Chapter 13
Case Study: Supporting Multiple Tile Clients

Chapter 9 presented techniques for serving tiled images according to our simple tile protocol and scheme. However, most mapping tools do not support this protocol by default. The purpose of this chapter is to present specialized techniques for tile serving that will support a wide variety of mapping tools. We will build an interface for Google Earth using the Keyhole Markup Language (KML). We will also build an Open Geospatial Consortium (OGC) Web Map Service (WMS) server.

13.1 KML Server

The KML language is a fairly expressive display control language for 3-D globe-oriented mapping tools like Google Earth. It allows for both dynamic positioning of the globe view and dynamic placement of mapping objects on the globe. Mapping objects can include vector features or image overlays. We will be using its image overlay capability to display tiled images on Google Earth.

13.1.1 Static KML Example

Our first example will be a statically generated KML file that links to tiled images from the local computer. The KML file is generated ahead of time and its contents do not change. The KML file in Listing 13.1 creates GroundOverlay objects corresponding to tiles from our Blue Marble tile set, zoom level 2.

Because we have provided the bounding boxes for each tiled image, Google Earth can draw the images on the globe in the correct position. Figure 13.1 shows a screenshot of the static KML file loaded into Google Earth. The KML file requires the image files be stored with it in the same folder. By replacing the file names in the above KML document with URLs we can force Google Earth to retrieve the images not from local disk but directly from a server. We can form URLs using the

J.T. Sample and E. Ioup, *Tile-Based Geospatial Information Systems: Principles and Practices*, DOI 10.1007/978-1-4419-7631-4_13,

method presented Section 9.2. Because KML is written in XML, we have to write the ampersand character & as `&` in our URL. Instead of:

```
<Icon>
    <href>2-0-2.jpg</href>
</Icon>
```

we would put:

```
<Icon>
    <href>http://www.sometileserver.com/tiles?REQUEST=GETTILE&amp;LAYER=
        BlueMarble&amp;LEVEL=2&amp;ROW=0&amp;COLUMN=2</href>
</Icon>
```

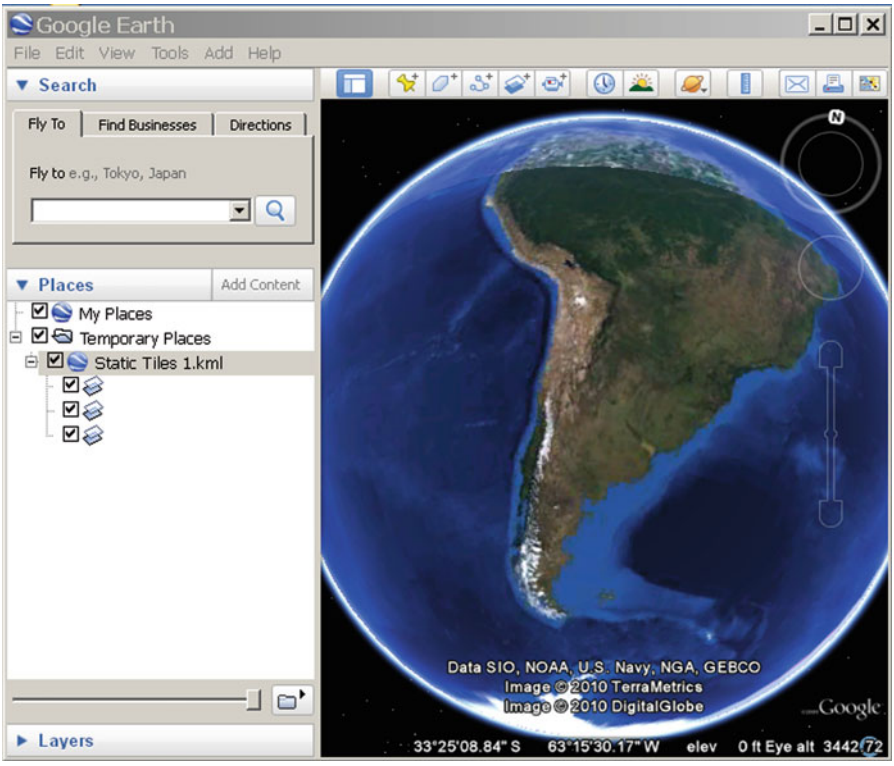

Fig. 13.1 Static KML loaded into Google Earth

This example is effective for loading images into Google Earth, but as the user zooms in or moves the globe around, the tiled images will not change. It would be better to create a method that could dynamically load and position tiled images as the user moves the globe.

