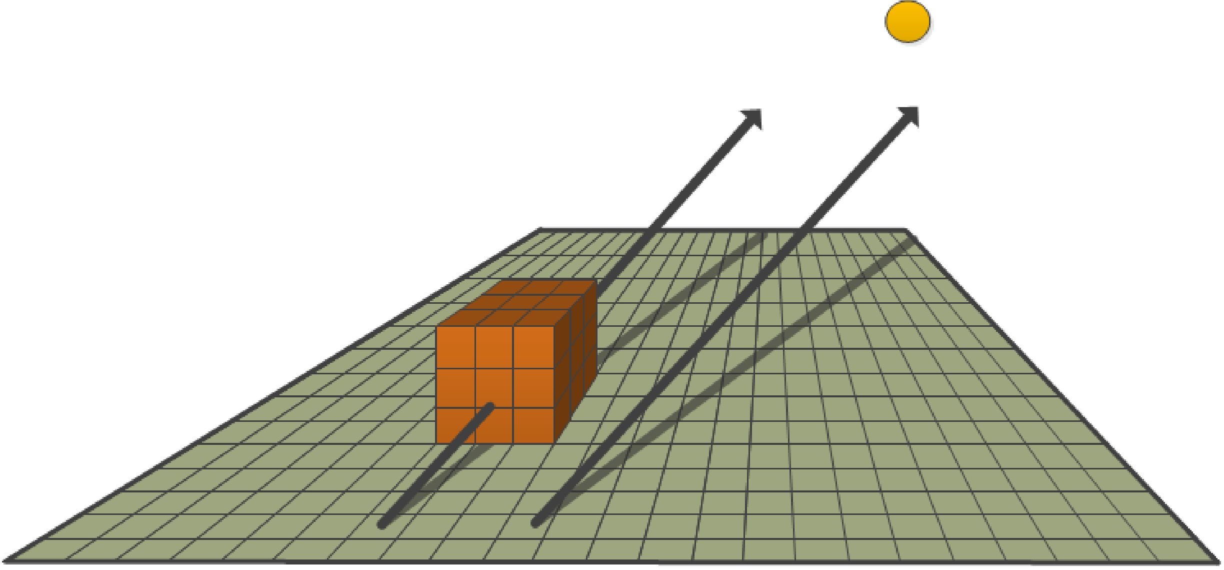


Figure 3 Blocked and unblocked ray over a grid

4.2 Implementation

The actual implementation is just a bunch of math. First, to enforce accuracy and reduce duplication, the radius is set according to the latitude/longitude resolutions (grid size) to make each sure each check checks a new value once. Using the Pythagorean Theorem the radius interval can be calculated as the hypotenuse to a right triangle with the latitude and longitude resolutions as the two legs.

$$r_i = \sqrt{Res_{lat}^2 + Res_{lon}^2}$$

Equation 9

Each check will expand the search radius by this amount, which if each iteration were drawn would look like a wireframe cone.

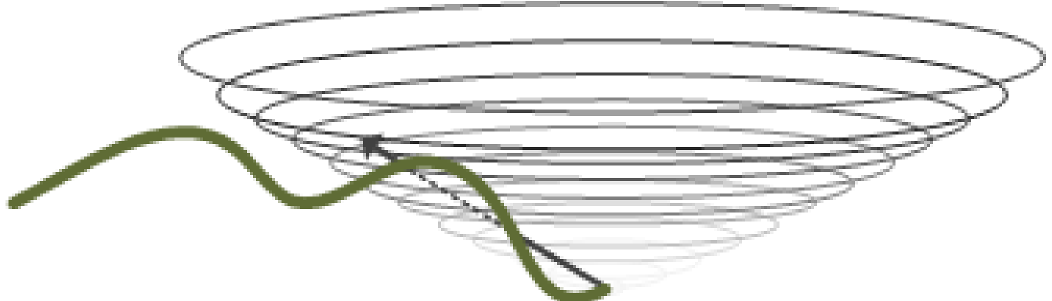


Figure 4 Ray along an increasing radius with a simulated block

The algorithm takes the form of a loop, repeating until the ray is determined whether it is blocked or not. The first radius check will equal the radius interval. That radius along with the light azimuth (θ) and elevation (φ) angles are passed to a spherical calculation to find the X/Y/Z points of the ray.

