Contingut

Introducción

En el módulo anterior, hemos aprendido a extender un SIG de escritorio (gvSIG) para adaptarlo a nuestras necesidades. También hemos aprendido a agregar nuevas capas por código y elementos georeferenciados en ellas.

En este bloque aprenderemos a hacer prácticamente lo mismo pero en entornos web. Estudiaremos cómo construir una página web y cómo incorporarle un mapa. Aprenderemos también a agregar elementos georeferenciados al mapa que nos servirán para marcar puntos, trazar caminos o delimitar áreas.

Aunque hay varios servicios de mapas que operan en Internet, como Yahoo Maps o Bing Maps, este módulo se centrará en el estudio de Google Maps. Sin embargo, las técnicas que se explicarán serán fácilmente trasladables a los otros servicios.

Objetivos

Al final de este módulo, deberíamos ser capaces de:

- Crear páginas web simples en las que se muestre un mapa.
- Mostrar un tipo de mapa u otro dependiendo de nuestras necesidades.
- Agregar marcadores georreferenciados a un mapa para señalar lugares.
- Dibujar líneas y polígonos georreferenciados en un mapa para mostrar rutas o delimitar áreas.
- Convertir las coordenadas de un mapa a coordenadas geográficas y viceversa.

El lenguaje HTML

La Web y el HTML

La World Wide Web, que se abrevia WWW y se conoce también como la Web, es un sistema de páginas enlazadas mediante el concepto de hipertexto. El hipertexto es un texto que refiere a otro texto al que el lector puede tener acceso inmediato. El lenguaje tradicional y predominante sobre el que se sustenta la Web y que tiene capacidades de hipertexto es el HTML (*HyperText Markup Language*, "lenguaje de marcas de hipertexto").

Hasta hace relativamente poco era necesario conocer el lenguaje HTML para poder publicar una página web en Internet. Si bien el HTML es un lenguaje muy directo, en el que buena parte del código es el contenido propiamente dicho del documento, como contrapartida es engorroso de escribir. Así, por ejemplo, para mostrar una palabra en negrita, es necesario intercalar en el texto más de media docena de caracteres extras.

Hoy en día han aparecido muchísimas herramientas que permiten generar páginas web de la misma forma que editamos cualquier otro documento. De hecho, casi todos los procesadores de textos modernos ofrecen la posibilidad de guardar sus documentos en formato HTML. Además, hay herramientas altamente especializadas que generan páginas de altísima calidad en poco tiempo y con el mínimo esfuerzo, como por ejemplo Amaya, Bluefish o Dreamweaver.

Dada la existencia de estas herramientas de edición y puesto que el objetivo de estos materiales no es profundizar en el conocimiento del HTML, sólo daremos unas pocas pinceladas sobre este lenguaje, las necesarias para escribir páginas relativamente sencillas y para comprender los ejemplos que se trabajarán en el resto del módulo. Sin embargo, la UOC dispone de materiales abiertos que profundizan en el lenguaje HTML [1].

Elementos y etiquetas

El lenguaje HTML se utiliza para describir la estructura y el contenido de un documento, así como para complementar el texto con objetos (por ejemplo, mapas) e imágenes.

Un típico código HTML está formado por elementos que configuran las distintas partes del documento. Generalmente, estos elementos están delimitados por una etiqueta de inicio y otra de fin, que se escriben siempre entre corchetes angulares ("<" y ">").

Por ejemplo, la línea siguiente:

```
<p>La WWW nació en el CERN.</p>
```

define un elemento (párrafo) cuyo contenido es "La WWW nació en el CERN.". Como podemos ver, las etiquetas de inicio y fin (y respectivamente) delimitan el contenido del elemento. Es importante observar que la etiqueta de fin es igual que la de inicio pero precedida por una barra (?/?).

Hay elementos que sólo tienen una etiqueta de inicio, pues carecen de contenido textual. Es el caso, por ejemplo, de las imágenes. Las líneas siguientes:

```
<p>El primer logotipo de la WWW:</p>
<img src="http://upload.wikimedia.org/wikipedia/commons/thumb/b/b2/WWW_logo_by_Robert_Cailliau.svg
```

muestran un texto y la imagen que se encuentra en la dirección:
http://upload.wikimedia.org/wikipedia/commons/thumb/b/b2/WWW_logo_by_Robert_Cailliau.svg/120px-WWW_logo_b

El resultado será el siguiente:

El primer logotipo de la WWW:

Observad la estructura que aparece dentro de la etiqueta de inicio del elemento <img>:

```
src="http://upload.wikimedia.org/wikipedia/commons/thumb/b/b2/WWW_logo_by_Robert_Cailliau.svg/120p
```

Todos los elementos HTML tienen atributos que varían de un elemento a otro según sus características. Dichos atributos permiten variar la forma en la que se presenta el elemento. En el caso de las imágenes, tienen un atributo "src" (de *source*, "fuente") que indica la dirección en la que se encuentra la imagen. Como se puede apreciar en el ejemplo, el valor de los atributos siempre especifíca en la etiqueta de inicio.

Una última característica importante de los elementos HTML es que se pueden imbricar. Es decir, en su contenido un elemento puede contener otros elementos que, a su vez, pueden contener otros, y así sucesivamente, formando una jerarquía de elementos.

El código siguiente:

```
<p>WWW significa <em>World Wide Web</em>.</p>
```

define un párrafo que contiene una texto enfatizado (World Wide Web). Es importante observar que los elementos se cierran en orden inverso a aquel en el que se han abierto. El resultado será el siguiente:

WWW significa *World Wide Web*.