13.1.2 Dynamic KML Example

Fortunately, Google Earth allows dynamic content generation based on communication between the client and the server. Using the `<NetworkLink>` KML tag, we can tell Google Earth to load tiled images based on the current globe position, and we can also tell Google Earth to refresh the content each time the globe moves. The following KML snippet creates a NetworkLink object named "bluemarble" and points it to the URL: `http://www.sometileserver.com/tiles/kml`. It tells Google Earth to refresh the network link each time the globe is moved and stops moving.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
    <NetworkLink>
      <name>bluemarble</name>
      <Link id="ID">
        <href>http://www.sometileserver.com/tiles/kml</href>
        <refreshMode>onChange</refreshMode>
        <refreshInterval>0</refreshInterval>
        <viewRefreshMode>onStop</viewRefreshMode>
        <viewRefreshTime>0</viewRefreshTime>
        <viewBoundScale>1</viewBoundScale>
        <viewFormat>BBOX=[bboxWest],[bboxSouth],[bboxEast],[bboxNorth]&amp;
            HEIGHT=[horizPixels]&amp;WIDTH=[vertPixels]</viewFormat>
      </Link>
    </NetworkLink>
</kml>
```

The values `bboxWest`, `bboxSouth`, `bboxEast`, `bboxNorth`, `horizPixels`, and `vertPixels` are built-in Google Earth parameters. The `<viewFormat>` tag tells Google Earth to append these values to the base URL for the NetworkLink. When activated, it will take the base URL `http://www.sometileserver.com/tiles/kml` and add parameters in the following manner:

```
http://www.sometileserver.com/tiles/kml?BBOX=-176.5,
27.57,-42.23,49.73&HEIGHT=1315&WIDTH=1154
```

We can then use these parameters to generate a dynamic version of the KML file presented in the previous section. The code in Listing 13.2 is a simple Java servlet that will dynamically generate a KML document based on the view parameters provided.

13.2 WMS Server

While KML is very effective for integrating mapping content with Google Earth, we want a more general solution to provide tile content to a larger set of mapping clients. The Open Geospatial Consortium (OGC) publishes a set of specifications for standardized communication between clients and servers. The Web Map Service

(WMS) standard provides for simple HTTP based access to imagery and rendered map images [1].

WMS is a simple protocol but is exceptionally powerful. It is supported by many mapping applications include Gaia by The Carbon Project, Google Earth, uDig, and OpenLayers. In general, WMS sessions are a two step process. The first step issues a ”GetCapabilities” request to get a list of available layers from a WMS server. The second (and subsequent steps) issues a ”GetMap” request to retrieve a map image. GetCapabilities requests are executed by forming a URL in the following manner:

```
http://www.sometileserver.com/wms?
REQUEST=GetCapabilities&VERSION=1.1.1&SERVICE=WMS
```

This request will return an XML document as shown in Listing 13.3. This document provides information about the service provider, methods of access and a list of available layers. Each available map layer is given with information about its map bounds and geospatial projection. Client applications can request map images by forming URLs in the following manner:

```
http://www.sometileserver.com/wms?REQUEST=GetMap&
SERVICE=WMS&VERSION=1.1.1&LAYERS=bluemarble&
FORMAT=image/png&SRS=EPSG:4326&BBOX=-130.04,-9.20,-52.
16,54.75&WIDTH=1225&HEIGHT=1006
```

For our Blue Marble imagery, the URL above should return the image in Figure 13.2.

13.2.1 WMS Servlet Implementation

To create a basic WMS implementation, we need to support only the GetCapabilites and GetMap requests. For our example, we can return a simple XML document for the GetCapabilities request. Implementing the GetMap request will require a little more effort. From the GetMap request URL, we can see the following parameters:

```
BBOX=-130.04,-9.20,-52.16,54.75

WIDTH=1225

HEIGHT=1006
```

These parameters illustrate how WMS differs from our previous tiled mapping examples. Our tiled images are created to match a sequence of fixed, discrete scales. In contrast, WMS requests allow users to select maps of varied sizes and resolutions that allow viewing of continuous scales.

Fig. 13.2 Bluemarbe image resulting from a WMS request.

To use our tile-based data sets to provide a proper GetMap response to this query, we must do two things. First, we need to take the provided bounding box and image dimensions and determine which tiles should be used to create the output image. Second, we have to merge the tiled images into a single composite image. This process is very similar to the one described in Section 3.1.2.

First, we select the scale that we will use. This is done by computing the degrees per pixel for the requested image and finding the closest tile zoom level (Equation (3.3)). Recall that for our tile scheme, each zoom level has its own uniform DPP value. Next, we compute the tiles needed to generate the map (Equation (2.8)). Given the list of tiled images, we can retrieve those images and draw them together in the same image. Equation (2.3) shows how to compute the geographic bounds for each, and Listing 4.10 shows how to draw the tiled images into a single image.

The code in Listing 13.4 shows a simple Java Servlet implementation of a Web Map Service that draws its imagery from a store of tiled images. The example code is intended to support version 1.1.1 of the WMS specification. A production WMS server should implement all versions of the WMS specification. Our example does not validate the query parameters or provide error handling. It also uses a completely static ”Capabilities” document; a production version would dynamically generate

the Capabilities document to reflect the names and bounds of available layers, which might change.

Another feature of the WMS specification is the ”GetFeatureInfo” operation. This allows WMS clients to query map layers for information at a point on the map. GetFeatureInfo requests pass the original coordinates and size of the requested map to the server and the x and y coordinates representing a mouse click. Responses to the GetFeatureInfo query can come in many forms, but will usually provide some text information on map content at the location queried. Developers of tile-based systems might use the GetFeatureInfo operation to provide users with the names of the source files used to make the tile at the location queried.

Listing 13.1 KML file for static inclusion of tiled images.

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <kml xmlns="http://earth.google.com/kml/2.1">
4      <Document>
5          <GroundOverlay>
6              <drawOrder>21</drawOrder>
7              <Icon>
8                  <href>2-0-0.jpg</href>
9              </Icon>
10             <LatLonBox>
11                 <north>0.0</north>
12                 <south>-90.0</south>
13                 <east>-90.0</east>
14                 <west>-180.0</west>
15             </LatLonBox>
16         </GroundOverlay>
17         <GroundOverlay>
18             <drawOrder>21</drawOrder>
19             <Icon>
20                 <href>2-0-1.jpg</href>
21             </Icon>
22             <LatLonBox>
23                 <north>0.0</north>
24                 <south>-90.0</south>
25                 <east>0.0</east>
26                 <west>-90.0</west>
27             </LatLonBox>
28         </GroundOverlay>
29         <GroundOverlay>
30             <drawOrder>21</drawOrder>
31             <Icon>
32                 <href>2-0-2.jpg</href>
33             </Icon>
34             <LatLonBox>
35                 <north>0.0</north>
36                 <south>-90.0</south>
37                 <east>90.0</east>
38                 <west>0.0</west>
39             </LatLonBox>
40         </GroundOverlay>
41
42     </Document>
43 </kml>
```