$$y = \sqrt{r^2 - (r * \cos\varphi)^2}$$

$$r_0 = \sqrt{r^2 - y^2}$$

$$b = r_0 + \cos\theta$$

$$z = \sqrt{r_0^2 - b^2}$$

$$x = \sqrt{r_0^2 - z^2}$$

$$x = \begin{cases} -x, if\ \theta > 90\ and\ \theta \leq 270 \\ x\ if\ \theta \leq 90\ or\ \theta > 270 \end{cases}$$

$$y = \begin{cases} |y|, if\ \varphi \geq 0 \\ |y| * -1, if\ \varphi < 0 \end{cases}$$

$$z = \begin{cases} z * -1, if\ \theta \leq 180 \\ z, if\ \theta > 180 \end{cases}$$

Equation 10

Since the X/Y/Z points are already values in relation to coordinates, we find the ray's location by adding X to the center latitude and subtracting Z from the center longitude. Calculating the elevation is a function of the elevation at the center point, the Y value of the ray path, the radius interval and the data resolution in meters.

$$e_p = e_c + \frac{y}{r_i} * res_m$$

Equation 11

With the ray path latitude, longitude, and elevation known, we fetch the true elevation from the data and perform some tests (see RayTracing.java in the appendix for full code listing). Note: It is important at this point to make sure that the tests cover any possible condition to avoid remaining in an infinite loop.

Test 1: If the terrain elevation is greater (higher) than the ray path elevation, then the ray is blocked.

```
if (pointElevation > rayElevation) {
      isBlocked = true;
      break;
}
```

Test 2: If the ray path latitude/longitude coordinates fall outside of the bounds of the source data the ray is assumed not to be blocked since there is no more data to prove otherwise.

```
if (latitude > northLimit ||
           latitude < southLimit ||
           longitude > eastLimit ||
           longitude < westLimit) {
     isBlocked = false;
     break;
}
```

Test 3: If the terrain elevation is a "No Data" value, the ray cannot be determined blocked or not, and the loop continues to the next radius.

```
if (pointElevation == ELEV_NO_DATA) {
     continue;
}
```

Test 4: If the path elevation is greater than the maximum elevation in the dataset, the ray is assumed not to be blocked as there are no other terrain features that could be in the way.

```
if (rayElevation > maxElevationValue) {
     isBlocked = false;
     break;
}
```