Estructura mínima de un documento HTML

Un documento HTML bien formado debe presentar siempre la siguiente estructura mínima:

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Título del documento</title>
</head>
<body>
Contenido del documento.
</body>
</html>
```

Como podemos observar, la estructura mínima de un documento HTML está formada por cinco elementos:

- **<DOCTYPE>**: Establece el tipo de documento (en el ejemplo, HTML, por defecto versión 5).
- **<html>**: Es el elemento raíz, el que define el documento HTML en sí. Contiene obligatoriamente los elementos <head> y <body>. Como puede apreciarse en el ejemplo, el elemento <html> puede contener el atributo "lang", que define el idioma del documento (?es? de español).
- **<head>**: Define la cabecera del documento, cuyo cometido es alojar información sobre el mismo. El título, las palabras clave y otros datos que no se consideran parte del contenido del documento irán dentro de este elemento.
- **<title>**: Define el título del documento.
- **<body>**: Define el contenido del documento. Este elemento contendrá otros que definirán el contenido del documento.

Definición del contenido

El contenido de un documento HTML puede definirse mediante tres tipos de elementos:

- **Estructurales**: Son aquellos que describen la estructura del texto. En HTML tenemos elementos estructurales para definir títulos, párrafos, tablas, enumeraciones, etc. Normalmente, los navegadores aplican diferentes estilos de presentación a cada elemento.
- **De presentación**: Son aquellos que describen el aspecto del texto, independientemente de su función. Los elementos que definen negritas, cursivas, texto enfatizado, etc., son elementos de presentación.
- **De hipertexto**: Son aquellos que permiten enlazar con otros documentos mediante la creación de hipervínculos. Son los más importantes dada la naturaleza hipertextual de la Web.

A continuación, se mostrarán algunos de los elementos más significativos de cada tipo. Además, ilustraremos su uso con ejemplos.

Elementos estructurales

Como ya se ha mencionado, los elementos estructurales son aquellos que describen el propósito del texto. Entre ellos podemos distinguir los siguientes:

- **<p>**, de *paragraph* (párrafo): Define un párrafo de texto. Las líneas siguientes:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>La WWW es un sistema de páginas enlazadas mediante el concepto de hipertexto.</p>
<p>La WWW nació en 1989 en el CERN, de la mano de Tim Berners-Lee.</p>
</body>
</html>
```

definen un par de párrafos. El resultado será el siguiente:

La WWW es un sistema de páginas enlazadas mediante el concepto de hipertexto.

La WWW nació en 1989 en el CERN, de la mano de Tim Berners-Lee.

Como podemos observar, al final de cada párrafo se agrega automáticamente un salto de línea.

- **<h1> ... <h6>**, de *heading* (título): Definen hasta seis niveles de títulos, desde el más importante (<h1>) hasta el menos importante (<h6>). Por ejemplo, las líneas siguientes:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1>Introducción</h1>
<p>La WWW es un sistema de páginas enlazadas mediante el concepto de hipertexto.</p>
<h2>Historia</h2>
<p>La WWW nació en 1989 en el CERN, de la mano de Tim Berners-Lee.</p>
</body>
</html>
```

definen un título de primer nivel (?Introducción?) y otro de segundo nivel (?La Web y el HTML?). Después de cada título, aparece algo de texto en forma de párrafo. El resultado será el siguiente:

Introducción

La WWW es un sistema de páginas enlazadas mediante el concepto de hipertexto.

Historia

La WWW nació en 1989 en el CERN, de la mano de Tim Berners-Lee.

- ** **, de *divider* (divisor): Este elemento define un bloque que puede contener otros elementos. Es muy parecido al párrafo, aunque está vacío de significado. El elemento es muy importante porque nos permite agrupar elementos sin caracterizarlos. Las líneas siguientes:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<div align="center">
<img src="http://upload.wikimedia.org/wikipedia/commons/thumb/b/b2/WWW_logo_by_Robert_Cailliau.svg
<p>El primer logotipo del WWW</p>
</div>
</body>
</html>
```

crean un bloque que contiene dos párrafos y lo centran en medio del documento mediante el atributo "align" (de *alignment*, alineación). Consecuentemente, los párrafos también quedarán alineados en el centro. El resultado será el siguiente:

El primer logotipo del WWW

- **
**, de *line break* (salto de línea): Permite emplazar un salto de línea, por ejemplo, en medio de un párrafo. Este elemento no tiene etiqueta de fin. El código siguiente:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>En la página: <br> http://www.w3c.org <br> se puede obtener información sobre el WWW Consortium
</body>
</html>
```

escribirá:

En la página:
http://www.w3c.org
se puede obtener información sobre el WWW Consortium.

Observad que, a diferencia de lo que sucede en el párrafo, el salto de línea no modifica el espaciado entre líneas.

Elementos de presentación

Como ya hemos mencionado, los elementos de presentación son aquellos que describen el aspecto del texto, independientemente de su función. Entre ellos destacan los siguientes:

- ****, de *emphasis* (énfasis): Este elemento enfatiza el texto. Normalmente, el texto se mostrará en cursiva. Las líneas siguientes:

```
<!DOCTYPE html>
<html>
<head>
```

```
</head>
<body>
<p>La <em>World Wide Web</em> es un sistema de páginas enlazadas mediante el concepto de hipertext
</body>
</html>
```

se traducirán en:

La *World Wide Web* es un sistema de páginas enlazadas mediante el concepto de hipertexto.

- ****, de *strong emphasis* (énfasis fuerte): Este elemento es parecido al anterior, pero pone un énfasis aún mayor. Normalmente, el texto se mostrará en negrita. Las líneas siguientes:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>La <strong>World Wide Web</strong> es un sistema de páginas enlazadas mediante el concepto de h
</body>
</html>
```

se traducirán en:

La **World Wide Web** es un sistema de páginas enlazadas mediante el concepto de hipertexto.

