Location-aware Computing

Geo-Data Formats

http://www.cs.unibo.it/~difelice/

Context Aware Systems

Prof. Marco Di Felice
Department of Computer Science and Engineering
University of Bologna
APIs for Location-based services

We will consider data formats for Location-based Service and Content Providers

COMPONENTS of LOCATION-BASED SERVICES ARCHITECTURE

Source: http://geoawesomeness.com/
GIS Definition

“A computer system capable of assembling, storing, manipulating, and displaying geographically referenced information”. (United States Geological Survey (USGS 2005)).

✧ GIS can be considered a specific instance of information systems for location-based features.

✧ GIS can support several different technologies, processes, techniques and methods, related to: Spatial Data Capture, Spatial Data Storage and Management, Spatial and Attribute Analysis, Visualization.
GIS Definition

GIS Functionalities

- Spatial Data Capture
- Spatial Data Storage and Management
- Spatial and Attribute Analysis
- Visualization

Source: http://flash.lakeheadu.ca/~forspatial/
Spatial Data
GIS Data Formats

- **Spatial data** stores the *geographic location* of particular features, along with information describing what these features represent.
GIS Data Formats

✧ **Raster data** ➔ regular grid of cells divided into rows and columns. Each cell describes a portion of the geographic location.

✧ Data values for a given parameter are stored in each cell: values are application-dependant (e.g. elevation from the sea, density of the population, etc).

✧ The spatial resolution of the raster data set is determined by the size of the cells.
GIS Data Formats

✧ Representing data changing continuously over a landscape

✧ Combining data from multiple sources and classifying them

SURFACE MAPS

THEMATIC MAPS

Raster Image (BaseMaps)

GIS Data Formats

Raster Data: Advantages

を持っている

1. Relatively simple data structure
2. Good support for spatial data analytics (e.g. compare the value of a cell with other cells in its neighborhood and compute the max)
3. Easy to store continuously changing values (e.g. elevation)
GIS Data Formats

Raster Data: Disadvantages

✧ Spatial accuracy depends on the cell resolution.
✧ Processing and storage might be cumbersome in presence of large amount of data.
✧ It might be difficult to edit or update geographic features.

GIS Data Formats

**Raster Data Formats: Some Examples**

- GeoTIFF
- JPEG2000
- ESRI GRID
- ...

Standard .tiff file including additional spatial (georeferencing) information embedded in the .tiff file as tags.

GIS Data Formats

✧ **Vector data** → geographic features are represented as **discrete geometries**, specifically: points, lines and polygons.

✧ **Point** → X,Y location
✧ **Line** → One-dimensional feature with a starting and an ending point
✧ **Polygon** → Enclosed area, a two-dimensional feature with at least three sides (and therefore with an area).
✧ Each geometry can be associated to user-defined **features**.
GIS Data Formats

✧ **Vector data** ➔ geographic features are represented as discrete geometries, specifically: points, lines and polygons.
GIS Data Formats

✧ **Topology** → way to define *spatial relationships* between vector (geographical) data.

**ARC-NODE DATA STRUCTURE**

✧ **Ingredients**: nodes, arcs and vertices

✧ **Nodes** define the two endpoints of an arc; they may or may not connect two or more arcs.

✧ An **arc** is the line segment between two nodes. An arc is composed of its two nodes and an ordered series of points which define its shape, called **vertices**.

GIS Data Formats

凫 Topology ➔ way to define spatial relationships between vector (geographical) data.

GIS Data Formats

Vector Data: Advantages

✧ High resolution, high spatial accuracy
✧ Easy editing, update, and management of the spatial features; possibility to represent natural shapes
✧ Much higher data storage efficiency than raster data

GIS Data Formats

Vector Data: Disadvantages

✧ Vector data formats may be more difficult to manage than raster formats.
✧ Easy for computers to understand, but not easily managed by end-users.
✧ While “low-end” computers can operate raster-based GISs, vector formats require more powerful or high-tech machines.

GIS Data Formats

**Vector Data Formats: Some Examples**

- GeoJSON
- KML
- SHP
- GML
- …

GEOJSON

- Format for encoding a variety of geographic data structures, based on JSON (JavaScript Object Notation).
- Released in 2005 by the Internet Engineering Task Force
- Supported by a variety of GIS platforms and Mapping APIs (e.g. Google Maps v3, OpenLayers)
- TopoJSON → GEOJSON extension for topology encoding
- Current specifications (RFC 7946)

GEOJSON

- Javascript Object Notation (JSON)
  - A JSON file is also called a “Document”.
  - JSON document can be easily parsed by machines.
  - Favour system integration and interoperability among third-party software components.
    - A JSON document is surrounded by brackets `{}`
    - Each data entry is a `<key, value>` couple.

```json
{ givenname: "mario" }
{ givenname: "mario", lastname: "rossi"}
```
GEOJSON

```json
{  
givenname: "Mario",
lastname: "Rossi",
age: 45,
employed: true,
salary: 1200.00,
phones: ["0243434", "064334343"],
office: [
  {name: "A", street: Zamboni, number: 7},
  {name: "B", street: Irnerio, number: 49}
]
}
```