Listing 13.2 Servlet for dynamic XML content.

```
1  public class KMLServlet extends HttpServlet {
2
3      DataStore dataStore;
4
5      public void doGet(HttpServletRequest request, HttpServletResponse response)
           {
6          String layer = "bluemarble";
7          //set the content type for KML
8          response.setContentType("application/keyhole");
9
10         //get the view format parameters from the query
11         int width = Integer.parseInt(request.getParameter("WIDTH"));
12         int height = Integer.parseInt(request.getParameter("HEIGHT"));
13         String bbox = request.getParameter("BBOX");
14         String[] bboxdata = bbox.split(",");
15         double minx = Double.parseDouble(bboxdata[0]);
16         double maxx = Double.parseDouble(bboxdata[2]);
17         double miny = Double.parseDouble(bboxdata[1]);
18         double maxy = Double.parseDouble(bboxdata[3]);
19
20         //this section determines which zoom level to use
21         double currentDPP = 0.8 * (maxx - minx) / width;
22         double[] zoomLevels = TileStandards.zoomLevels;
23         int scale = 18;
24         for (int i = 2; i < zoomLevels.length; i++) {
25             if (zoomLevels[i] < currentDPP) {
26                 scale = i;
27                 break;
28             }
29         }
30
31         int minscale = dataStore.getMinScale(layer);
32         int maxscale = dataStore.getMaxScale(layer);
33
34         if (scale > maxscale) {
35             scale = maxscale;
36         }
37         if (scale < minscale) {
38             scale = minscale;
39         }
40         //determine the range of tiles to use in the response
41         long startRow = (long) Math.floor(TileStandards.getRowForCoord(miny,
               scale));
42         long endRow = (long) Math.floor(TileStandards.getRowForCoord(maxy,
               scale));
43         long startCol = (long) Math.floor(TileStandards.getColForCoord(minx,
               scale));
44         long endCol = (long) Math.floor(TileStandards.getColForCoord(maxx,
               scale));
45
46         //get the output writer
47         PrintWriter pw=null;
48         try {
49             pw = response.getWriter();
50         } catch (IOException e) {
51             e.printStackTrace();
52         }
53
54         //print document header
55         pw.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
56                    + "<kml xmlns=\"http://earth.google.com/kml/2.1\">"
57                    + "<Document>");
58
59         double degsPerTile = getDegreesPerTile(scale);
60         for (long i = startRow; i <= endRow; i++) {
61             for (long j = startCol; j <= endCol; j++) {
```

```
62                  double tminx = j * degsPerTile - 180.0;
63                  double tmaxx = (1 + j) * degsPerTile - 180.0;
64                  double tminy = i * degsPerTile - 90.0;
65                  double tmaxy = (1 + i) * degsPerTile - 90.0;
66                  pw
67                     .println("<GroundOverlay>"
68                              + "<drawOrder>21</drawOrder>"
69                              + "<Icon>"
70                              + " <href>http://www.sometileserver.com/tiles&amp;
                                    REQUEST=GETTILE&amp;"
71                              + "layer=bluemarble&amp;" + "row=" + i + "&amp;col="
                                    + j
72                              + "&amp;level=" + scale + "</href>"
73                              + "</Icon>"
74                              + "<LatLonBox>"
75                              + " <north>"+ tmaxy + "</north>"
76                              + "<south>" + tminy + "</south>"
77                              + "<east>"  + tmaxx + "</east>"
78                              + "<west>"  + tminx + "</west>"
79                              + "</LatLonBox>"
80                              + "</GroundOverlay>");
81              }
82          }
83          //print document footer
84          pw.println("</Document></kml>");
85
86      }
87
88      private double getDegreesPerTile(int scale) {
89          double degs = 360.0 / (Math.pow(2, scale));
90          return degs;
91      }
92  }
```