Elementos de hipertexto

El elemento principal que permite navegar de un documento a otro es <a>, de *anchor* (ancla). Dicho elemento permite definir un texto que, cuando se activa (normalmente haciendo clic sobre él), nos lleva a otro documento. Por ejemplo, las líneas siguientes:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>La WWW nació en 1989 en el CERN, de la mano de <a href="http://es.wikipedia.org/wiki/ Tim_Berne
</body>
</html>
```

crean un enlace entre el nombre del inventor de la WWW y el artículo correspondiente de la Wikipedia. Como se puede observar, el elemento ancla tiene un atributo "href" (de *hypertext reference*, referencia de hipertexto), que establece el documento con el que está enlazado el texto. El resultado será el siguiente:

La WWW nació en 1989 en el CERN, de la mano de Tim Berners-Lee.

Al activar el enlace, ya sea haciendo clic sobre él o por otros medios, iremos a la página http://es.wikipedia.org/wiki/Tim_Berners-Lee.

Google Maps

Google Maps es la herramienta SIG en línea de Google. Permite explorar mapas en 2D de casi cualquier parte del mundo. La popularidad de la que goza Google Maps en la actualidad se debe principalmente a su

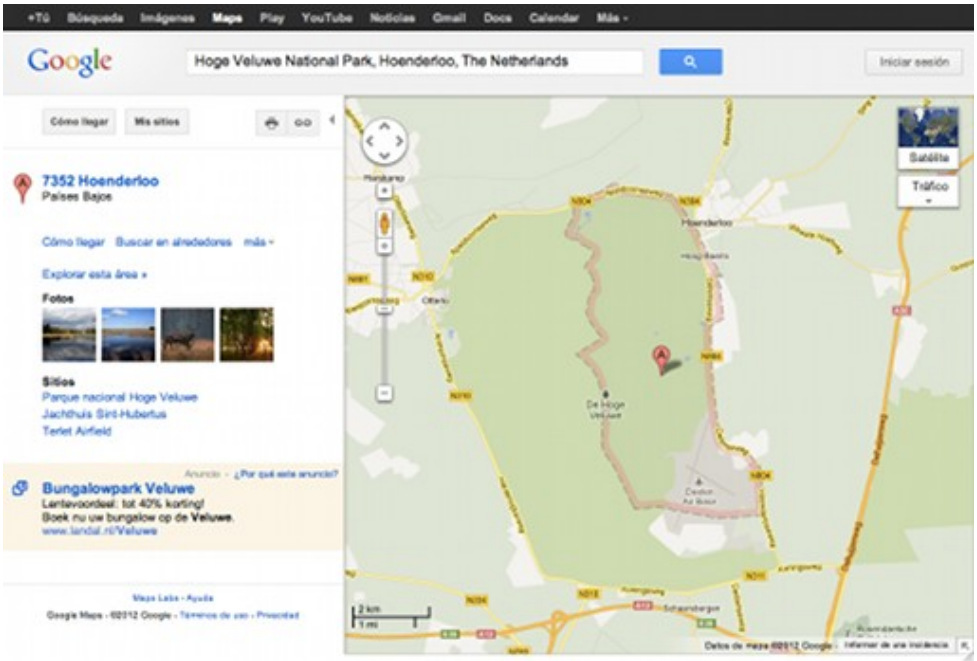
buen rendimiento, su vasta cartografía y la posibilidad de incorporar toda esta tecnología a cualquier sitio web mediante el uso de la API [2] (de *Application Programming Interface*, "interfaz de programación de aplicaciones") de Google Maps.

En los apartados siguientes, veremos cómo incorporar un mapa a nuestra página y cómo hacer uso de la API para trabajar sobre él.

Incorporar un mapa en una página

La forma más sencilla de incorporar un mapa en nuestra página es utilizando el web de Google Maps y, una vez seleccionada la ubicación que deseamos mostrar, incorporar el código HTML que nos proporciona Google a nuestra página.

Veámoslo paso a paso. En primer lugar, debemos seleccionar una ubicación en http://maps.google.es. Esto se puede hacer mediante la caja de búsqueda o navegando por el mapa.



En segundo lugar, tenemos que presionar el botón "Enlazar" (cuyo icono es una cadena) y copiar el código HTML que aparece en la caja ?Pegar enlace en mensaje de correo electrónico o mensaje instantáneo?.

El tercer y último paso será incorporar este código en nuestra página HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Nationaal Park De Hoge Veluwe</title>
</head>
<body>
<p>El <em>Nationaal Park De Hoge Veluwe</em> (el Parque Nacional <em>De Hoge Veluwe</em> es un par
<iframe width="425" height="350" frameborder="0" scrolling="no" marginheight="0" marginwidth="0" s
<small><a href="http://maps.google.es/maps?f=q&source=embed&hl=es&geocode=&q=Hoge+Veluwe+National+
</body>
</html>
```

El resultado será el siguiente:

El *Nationaal Park De Hoge Veluwe* (el Parque Nacional *De Hoge Veluwe* es un parque nacional neerlandés situado en la provincia de Gelderland, cerca de las ciudades de Ede, Arnhem y Apeldoorn.

Ver mapa más grande

Si bien el mapa que obtenemos es interactivo (podemos hacer zoom y desplazarnos por él) y algo personalizable (podemos cambiar mínimamente el aspecto de nuestro mapa mediante la opción "Personalizar y obtener vista previa del mapa que se va a insertar"), las posibilidades que nos ofrece son muy limitadas. Por ejemplo, no se pueden añadir o quitar capas, y no se puede asociar información a determinados puntos del mapa. Sin embargo, si nuestra única intención es situar un lugar en el mapa, ésta es una solución fácil y rápida.

La API de Google Maps

Afortunadamente, Google Maps nos ofrece muchas más posibilidades que las que hemos visto hasta ahora. Sin embargo, éstas se obtienen a costa de la facilidad de uso, pues se requieren unos mínimos conocimientos de programación para poder sacarles partido.

A esta funcionalidad adicional se accede por medio de la API de Google Maps, mediante un conjunto de tecnologías agrupadas bajo la denominación AJAX (*Asynchronous JAvascript and XML*, "JavaScript

asíncrono y XML"). Como su nombre indica, AJAX está compuesto por dos tecnologías:

- **JavaScript**: Es un lenguaje muy parecido a Java que se usa para alterar documentos HTML de forma dinámica. Es importante remarcar que no es Java; sólo comparten una base comuna.
- **XML**: Es un lenguaje usado para el intercambio de datos (sigla de *eXtensible Markup Language*, "lenguaje de marcas extensible"). Es una forma genérica de escribir documentos maximizando su simplicidad, su generalidad y su usabilidad.

Ambas tecnologías, integradas en un documento HTML, permiten obtener mapas y datos de los servidores de Google Maps y mostrarlos en una página web.

En los siguientes capítulos estudiaremos la Versión 3 de la API de Google Maps, la versión oficial de la API cuando se actualizaron por última vez estos materiales.

Crear un mapa

La función más elemental que nos ofrece la API de Google Maps es crear un mapa. El mapa se mostrará de forma muy parecida a cuando se copia el código HTML del sitio de Google Maps: centrado en un punto y con un nivel de zoom determinado. Además, se mostrarán una serie de controles mínimos que nos permitirán desplazarnos por el mapa, variar su nivel de zoom y seleccionar el tipo de mapa mostrado.