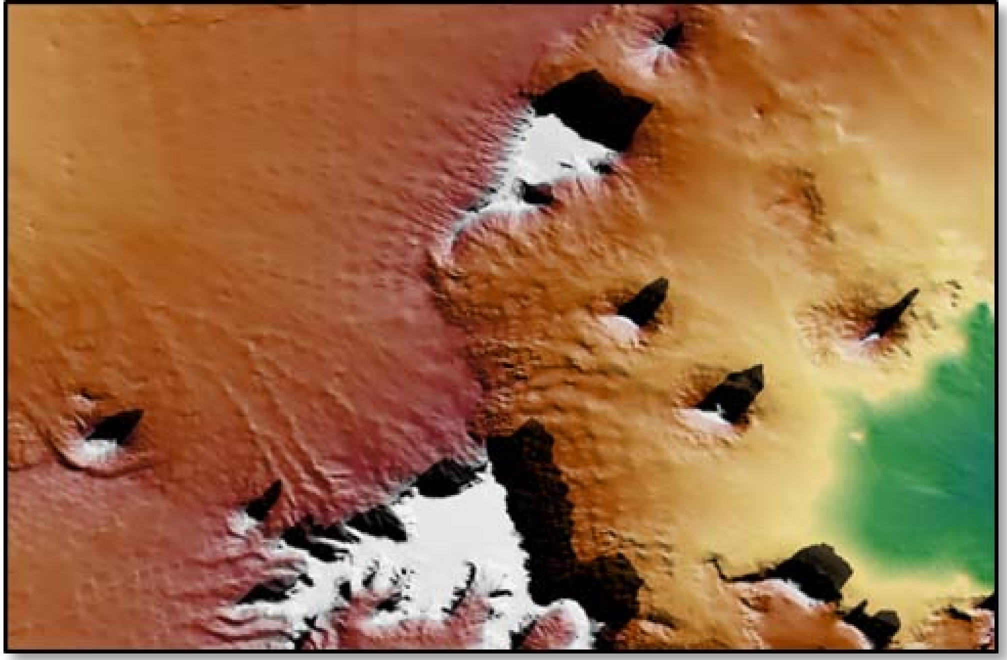


Figure 5 Model with ray traced shadows

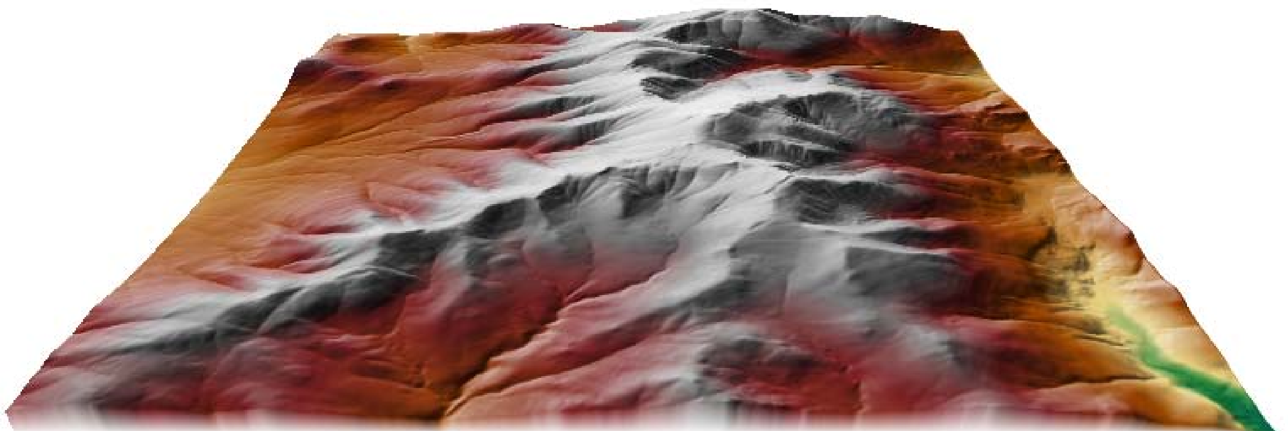


Figure 6 Mountains without ray traced shadows

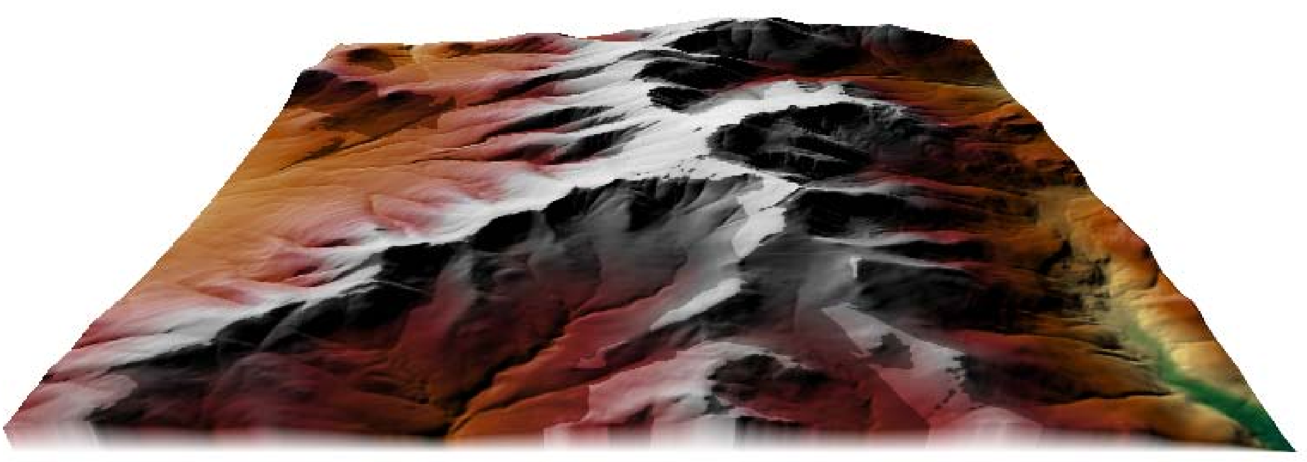


Figure 7 Mountains with ray traced shadows

4.3 Limitations

The ray tracing algorithm as presented is a basic implementation and does not cover such things are data validation which vary depending on the software requirements. It is better suited for smaller areas (as opposed to entire continents or planets) as it does not account for the curvature of the Earth. It does, however, add a certain amount of overhead to a rendering process and would not be suited for real-time rendering at higher resolutions.