Listing 13.3 Example WMS Capabilities document.

```
1  <WMT_MS_Capabilities version="1.1.1" updateSequence="0">
2      <Service>
3          <Name>WMS</Name>
4          <Title>Tile_Server</Title>
5          <Abstract>
6          </Abstract>
7          <KeywordList>
8              <Keyword>Some Keywords</Keyword>
9          </KeywordList>
10         <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="
               simple" xlink:href="http://www.sometileserver.com/" />
11         <ContactInformation>
12             <ContactPersonPrimary>
13                 <ContactPerson>John Q. Developer</ContactPerson>
14                 <ContactOrganization>Some Company</ContactOrganization>
15             </ContactPersonPrimary>
16             <ContactPosition />
17             <ContactAddress>
18                 <AddressType>postal</AddressType>
19                 <Address>123 Easy Street</Address>
20                 <City>Sometown</City>
21                 <StateOrProvince>BG</StateOrProvince>
22                 <PostCode>31111</PostCode>
23                 <Country>USA</Country>
24             </ContactAddress>
25             <ContactVoiceTelephone>+1 321 456-4200</ContactVoiceTelephone>
26             <ContactFacsimileTelephone>+1 321 456-4443</
                   ContactFacsimileTelephone>
27             <ContactElectronicMailAddress>developer@somecompany.com</
                   ContactElectronicMailAddress>
28         </ContactInformation>
```

```
29          <Fees>none</Fees>
30          <AccessConstraints>none</AccessConstraints>
31      </Service>
32      <Capability>
33          <Request>
34              <GetCapabilities>
35                  <Format>application/vnd.ogc.wms_xml</Format>
36                  <DCPType>
37                      <HTTP>
38                          <Get>
39                              <OnlineResource xmlns:xlink="http://www.w3.org
                                    /1999/xlink" xlink:type="simple" xlink:href="
                                    http://www.sometileserver.com/wms?" />
40                          </Get>
41                      </HTTP>
42                  </DCPType>
43              </GetCapabilities>
44              <GetMap>
45                  <Format>image/png</Format>
46                  <Format>image/jpeg</Format>
47                  <DCPType>
48                      <HTTP>
49                          <Get>
50                              <OnlineResource xmlns:xlink="http://www.w3.org
                                    /1999/xlink" xlink:type="simple" xlink:href="
                                    http://www.sometileserver.com/wms?" />
51                          </Get>
52                      </HTTP>
53                  </DCPType>
54              </GetMap>
55          </Request>
56          <Exception>
57              <Format>application/vnd.ogc.se_xml</Format>
58          </Exception>
59          <VendorSpecificCapabilities />
60          <Layer>
61              <Title>Tile_Server</Title>
62              <Layer queryable="1">
63                  <Name>bluemarble</Name>
64                  <Title>bluemarble</Title>
65                  <SRS>EPSG:4326</SRS>
66                  <LatLonBoundingBox SRS="EPSG:4326" minx="-180.0" miny="-90.0"
                        maxx="180.0" maxy="90.0" />
67                  <BoundingBox SRS="EPSG:4326" minx="-180.0" miny="-90.0" maxx="
                        180.0" maxy="90.0" />
68              </Layer>
69          </Layer>
70      </Capability>
71  </WMT_MS_Capabilities>
```

Listing 13.4 Java Servlet implementation of simple WMS server.

```
public class WMSTileServlet extends HttpServlet {

    DataStore tileStorage;

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        {
        //collect the parameters from the URL
        String service = request.getParameter("SERVICE");
        //service should equal WMS
        String version = request.getParameter("VERSION");
        //version should equal 1.1.1

        String requestType = request.getParameter("REQUEST");
        if (requestType.equalsIgnoreCase("GetCapabilities")) {
            printCapabilities(request, response);
        }

        if (requestType.equalsIgnoreCase("GETMAP")) {
            String layers = request.getParameter("LAYERS");
            //layers should equal blue marble
            String srs = request.getParameter("SRS");
            //should be equal to EPSG:4326

            String widthStr = request.getParameter("WIDTH");
            String heightStr = request.getParameter("HEIGHT");
            int width = Integer.parseInt(widthStr);
            int height = Integer.parseInt(heightStr);
            String format = request.getParameter("FORMAT");
            //get bounding box from request
            String bbox = request.getParameter("BBOX");
            String bbdata[] = bbox.split(",");
            double minx = Double.parseDouble(bbdata[0]);
            double miny = Double.parseDouble(bbdata[1]);
            double maxx = Double.parseDouble(bbdata[2]);
            double maxy = Double.parseDouble(bbdata[3]);
            BoundingBox imageBounds = new BoundingBox(minx, miny, maxx, maxy);
            //compute scale to use
            double dpp = ((maxx - minx) / width + (maxy - miny) / height) /
                2.0;
            double[] standardScales = TileStandards.zoomLevels;
            int maxscale = tileStorage.getMaxScale("bluemarble");

            int scaleToUse = maxscale;
            for (int i = 0; i &lt; maxscale; i++) {
                if (standardScales[i] &lt; dpp) {
                    scaleToUse = i;
                    break;
                }
            }
            double tileSize = 360.0 / (Math.pow(2, scaleToUse));
            //calculate bounds of image in tile coordinates
            long mincol = (long) Math.max(0, Math.floor((minx + 180.0) /
                tileSize));
            long maxcol = (long) Math.floor((maxx + 180.0) / tileSize);
            long minrow = (long) Math.max(0, Math.floor((miny + 90.0) /
                tileSize));
            long maxrow = (long) Math.floor((maxy + 90.0) / tileSize);

            //iterate over tile range, retrieve each tile and draw to new image
            BufferedImage bi = new BufferedImage(width, height,
            BufferedImage.TYPE_INT_RGB);
            for (long r = minrow; r &lt;= maxrow; r++) {
                for (long c = mincol; c &lt;= maxcol; c++) {
                    //have to check to see if the cluster actually has any data
                        ...
                    TileAddress tileAddress = new TileAddress(r, c, scaleToUse)
                        ;

                    BoundingBox tileBounds = tileAddress.getBoundingBox();
                    byte[] imageData = tileStorage.getTileImage("bluemarble",
                    scaleToUse, r, c);