JSON Document
GEOJSON Elements

- **Coordinates**
  - Single number representing a single dimension
  - Example of dimension: the latitude
  - Represented in single decimal format

- **Position**
  - Array of coordinates
  - Order: [longitude; latitude; elevation]
GEOJSON Elements

- **Geometry**
  - Define a shape
  - Single-Geometry vs Multi-Geometry
  - Single-Geometry consists of a type + coordinates

- **Points** (no internal area)

  ```json
  { "type": "Point", "coordinates": [0, 0] }
  ```

- **LineStrings** (no internal area)

  ```json
  { "type": "LineString", "coordinates": [[0, 0], [10, 10]] }
  ```

GEOJSON Tutorial: https://macwright.org/2015/03/23/geojson-second-bite#what-you-can't-do-with-geojson
GEOJSON Elements

- **Polygons** (internal area, can also have cut-outs)

```
{
  "type": "Polygon",
  "coordinates": [
    [[30, 10], [40, 40], [20, 40],
     [10, 20], [30, 10]]
  ]
}
```

```
{
  "type": "Polygon",
  "coordinates": [
    [[35, 10], [45, 45], [15, 40],
     [10, 20], [35, 10]],
    [[20, 30], [35, 35], [30, 20],
     [20, 30]]
  ]
}
```
GEOJSON Elements

- Multi-Point and MultiLineString
  - Array of simple geometry elements

```
{
  "type": "MultiPoint",
  "coordinates": [
    [10, 40], [40, 30], [20, 20], [30, 10]
  ]
}
```

```
{
  "type": "MultiLineString",
  "coordinates": [
    [[10, 10], [20, 20], [10, 40]],
    [[40, 40], [30, 30], [40, 20], [30, 10]]
  ]
}
```
GEOJSON Elements

- **Multi-Polygon**
  - **Array of simple geometry elements**

```json
{
    "type": "MultiPolygon",
    "coordinates": [
        [
            [[30, 20], [45, 40], [10, 40], [30, 20]]
        ],
        [
            [[15, 5], [40, 10], [10, 20], [5, 10], [15, 5]]
        ]
    }
}
```
GEOJSON Elements

- **Multi-Geometry**
  - Combination of different geometries

```json
{
  "type": "Feature",
  "geometry": {
    "type": "GeometryCollection",
    "geometries": [
      {
        "type": "Point",
        "coordinates": [0, 0]
      },
      {
        "type": "LineString",
        "coordinates": [[0, 0], [1, 0]]
      }
    ]
  },
  "properties": {
    "name": "null island"
  }
}
```

Shape composed of:
- 1 Point
- 1 LineString
GEOJSON Elements

- **Features**
  - Combination of *geometry + properties*
  - Property ➔ any kind of JSON document

```json
{
    "type": "Feature",
    "geometry": {
        "type": "Point",
        "coordinates": [0, 0]
    },
    "properties": {
        "name": "My Point of Interest"
    }
}
```

Any kind of key/value couple
GEOJSON Elements

- **FeatureCollection**
  - Array of Features, top level of a GEOJSON file

```json
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [0, 0]
      },
      "properties": {
        "name": "My Point"
      }
    }
  ]
}
```
TopoJSON

✧ **Extension** of GeoJSON encoding topologies
✧ More efficient rendering and storage (up to 80% data reduction).
✧ Avoid duplicates of features (e.g. a boundary)
✧ Transform coordinates from real to integer
✧ It is based on the arc-node data structure
✧ It introduces a new type (**Topology**) that contains a GeoJSON Object.
TopoJSON

- A topology contains an **Object** field:
  - **Name** → unique id
  - **Type** → standard GeoJSON geometry (e.g. Polygon, LineString)
  - Differently from GeoJSON, coordinates are not stored in the topology, but in arcs array
  - An **arc** is a sequence of points, like a lineString
TopoJSON

```
{
  "type": "Topology",
  "objects": {
    "island": {
      "type": "Polygon",
      "arcs": [[[0]],
                "id": 533
              ]
    },
    "arcs": [
      [[3058, 5901], [0, -2], [-2, 1], [-1, 3], [-2, 3], [0, 3], [1, 1], [1, -3], [2, -5], [1, -1]]
    ]
  }
}
```

- Coordinates of the geometry
- Array of arrays of arcs

TOPOJSON Tutorial: https://github.com/topojson/topojson/wiki
TopoJSON

```
{
  "type": "Topology",
  "objects": {
    "path": {
      "type": "LineString",
      "arcs": [[0,1]],
      "id": 533
    }
  },
  "arcs": [
    [[3058, 5901],[0,-1],[1,2]],
    [[3059,5903],[0,-2],[2,4],[-1,-3]]
  ]
}
```

Array of arrays of arcs
0: A → B → C
1: C → D → E → F
Path: A→B→C→D→E→F
TopoJSON

```json
{
  "type": "Topology",
  "transform": {
    "scale": [0.036003600360036005, 0.017361589674592462],
    "translate": [-180, -89.99892578124998]
  }
},
"objects": {
  "island": {
    "type": "Polygon",
    "arcs": [[0]],
    "id": 533
  }
},
"arcs": [
  [[3058, 5901], [0, -2], [-2, 1]]
]
}
```