5.0 Formula Reference

5.1 Cross Product

$$Cp_x = P0_y * P1_z - P1_y * P0_z$$

$$Cp_y = P0_z * P1_x - P1_z * P0_x$$

$$Cp_z = P0_x * P1_y - P1_x * P0_y$$

Equation 12

5.2 Dot Product

$$A = Normalize(P_0)$$

$$B = Normalize(P_1)$$

$$dot = A_x * B_x + A_y * B_y + A_z * B_z$$

Equation 13

5.3 Vertex Length

$$L = \sqrt{P_x^2 + P_y^2 + P_z^2}$$

$$L = \begin{cases} L \ if \ L > 0 \\ 1 \ if \ L = 0 \end{cases}$$

Equation 14

5.4 Surface Normal

$$A = P_0 - P_1$$

$$B = P_1 - P_2$$

$$C = CrossProduct\,(A, B)$$

$$N = Normalize(C)$$

Equation 15

5.5 Vertex Normalize

$$L = Length(P)$$

$$P_x = \frac{P_x}{L}$$

$$P_y = \frac{P_y}{L}$$

$$P_z = \frac{P_z}{L}$$

Equation 16

6 Code Listings

6.1 Spheres.java

```
public class Spheres
{

        private static final double RAD_90 = Math.toRadians(90.0);
        private static final double RAD_180 = Math.toRadians(180.0);
        private static final double RAD_270 = Math.toRadians(270.0);


        public static void getPoint3D(double theta,
                                      double phi,
                                      double radius,
                                      double[] points)
        {
                theta = Math.toRadians(theta);
                phi = Math.toRadians(phi);

                double _y = Math.sqrt(Math.pow(radius, 2) - Math.pow(radius * Math.cos(phi), 2));
                double r0 = Math.sqrt(Math.pow(radius, 2) - Math.pow(_y, 2));

                double _b = r0 * Math.cos(theta );
                double _z = Math.sqrt(Math.pow(r0, 2) - Math.pow(_b, 2));
                double _x = Math.sqrt(Math.pow(r0, 2) - Math.pow(_z, 2));


                if (theta <= RAD_90) {
                        _z *= -1.0;
                } else if (theta  <= RAD_180) {
                        _x *= -1.0;
                        _z *= -1.0;
                } else if (theta  <= RAD_270) {
                        _x *= -1.0;
                }

                if (phi >= 0) {
                        _y = Math.abs(_y);
                } else {
                        _y = Math.abs(_y) * -1;
                }

                points[0] = _x;
                points[1] = _y;
```

```
                points[2] = _z;

        }
}
```

6.2 RasterDataFetchHandler.java

```
public interface RasterDataFetchHandler
{
        public double getRasterData(double latitude, double longitude) throws Exception;
}
```

6.3 RayTracingException.java

```
@SuppressWarnings("serial")
public class RayTracingException extends Exception
{
        public RayTracingException()
        {
                super();
        }

        public RayTracingException(String message, Throwable cause)
        {
                super(message, cause);
        }

        public RayTracingException(String message)
        {
                super(message);
        }

        public RayTracingException(Throwable cause)
        {
                super(cause);
        }
}
```