```

```
67                      if (imageData != null) {
68                          try {
69                              BufferedImage tileImage = ImageIO.read(new
                                    ByteArrayInputStream(imageData));
70                              drawImageToImage(tileImage, tileBounds, bi,
                                    imageBounds);
71                              } catch (IOException e) {
72                              e.printStackTrace();
73                          }
74                      }
75                  }
76              }
77
78              //encode image and write to client
79              try {
80                  ByteArrayOutputStream baos = new ByteArrayOutputStream();
81                  if (format.matches(".*png.*")) {
82                      ImageIO.write(bi, "png", baos);
83                      response.setContentType("image/png");
84                      } else {
85                      ImageIO.write(bi, "jpg", baos);
86                      response.setContentType("image/jpg");
87                  }
88                  ServletOutputStream os = response.getOutputStream();
89                  os.write(baos.toByteArray());
90                  os.close();
91                  } catch (IOException e) {
92                  e.printStackTrace();
93              }
94          }
95      }
96
97      public static void drawImageToImage(BufferedImage source,BoundingBox
            source_bb, BufferedImage target, BoundingBox target_bb) {
98          double xd = target_bb.maxx - target_bb.minx;
99          double yd = target_bb.maxy - target_bb.miny;
100         double wd = (double) target.getWidth();
101         double hd = (double) target.getHeight();
102         double targdpx = xd / wd;
103         double targdpy = yd / hd;
104         double srcdpx = (source_bb.maxx - source_bb.minx) / source.getWidth();
105         double srcdpy = (source_bb.maxy - source_bb.miny) / source.getHeight();
106         int tx = (int) Math.round(((source_bb.minx - target_bb.minx) / targdpx)
                );
107         int ty = target.getHeight()
108         - (int) Math.round(((source_bb.maxy - target_bb.miny) / yd) * hd)
109         - 1;
110         int tw = (int) Math.ceil(((srcdpx / targdpx) * source.getWidth()));
111         int th = (int) Math.ceil(((srcdpy / targdpy) * source.getHeight()));
112         Graphics2D target_graphics = (Graphics2D) target.getGraphics();
113
114         target_graphics
115         .setRenderingHint(RenderingHints.KEY_INTERPOLATION,
116         RenderingHints.VALUE_INTERPOLATION_BILINEAR);
117
118         target_graphics.drawImage(source, tx, ty, tw, th, null);
119     }
120
121     private void printCapabilities(HttpServletRequest request,
            HttpServletResponse response) {
122         try {
123             response.setContentType("text/xml");
124             PrintWriter pw = response.getWriter();
125             pw.print(capabilitiesContents);
126             pw.close();
127             } catch (IOException e) {
128             e.printStackTrace();
```

```
129         }
130     }
131
132     public static String capabilitiesFile = "capabilities.xml";
133     public static String capabilitiesContents;
134     //read the capabilities file into memory and hold it
135     static {
136         StringBuffer sb = new StringBuffer();
137         try {
138             RandomAccessFile raf = new RandomAccessFile(capabilitiesFile, "r");
139             byte[] data;
140             data = new byte[(int) raf.length()];
141             raf.readFully(data);
142             raf.close();
143             capabilitiesContents = new String(data);
144             } catch (IOException e) {
145             e.printStackTrace();
146         }
147     }
148 }
```

References

1. de La Beaujardière, J.: Web Map Service Implementation Specification. Open Geospatial Consortium Specification pp. 06–042 (2006)