El código siguiente muestra un mapa centrado en el rectorado de la UOC:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Universitat Oberta de Catalunya</title>
<script type="text/javascript"
    src="http://maps.googleapis.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
function InicializarMapa() {
        var opciones = {
                center: new google.maps.LatLng(41.414829, 2.133003),
                zoom: 17,
                mapTypeId: google.maps.MapTypeId.ROADMAP
        }
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opciones);
}
</script>
</head>
<body onload="InicializarMapa()">
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
</body>
</html>
```

El resultado será el siguiente:

Como podemos observar, el código HTML contiene una serie de elementos <script> que son los que introducen el código JavaScript. Estos elementos pueden emplazarse tantas veces como sea necesario en la cabecera del documento (<head>) o en el contenido (<body>). Es importante remarcar el uso del atributo "type" ("tipo") del elemento <script>, que sirve para identificar el tipo de código que contiene (en nuestro caso JavaScript).

Si observamos detenidamente el código JavaScript contenido en los elementos <script>, las similitudes con Java saltan rápidamente a la vista pero también las diferencias; Quizá la más notable sea la ausencia de tipos. Todas las variables son del mismo tipo (*var*), aunque internamente puedan tener atributos distintos. La declaración de funciones también cambia, pues en JavaScript siempre toma la forma: ***function*** *nombre-función*.

En los siguientes apartados, analizaremos con detalle el código del ejemplo y descubriremos los pasos que hay que seguir para mostrar un mapa en un página web.

Crear el elemento que albergará el mapa

En primer lugar, es necesario crear un elemento HTML que albergue el mapa. Aunque en la práctica hay diversos elementos que pueden desempeñar esta función, es recomendable utilizar el elemento porque es el elemento más neutro. Al elemento utilizado se debe darle un nombre mediante el atributo ?id? (de *identifier*, "identificador") y unas dimensiones, que se especificarán mediante los atributos estilísticos *width* (ancho) y *height* (alto). En el ejemplo anterior, hemos creado un elemento llamado "elemento_mapa" con un tamaño de 500 por 300 píxeles mediante el código siguiente:

```
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
```

Incluir la API de Google Maps

Para poder acceder a la API de Google Maps es necesario incluirla (incorporarla) a nuestro código. En el ejemplo anterior, esto lo hemos conseguido mediante las líneas siguientes:

```
<script type="text/javascript"
    src="http://maps.googleapis.com/maps/api/js?sensor=false">
</script>
```

Estas líneas pueden cambiar ligeramente de una versión a otra de la API, y también si se desea acceder a otras capacidades de Google Maps, como por ejemplo, obtener las coordenadas geográficas del usuario que está accediendo a la página, usar librerías adicionales, etc.

Inicializar el mapa

Una vez creado el elemento que contendrá el mapa, entra en juego el código JavaScript para pedir a la API de Google Maps que dibuje un mapa sobre dicho elemento.

En el ejemplo anterior, esto lo hemos realizado mediante las líneas siguientes:

```
<script type="text/javascript">
function InicializarMapa() {
        var opciones = {
                center: new google.maps.LatLng(41.414829, 2.133003),
                zoom: 17,
                mapTypeId: google.maps.MapTypeId.ROADMAP
        }
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opciones);
}
</script>
```

Como podemos observar, en la última línea de la función "InicializarMapa" se instancia un objeto de la clase "google.maps.Map" que es el mapa en sí. En la llamada al constructor se le pasa el elemento que debe

contener el mapa (en el ejemplo, "elemento_mapa") y una serie de opciones que determinarán el centro del mapa, el nivel de zoom inicial y el tipo de mapa mostrado. Una vez instanciado el objeto "Map", la API de Google Maps dibuja el mapa sobre el elemento especificado.

Cabe destacar que las opciones se especifican mediante un objeto "opciones" que contiene tres atributos:

- **center**: que es un objeto de la clase "google.maps.LatLng" que especifica las coordenadas WGS84 del centro del mapa;
- **zoom**: que es un número entero que indica el nivel de zoom inicial, comprendido entre 0 (más alejado) y 19 (más próximo);
- **mapTypeId**: que es un identificador que especifica el tipo del mapa ("google.maps.MapTypeId.ROADMAP" para el mapa de carreteras).

Es obligatorio especificar, al menos, estos tres atributos.

Llamar a la función de inicialización

Finalmente, sólo queda indicar al navegador cuándo se debe inicializar el mapa o, lo que es lo mismo, cuándo debe ejecutarse la función "InicializarMapa".

Aunque pueda parecer un poco raro al principio, un documento HTML cambia a medida que va leyéndose. Un navegador web lee en primer lugar el código HTML (es decir, el texto del documento) y después las imágenes u otros objetos (vídeos, etc.). Dadas las circunstancias, hasta que todos los objetos no están cargados no se puede mostrar el documento tal y como es, pero esperar a recibir todos los objetos para mostrar el texto puede ser desesperante para el usuario, sobretodo si el servidor web es lento o la cantidad de datos a transferir es muy grande. En general, los navegadores optan por ir mostrando los objetos a medida que los van recibiendo, componiendo el documento sobre la marcha.

No obstante, esta política de los navegadores puede causar problemas a Google Maps. Si el mapa se inicializa cuando el documento aún no ha sido cargado del todo, se pueden obtener resultados inesperados. En este sentido, Google recomienda inicializar el mapa sólo cuando el documento haya sido leído completamente.

Para ello, se utiliza el atributo "onload" (cuando esté cargado) del elemento HTML <body>. En él se puede introducir código JavaScript que se ejecutará cuando la página haya sido cargada completamente. En el ejemplo que nos ocupa, le hemos puesto puesto una llamada a la función "InicializarMapa":

```
<body onload="InicializarMapa()">
```

Tipos de mapas

En en ejemplo anterior, hemos creado un mapa de carreteras. Aunque el tipo de mapa puede cambiarse a posteriori mediante los botones situados en la parte superior derecha del mapa, a veces es deseable mostrar un u otro tipo desde buen comienzo.

Así, el atributo "mapTypeId" del objeto mapa puede tomar los valores siguientes:

- **ROADMAP**: mapa de calles y carreteras.
- **SATELLITE**: mapa basado en imágenes tomadas por satélite.
- **HYBRID**: mapa híbrido, con las calles y las carreteras superpuestas a las imágenes tomadas por satélite.
- **TERRAIN**: mapa de relieve, una especie de mapa topográfico.

Si en el ejemplo anterior cambiamos el valor del atributo "mapTypeId" por "google.maps.MapTypeId.HYBRID" obtenemos el resultado siguiente:

El código completo del ejemplo queda así:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Universitat Oberta de Catalunya</title>
<script type="text/javascript"
    src="http://maps.googleapis.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
function InicializarMapa() {
        var opciones = {
                center: new google.maps.LatLng(41.414829, 2.133003),
                zoom: 17,
                mapTypeId: google.maps.MapTypeId.HYBRID
        }
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opciones);
}
</script>
</head>
<body onload="InicializarMapa()">
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
</body>
</html>
```

Transformación de coordenadas

La característica más relevante y distintiva de los SIG es que facilitan la transformación de coordenadas de pantalla a coordenadas geográficas y viceversa. Este mecanismo nos permite determinar (sin hacer cálculos complicados) las coordenadas geográficas de un punto de mapa, o situar un elemento (un marcador, por ejemplo) en unas coordenadas geográficas específicas. En los siguientes apartados veremos como se resuelven ambas tareas con Google Maps.