6.4 RayTracing.java

```
public class RayTracing
{
        protected static final double ELEV_NO_DATA = -9999.99;

        private RasterDataFetchHandler rasterDataFetchHandler;

        private double longitudeResolution;
        private double latitudeResolution;
        private double radiusInterval;

        private double metersResolution;

        private double northLimit;
        private double southLimit;
        private double eastLimit;
        private double westLimit;

        private double maxElevationValue;

        private double[] points = new double[3];


        public RayTracing(
                                double latitudeResolution,
                                double longitudeResolution,
                                double metersResolution,
```

```
                                double northLimit,
                                double southLimit,
                                double eastLimit,
                                double westLimit,
                                double maxElevationValue,
                                RasterDataFetchHandler rasterDataFetchHandler)
{
        setRasterDataFetchHandler(rasterDataFetchHandler);
        setLatitudeResolution(latitudeResolution);
        setLongitudeResolution(longitudeResolution);
        setMetersResolution(metersResolution);
        setNorthLimit(northLimit);
        setSouthLimit(southLimit);
        setEastLimit(eastLimit);
        setWestLimit(westLimit);
        setMaxElevationValue(maxElevationValue);

        // Calculate a default radius interval as the
        // hypotenuse of a right triangle with the latitude
        // and longitude resolutions as the legs.
        setRadiusInterval(Math.sqrt(Math.pow(getLatitudeResolution(), 2)
                 + Math.pow(getLongitudeResolution(), 2)));
}


/** Ray trace from coordinate to the source of light. If there is another
 * higher elevation that blocks the way, then return true. Return false if
 * the trace either not blocked or it exceeds the maximum data elevation and
 * it can then be assumed to not be blocked. Points with no data are skipped
 * and the loop continues on. Note: This initial implementation
 * assumes a flat Earth and is therefore not technically accurate.
 *
 * @param centerLatitude Geographic latitude of the point being tested for shadow
 * @param centerLongitude Geographic longitude of the point being tested for shadow
 * @param centerElevation Elevation of the test point.
 * @return True if the ray's path is blocked, otherwise returns false.
 * @throws RayTracingException Thrown if an error is detected when fetching an
 * elevation along the ray path.
 */
public boolean isRayBlocked(double remoteElevationAngle,
                        double remoteAzimuth,
                        double centerLatitude,
                        double centerLongitude,
                        double centerElevation) throws RayTracingException
{
        boolean isBlocked = false;

        // Variables for use during each pass
        double radius = radiusInterval;

        while (true) {

                // Fetch points in space following the path of the azimuth and elevation angles
                // at the current radius.
                Spheres.getPoint3D(remoteAzimuth, remoteElevationAngle, radius, points);

                // Latitude/Longitude pair for the path at the current radius
                double latitude = centerLatitude + points[0];
                double longitude = centerLongitude - points[2];

                // Check if the latitude/longitude point is outside of the dataset.
                // If we get to this point, we assume that the ray is not blocked because
                // we are unable to prove otherwise.
                if (latitude > northLimit ||
                                latitude < southLimit ||
                                longitude > eastLimit ||
                                longitude < westLimit) {
                        isBlocked = false;
                        break;
```

```
                }

                // Calculate the elevation of the path at the current radius.
                double resolution = (points[1] / radiusInterval);
                double rayElevation = centerElevation + (resolution * metersResolution);

                // Fetch the elevation value
                double pointElevation = 0;
                try {
                        pointElevation = rasterDataFetchHandler.getRasterData(latitude,
                                                                        longitude);
                } catch (Exception ex) {
                        throw new RayTracingException("Failed to get elevation for point: "
                                        + ex.getMessage(), ex);
                }

                // Increment for the next pass radius
                radius += radiusInterval;

                // If the elevation at the current point is invalid, we skip it and
                // continue. Given this condition we assume the path to not be blocked
                // since we cannot prove otherwise.
                if (pointElevation == ELEV_NO_DATA) {
                        continue;
                }

                // If the elevation at the current point exceeds the elevation of the ray path
                // then the ray is blocked.
                if (pointElevation > rayElevation) {
                        isBlocked = true;
                        break;
                }

                // If the elevation of the ray path at the current radius exceeds
                // the maximum dataset elevation then we can safely assume that the
                // ray is not blocked.
                if (rayElevation > this.maxElevationValue) {
                        isBlocked = false;
                        break;
                }

        }


        return isBlocked;

    }

    // Getters + Setters

}
```

7 Resources

jDem846 – http://code.google.com/p/jdem846

DEM data courtesy of the U.S. Geological Survey.

- USGS: http://www.usgs.gov/
- USGS The National Map Viewer: http://viewer.nationalmap.gov/viewer/

NOAA Solar Calculator: http://www.esrl.noaa.gov/gmd/grad/solcalc/

8 References

[1] Wikipedia Contributors. Digital Elevation Model. Retrieved March 19, 2012, from <http://en.wikipedia.org/wiki/Digital_elevation_model>.

---

* **KEVIN M. GILL** has been a Senior Software Engineer with Fidelity Investments for 8 years. Before that he was a stock trader with Fidelity and a computer technician and rifle instructor in the U.S. Marine Corps. He is a 2011 graduate of Rivier College with a M.S. in Computer Science and a 2009 graduate of the University of Massachusetts Lowell with a B.S. in Information Technology. He resides in Nashua, NH with his wife, a 5 year old child and 9 month old twins. He currently maintains the open source GIS project, jDem846.