**Delta-encoding:**

\[
\begin{align*}
X_1(A) &= X_1(R) \times \text{ScaleX} + \text{TranslateX} \\
Y_1(A) &= Y_1(R) \times \text{ScaleY} + \text{TranslateY} \\
X_2(A) &= X_1(A) + \Delta X \\
Y_2(A) &= Y_1(A) + \Delta Y
\end{align*}
\]
GIS Data Formats

Vector Data Formats: Some Examples

- GeoJSON
- KML
- SHP
- GML
- …
Keyhole Markup Language

- Format for encoding a variety of geographic data structures (alternative to GEOJSON)
  - Originally proposed for use with Google Earth
  - Standard of the Open Geospatial Consortium since 2009
  - 3D geographic coordinates: longitude, latitude and altitude
  - Allows modeling time-intervals and animate a model over time
  - XML tag-based structure

https://developers.google.com/kml/documentation/
Keyhole Markup Language

- Placemark ➔ position on the Earth's surface, denoted with a yellow pushpin (by default)
  - User can specify name, HTML description, icon

```xml
<?xml version="1.0" encoding="UTF-8"?><kml xmlns="http://www.opengis.net/kml/2.2"><Placemark><name>Simple placemark</name><description>Department of Computer Science and Engineering.</description><Point><coordinates>11.3472152,44.4972641,0</coordinates></Point></Placemark></kml>
```
Keyhole Markup Language

- **GroundOverlay** ➔ add an image onto the Earth's terrain

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Folder>
    <name>Ground Overlays</name>
    <description>Examples of ground overlays</description>
    <GroundOverlay>
      <name>Large-scale overlay on terrain</name>
      <description>Overlay shows Mount Etna erupting on July 13th, 2001.</description>
      <Icon>
        <href>https://developers.google.com/kml/documentation/images/etna.jpg</href>
      </Icon>
      <LatLonBox>
        <north>37.91904192681665</north>
        <south>37.46543388598137</south>
        <east>15.35832653742206</east>
        <west>14.60128369746704</west>
        <rotation>-0.1556640799496235</rotation>
      </LatLonBox>
    </GroundOverlay>
  </Folder>
</kml>
```
Keyhole Markup Language

- **Paths** ➔ line connecting Points, `<LineString>` tag
- **Polygon** ➔ simple building, `<Polygons>` tag
- **Style** ➔ define a proper Style for a Vector (e.g. Polygon)
- **Network link** ➔ a `<Link>` element with an `<href>` that loads a file (image, icon, KML file)
- **CGI Scripting** ➔ a `<Link>` element with an `<href>` point to data that is dynamically generated by a CGI script (e.g. PHP, Python, Perl)
Keyhole Markup Language

TOURING MODE

A tour is composed of a list of sequential elements:

```xml
<gx:FlyTo>
<gx:Wait>
<gx:TourControl>
```

https://developers.google.com/kml/documentation/touring
A tour is composed of a list of sequential elements:

```xml
<gx:FlyTo>
  <gx:duration>4.0</gx:duration>
  <gx:flyToMode>smooth</gx:flyToMode>
  <LookAt>
    <longitude>-121.303179</longitude>
    <latitude>45.993041</latitude>
    <altitude>0</altitude>
    <heading>-27.923387</heading>
    <tilt>71.600075</tilt>
    <range>22570.546801</range>
    <altitudeMode>relativeToGround</altitudeMode>
  </LookAt>
</gx:FlyTo>
```
Keyhole Markup Language

TOURING MODE

A tour is composed of a list of sequential elements:

```xml
<gx:FlyTo>
<gx:Wait>
<gx:TourControl>
  <gx:Wait>
    <gx:duration>10.0</gx:duration>
  <!-- wait time in seconds -->
</gx:Wait>
</gx:Wait>

<gx:SoundCue>
  <href>http://file.mp3</href>
</gx:SoundCue>
```

A tour is composed of a list of elements executed in parallel:

```xml
<gx:AnimatedUpdate>
  <gx:SoundCue>
    <href>http://file.mp3</href>
  </gx:SoundCue>
</gx:AnimatedUpdate>
```
GIS Data Formats

**Vector Data Formats: Some Examples**

- GeoJSON
- KML
- SHP
- GML
- …
ShapeFile

Geospatial data format (standard from ESRI)

- Vector data representation
- Differently from GeoJSON and KML, the spatial data are stored using a binary representation.

Each Shapefile document is composed of (at least) three files:

- .shp — the file contains the geometries (e.g. points, lines, polygons)
- .shx — shape index format; it speeds up the search operations into the shp file
- .dbf — attribute file; it stores the attributes/features for each shape
EXERCISE

- Download EXERCISE-JSON support files from here:
  
  https://site.unibo.it/iot/en/teaching-1/context-aware-systems

1. Create a GEO-JSON file corresponding to the boundaries of the old neighbourhoods of Bologna

2. Display the GEO-JSON file on a Webpage by properly completing the template.html source code