Situar un elemento sobre el mapa

Supongamos que queremos marcar un lugar concreto sobre el mapa. Para este propósito, la API de Google Maps nos proporciona los marcadores, que se pueden utilizar para apuntar sobre unas coordenadas determinadas. Estos marcadores están implementados por la case "google.maps.Marker", cuyo constructor espera las coordenadas geográficas del punto que debe señalar y el mapa sobre el que se emplazará.

El ejemplo siguiente destaca la situación del rectorado de la UOC (cuyas coordenadas son: 41,414829º N, 2,133003º E) en medio de Barcelona mediante el uso de un marcador:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Universitat Oberta de Catalunya</title>
<script type="text/javascript"
        src="http://maps.googleapis.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
function InicializarMapa() {
```

```
        var posicionRectorado = new google.maps.LatLng(41.414829, 2.133003);

        var opcionesMapa = {
                center: posicionRectorado,
                zoom: 11,
                mapTypeId: google.maps.MapTypeId.ROADMAP
        }
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opcionesMapa);

        var opcionesMarcador = {
                position: posicionRectorado,
                map: mapa,
                title: "El rectorado de la UOC."
        }
        var marcador = new google.maps.Marker(opcionesMarcador);
}
</script>
</head>
<body onload="InicializarMapa()">
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
</body>
</html>
```

El resultado es el siguiente:

Como se puede observar en el código del ejemplo, en primer lugar hemos instanciado un objeto de la clase "google.maps.LatLng" que representa las coordenadas geográficas del rectorado de la UOC, y que se ha utilizado más tarde para establecer el centro del mapa y la posición del marcador:

```
var posicionRectorado = new google.maps.LatLng(41.414829, 2.133003);
```

Después, hemos creado el mapa, de la misma forma que lo hemos hecho en los ejemplos anteriores.

En última instancia hemos creado el marcador llamando al constructor de la clase "google.maps.Marker". De la misma forma que lo hacíamos con los mapas, nos hemos ayudado de un objeto "opcionesMarcador" para especificar las opciones del marcador:

```
var opcionesMarcador = {
        position: posicionRectorado,
        map: mapa,
        title: "El rectorado de la UOC."
}
var marcador = new google.maps.Marker(opcionesMarcador);
```

Es importante observar como los valores de los atributos del objeto "opcionesMarcador" determinan la posición y el aspecto del marcador. A continuación se detalla el significado de cada atributo:

- **position**: Las coordenadas geográficas del lugar que debe señalar el marcador.
- **map**: El mapa sobre el que debe señalar el marcador. Hay que tener en cuenta que podemos tener dos o más mapas en una misma página.
- **title**: Una pequeña descripción del marcador, que aparecerá al cabo de unos instantes de situarse sobre él.

Los atributos "position" y "map" deben especificarse obligatoriamente. No así el atributo "title", que es opcional.

Determinar las coordenadas de un punto del mapa

Supongamos ahora que deseamos hacer el proceso inverso: determinar las coordenadas geográficas de un lugar del mapa. Es necesario recurrir a esta transformación cuando, por ejemplo, deseamos conocer las coordenadas geográficas de un punto del mapa sobre el que acabamos de hacer clic.

El ejemplo siguiente hace exactamente eso: mostrar las coordenadas geográficas (expresadas en grados Norte y Este) del punto seleccionado:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Determinar coordenadas</title>
<script type="text/javascript"
        src="http://maps.googleapis.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
function MostrarCoordenadas(mapa, latLng)
{
        var opcionesVentana = {
                content: "Las coordenadas del punto son:<br>(" +
                        latLng.lat() + " °N, " +
                        latLng.lng() + " °E)",
                position: latLng
        }
        var ventanaInfo = new google.maps.InfoWindow(opcionesVentana);

        ventanaInfo.open(mapa);
}

function InicializarMapa() {
        // establece las opciones y crea el mapa
        var opcionesMapa = {
                center: new google.maps.LatLng(41.60340967242644, 1.811567071655304),
                zoom: 11,
                mapTypeId: google.maps.MapTypeId.TERRAIN
        }
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opcionesMapa);

        // activa la escucha del evento
        google.maps.event.addListener(mapa, "click", function(evento) {
                // mostrar las coordenadas del punto seleccionado
                MostrarCoordenadas(mapa, evento.latLng);
        });
}
</script>
</head>
<body onload="InicializarMapa()">
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
</body>
</html>
```

El resultado es el siguiente (haz clic sobre el mapa!):

Para que nuestro código responda a los clics del ratón, hemos llamado a la función "addListener()" de la clase "google.maps.event". Esta función permite asociar un evento (un clic, un doble clic, un cambio de zoom, etc.) de un objeto (un mapa, un marcador, etc.) a una función, de forma que cada vez que suceda el evento se llame a la función especificada. En el ejemplo esto se conseguía por medio de las siguientes líneas:

```
google.maps.event.addListener(mapa, "click", function(evento) {
```

```
                // mostrar las coordenadas del punto seleccionado
                MostrarCoordenadas(mapa, evento.latLng);
});
```

Es importante observar que el tercer parámetro de la llamada a "addListener" es una función anónima que implementamos *ipso facto*. Podíamos haber declarado una función normal y corriente, y aquí solo especificar su nombre, pero esta construcción nos da la ventaja de que nos permite acceder a la variable "mapa" (que necesitamos para poder mostrar las coordenadas) sin tener que declararla fuera del ámbito de la función "InicializarMapa()".

También debéis fijaros en que la función anónima espera un parámetro "evento", que la API de Google Maps se encargará de proporcionarnos en forma de un objeto "MouseEvent". Este objeto nos interesa especialmente porque tiene un atributo "latLng" que contiene las coordenadas donde ha sucedido el evento del clic.

Cuando se desencadena el evento, mapa y coordenadas se pasan a la función "mostrarCoordenadas" que muestra, mediante un objeto "google.maps.InfoWindow" las coordenadas en cuestión debidamente formateadas:

```
var opcionesVentana = {
        content: "Las coordenadas del punto son:<br>(" +
                latLng.lat() + " °N, " +
                latLng.lng() + " °E)",
        position: latLng
}
var ventanaInfo = new google.maps.InfoWindow(opcionesVentana);

ventanaInfo.open(mapa);
```

Como se puede observar, la instanciación de un objeto "google.maps.InfoWindow" es análoga a la de mapas y marcadores. En este caso, los atributos a proporcionar son:

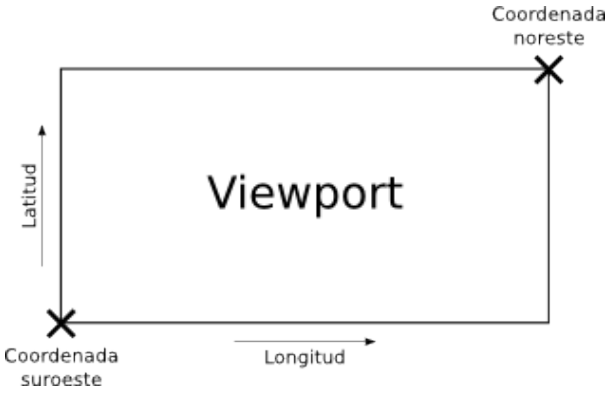
- **content**: El texto a mostrar.
- **position**: Las coordenadas sobre las que apuntará el mensaje.

El viewport

Aunque el módulo anterior (Adaptación y estensión de un SIG) ya se ha definido el termino *viewport*, a continuación volveremos a definirlo y veremos qué funciones de la API de Google Maps nos permiten acceder a sus atributos.

Cuando se muestra un mapa por pantalla, no siempre se muestra en toda su extensión. Que veamos una parte mayor o menor del mismo depende del nivel de zoom. De hecho, el área visible y el nivel de zoom se rigen por una regla de proporcionalidad inversa: a mayor zoom, menor área visible, y a menor zoom, mayor área visible.

El recuadro que delimita el área visible de un mapa se llama *viewport*. Normalmente, se define por las coordenadas geográficas de sus esquinas inferior izquierda (suroeste) y superior derecha (noreste). Latitud y longitud crecen desde la coordenada suroeste hasta la coordenada noreste.



Google Maps nos permite obtener los límites del *viewport* mediante la función "getBounds()" del objeto mapa. Esta función devuelve un objeto de la clase "LatLngBounds" que contiene las coordenadas geográficas de las esquinas suroeste y noreste del viewport. A su vez, dichas coordenadas las leeremos mediante las funciones "getSouthWest()" y "getNorthEast()" respectivamente.

El ejemplo siguiente hace uso de todas estas funciones para escoger aleatoriamente el destino de nuestras próximas vacaciones:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Viaja fácil, rápido y barato</title>
<script type="text/javascript"
        src="http://maps.googleapis.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
function EscogerDestino(limitesViewport, marcador) {
        // obtiene las coordenadas de las esquinas suroeste y noreste
        var esquinaSuroeste = limitesViewport.getSouthWest();
        var esquinaNoreste = limitesViewport.getNorthEast();

        // genera un destino aleatorio dentro de los límites del viewport
        var posicionDestino = new google.maps.LatLng(
                esquinaSuroeste.lat() + (esquinaNoreste.lat() - esquinaSuroeste.lat()) * Math.ranc
                esquinaSuroeste.lng() + (esquinaNoreste.lng() - esquinaSuroeste.lng()) * Math.ranc
        );

        // cambia la posición del marcador
        marcador.setPosition(posicionDestino);
}

function InicializarMapa() {
        // establece las opciones y crea el mapa
        var opcionesMapa = {
                center: new google.maps.LatLng(47.56844787466107, 9.327588800170929),
                zoom: 3,
                mapTypeId: google.maps.MapTypeId.TERRAIN
        }
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opcionesMapa);

        // crea un marcador sin apuntar a ninguna parte
        var opcionesMarcador = {
                map: mapa,
                title: "Aquí viajaremos el próximo verano."
        }
        var marcador = new google.maps.Marker(opcionesMarcador);
```

```
        // activa la escucha del evento "idle"
        google.maps.event.addListener(mapa, "idle", function(evento) {
                // escoge un destino al azar
                EscogerDestino(mapa.getBounds(), marcador);
        });
}
</script>
</head>
<body onload="InicializarMapa()">
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
</body>
</html>
```

El resultado es el siguiente:

Os gusta el destino que nos ha tocado? Si es que no, no pasa nada: basta con mover el mapa o cambiar el nivel de zoom y el destino cambiará aleatoriamente.

En lo referente al código, antes de entrar en detalles acerca del *viewport*, hay que tener en cuenta que los límites el *viewport* solo se pueden leer cuando el mapa está totalmente cargado. Si probamos de hacerlo antes no obtendremos ningún valor. Por eso, para leerlos en el momento oportuno, escuchamos el evento "idle", que sucede cuando el mapa ha terminado de dibujarse después de su inicialización, de un cambio de zoom o de un desplazamiento.

Así, cada vez que salta el evento "idle" llamamos a la función "EscogerDestino()". Esta función se encarga de escoger un nuevo destino aleatoriamente, y de cambiar el marcador de posición. Por ese motivo, recibe dos parámetros: los límites del *viewport* (que se obtienen llamando a la función "getBounds()") y el marcador:

```
// activa la escucha del evento "idle"
google.maps.event.addListener(mapa, "idle", function(evento) {
        // escoge un destino al azar
        EscogerDestino(mapa.getBounds(), marcador);
});
```

Es importante entender que el marcador es siempre el mismo. Lo que hacemos en la función "EscogerDestino()" es cambiar el marcador de sitio. También es importante observar que, después de crear el mapa, creamos el marcador aunque sin posición inicial:

```
// crea un marcador sin apuntar a ninguna parte
var opcionesMarcador = {
        map: mapa,
        title: "Aquí viajaremos el próximo verano."
}
var marcador = new google.maps.Marker(opcionesMarcador);
```

Observad que el atributo "position" del objeto "opcionesMarcador" (que contiene la posición sobre la que apunta el marcador) no está presente. La ausencia de atributo provoca que el marcador no se dibuje. Sin embargo, esto no nos supone ningún inconveniente porque, de hecho, aún no hemos escogido nuestro destino vacacional. No será hasta que el mapa no esté listo, es decir, recordad, hasta que no esté en estado "idle", o, lo que es lo mismo, hasta que no se llame la función "EscogerDestino()" y podamos obtener las dimensiones del *viewport*, que no se calculará el nuevo destino y se establecerá la posición del marcador.

La posición de un marcador se puede cambiar mediante la función "setPosition()" del objeto marcador. En el ejemplo, en la última línea de la función "EscogerDestino()", utilizamos esta función para cambiar la posición del marcador:

```
function EscogerDestino(limitesViewport, marcador) {
        // obtiene las coordenadas de las esquinas suroeste y noreste
        var esquinaSuroeste = limitesViewport.getSouthWest();
        var esquinaNoreste = limitesViewport.getNorthEast();

        // genera un destino aleatorio dentro de los límites del viewport
        var posicionDestino = new google.maps.LatLng(
                esquinaSuroeste.lat() + (esquinaNoreste.lat() - esquinaSuroeste.lat()) * Math.rand
                esquinaSuroeste.lng() + (esquinaNoreste.lng() - esquinaSuroeste.lng()) * Math.rand
        );

        // cambia la posición del marcador
        marcador.setPosition(posicionDestino);
}
```

Observad ahora las líneas que preceden la llamada a la función "setPosition()". En ellas obtenemos las coordenadas de las esquinas suroeste y noreste del *viewport* y generamos aleatoriamente un nuevo destino. Pero, ¿como generar un nuevo destino aleatorio que, además, esté dentro de los límites del *viewport*? Pues incrementando la latitud y la longitud de la coordenada suroeste un valor aleatorio entre 0 y la altura del *viewport*, en el caso de la latitud, y entre 0 y la anchura del *viewport*, en el caso de la longitud.

La altura del *viewport* la obtenemos restando las latitudes de las esquinas noreste y suroeste. La anchura, restando las longitudes. Ambos valores los multiplicamos por el resultado de la función "Math.random()" que devuelve un número aleatorio entre 0,0 y 1,0 (p.ej.: 0,2), obteniendo los incrementos en latitud y longitud que buscábamos.

Líneas y polígonos

Google Maps nos permite representar objetos sobre un mapa, de manera que su geometría esté ligada a unas coordenadas geográficas determinadas. Buen ejemplo de ello son las marcas que hemos visto hasta ahora: se acomodan a los desplazamientos del mapa y a los cambios de zoom para apuntar siempre sobre el mismo lugar.

Sin embargo, las opciones que nos ofrece Google Maps no se acaban en los marcadores: también podemos representar líneas, polilíneas [3], polígonos e incluso círculos.

Líneas y polilíneas

Supongamos que queremos representar la ruta que hay que seguir para ir de Plaça de Catalunya de Barcelona al rectorado de la UOC. El código siguiente dibuja la ruta mediante una polilínea [4]:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Como llegar al Rectorado de la UOC</title>
<script type="text/javascript"
        src="http://maps.googleapis.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
function InicializarMapa() {
        // coordenadas de Plaza Catalunya
        var posicionPlCatalunya = new google.maps.LatLng(41.386935, 2.169950);

        // establece las opciones y crea el mapa
        var opcionesMapa = {
```

```
                center: posicionPlCatalunya,
                zoom: 15,
                mapTypeId: google.maps.MapTypeId.ROADMAP
        };
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opcionesMapa);

        // dibuja la ruta entre Plaza Catalunya i el Rectorado de la UOC

        // especifica los puntos de paso
        var puntosDePaso = [
                posicionPlCatalunya,
                new google.maps.LatLng(41.387998, 2.17005),
                new google.maps.LatLng(41.396513, 2.15947),
                new google.maps.LatLng(41.399813, 2.15550),
                new google.maps.LatLng(41.402018, 2.15293),
                new google.maps.LatLng(41.403965, 2.15186),
                new google.maps.LatLng(41.405861, 2.15037),
                new google.maps.LatLng(41.406492, 2.14954),
                new google.maps.LatLng(41.408279, 2.14746),
                new google.maps.LatLng(41.411252, 2.14513),
                new google.maps.LatLng(41.411771, 2.14081),
                new google.maps.LatLng(41.413990, 2.13794),
                new google.maps.LatLng(41.414368, 2.13716),
                new google.maps.LatLng(41.415044, 2.13631),
                new google.maps.LatLng(41.414980, 2.13599),
                new google.maps.LatLng(41.414658, 2.13502),
                new google.maps.LatLng(41.414457, 2.13393),
                new google.maps.LatLng(41.414867, 2.13367),
                new google.maps.LatLng(41.414796, 2.13323)
        ];

        // especifica las opciones de la línea
        var opcionesPolilinea = {
                path: puntosDePaso,
                strokeColor: "#ff00ff",
                strokeWeitght: 10,
                strokeOpacity: 0.5,
                geodesic: true,
                map: mapa
        };

        // crea la línea
        var polilinea = new google.maps.Polyline(opcionesPolilinea);
}
</script>
</head>
<body onload="InicializarMapa()">
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
</body>
</html>
```

El resultado es el siguiente:

Como se puede observar en el código del ejemplo, las polilíneas se representan mediante objetos "Polyline". La llamada al constructor de esta clase es muy parecida al de los marcadores:

```
// especifica las opciones de la línea
var opcionesPolilinea = {
        path: puntosDePaso,
        strokeColor: "#ff00ff",
        strokeWeitght: 10,
        strokeOpacity: 0.5,
        geodesic: true,
```

```
        map: mapa
};

// crea la línea
var polilinea = new google.maps.Polyline(opcionesPolilinea);
```

Observad que la geometría y el aspecto de la polilínea vienen determinados por los atributos del objeto "opcionesPolilinea". A continuación se detalla el papel que juega cada atributo:

- **path**: Vector de puntos que representan los vértices de la polilínea. El primer vértice se unirá con el segundo, el segundo con el tercero, y así sucesivamente mediante sendas líneas rectas. En el ejemplo, este atributo se inicializa con el contenido del vector "puntosDePaso". Es importante observar que los objetos del vector son instancias de la clase "LatLng".
- **strokeColor**: Color de la polilínea expresado en notación hexadecimal [5]. En el ejemplo, este atributo se establece a "#ff00ff" (magenta).
- **strokeWeight**: Anchura de la polilínea expresada en píxeles. En el ejemplo, este atributo se establece a "10".
- **strokeOpacity**: Opacidad de la línea expresada como un valor entre 0,0 y 1,0. 0,0 és totalmente transparente, y 1,0, totalmente opaca. En el ejemplo, 0,5 establece una opacidad del 50%.
- **map**: Mapa sobre el que se dibujará la línea.

Es importante destacar que, de forma predeterminada, los segmentos que unen los puntos de una polilínea se dibujan rectos respecto a la proyección, no respecto a la superfície terrestre. Esto significa que, si los puntos de una polilínea están suficientemente separados (del orden de miles de kilómetros), los segmentos que sobre el mapa son rectos sobre el terreno serán más o menos curvos y viceversa. Esto es así porque la Tierra es esférica, y para representarla sobre un mapa necesitamos deformarla ligeramente.

Si deseamos que los segmentos de una polilínea sean líneas geodésicas (es decir, aquellas que representan la distancia más corta entre dos puntos de la superfície terrestre), debemos pasarle al constructor de la clase "Polyline" la opción "geodesic: true". En el ejemplo siguiente, se representa la ruta París - Moscú - Pequín por partida doble: en rojo mediante líneas rectas respecto la proyección, y en azul mediante líneas geodésicas:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>París - Moscú - Pequín</title>
<script type="text/javascript"
        src="http://maps.googleapis.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
function InicializarMapa() {
        // establece las opciones y crea el mapa
        var opcionesMapa = {
                center: new google.maps.LatLng(47.0, 59.0),
                zoom: 2,
                mapTypeId: google.maps.MapTypeId.TERRAIN
        };
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opcionesMapa);

        // coordenadas de París, Moscú y Pequín
        posicionParis = new google.maps.LatLng(48.856578,   2.351828);
        posicionMoscu = new google.maps.LatLng(55.752222,  37.615556);
        posicionPekin = new google.maps.LatLng(39.905556, 116.391389);

        // dibuja la ruta París - Moscú - Pequín, utilizando segmentos rectos
        // respecto a la proyección (polilínea estándar)
```

```
        var opcionesRutaParisPequin = {
                path: [ posicionParis, posicionMoscu, posicionPekin ],
                strokeColor: "#ff0000",
                strokeWeitght: 10,
                strokeOpacity: 0.5,
                map: mapa
        };
        var rutaParisPequin = new google.maps.Polyline(opcionesRutaParisPequin);

        // dibuja la misma ruta utilizando rectas geodésicas
        var opcionesGeodesicaParisPequin = {
                path: [ posicionParis, posicionMoscu, posicionPekin ],
                strokeColor: "#0000ff",
                strokeWeitght: 10,
                strokeOpacity: 0.5,
                geodesic: true,
                map: mapa
        };
        var geodesicaParisPequin = new google.maps.Polyline(opcionesGeodesicaParisPequin);
}
</script>
</head>
<body onload="InicializarMapa()">
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
</body>
</html>
```

El resultado se muestra a continuación. Notad que la ruta azul es la más corta sobre la superfície terrestre:

Polígonos

Aunque con un objeto "Polyline" sería posible dibujar un polígono (en el caso de que el último punto fuera igual al primero), Google Maps nos ofrece una clase especializada para este propósito: "Polygon".

Al igual que sucede con las polilíneas, podemos definir el color, el ancho y la opacidad del borde del polígono, pero además, el color y la opacidad del relleno.

Las líneas siguientes dibujan un polígono que delimita la superfície de la Plaça de Catalunya de Barcelona:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Plaça de Catalunya de Barcelona</title>
<script type="text/javascript"
        src="http://maps.googleapis.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
function InicializarMapa() {
        // establece las opciones y crea el mapa
        var opcionesMapa = {
                center: new google.maps.LatLng(41.387024, 2.169960),
                zoom: 16,
                mapTypeId: google.maps.MapTypeId.ROADMAP
        };
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opcionesMapa);

        var esquinasPlaza = [
                new google.maps.LatLng(41.387877, 2.169971),
                new google.maps.LatLng(41.387418, 2.168748),
```

```
                new google.maps.LatLng(41.385856, 2.170143),
                new google.maps.LatLng(41.386919, 2.171205)
        ];

        // delimita el contorno de la plaza
        var contornoPlaza = new google.maps.Polygon({
                paths: esquinasPlaza,
                strokeColor: "#ff0000",  // color del contorno
                strokeOpacity: 0.5,      // opacidad del contorno
                strokeWeight: 10,        // ancho del contorno
                fillColor: "#ff0000",    // color de relleno
                fillOpacity: 0.2,        // opacidad del relleno
                map: mapa                // mapa sobre el que se debe dibujar el polígono
        });
}
</script>
</head>
<body onload="InicializarMapa()">
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
</body>
</html>
```

El resultado es el siguiente:

Como se puede observar, el código es casi idéntico al que hemos escrito para dibujar una polilínea. Las únicas diferencias notables son el uso de la clase "Polygon" (en lugar de la clase "Polyline"), y el mayor número de opciones de los polígonos. El significado de cada una de las opciones se detalla a continuación. Observad que los atributos "paths", "strokeColor", "strokeWeight", "strokeOpacity" y "map" ya se encontraban entre las opciones de las polilíneas:

- **paths**: Vector de puntos que representan las esquinas del polígono. La primera esquina se unirá con la segunda, la segunda con la tercera, y así sucesivamente mediante sendas líneas rectas. En el ejemplo, este atributo se inicializa con el contenido del vector "esquinasPlaza". Es importante observar que los objetos del vector son instancias de la clase "LatLng". La "s" final de "paths" no es un error: se pueden especificar varios vectores de esquinas a la vez de manera que, por ejemplo, podríamos dibujar un anillo cuadrado.

- **strokeColor**: Color del borde del polígono expresado en notación hexadecimal. En el ejemplo, este atributo se establece a "#ff0000" (rojo).

- **strokeWeight**: Anchura del borde del polígono expresada en píxeles. En el ejemplo, este atributo se establece a "10".

- **strokeOpacity**: Opacidad de borde del polígono expresada como un valor entre 0,0 y 1,0. 0,0 es totalmente transparente, y 1,0, totalmente opaca. En el ejemplo, "0.5" establece una opacidad del 50%.

- **fillColor**: Color de relleno del polígono expresado en notación hexadecimal. En el ejemplo, este atributo se establece a "#ff0000" (rojo).

- **fillOpacity**: Opacidad del relleno del polígono expresada como un valor entre 0,0 y 1,0. 0,0 es totalmente transparente, y 1.0, totalmente opaco. En el ejemplo, "0.2" establece una opacidad del 20%.

- **map**: Mapa sobre el que se dibujará la línea.

Cálculo de áreas y longitudes

En los apartados precedentes hemos visto como dibujar polígonos y polilíneas. En éste retomaremos los ejemplos anteriores para mostrar como calcular la longitud y el área de los objetos representados.

Antes de empezar, es necesario que sepáis que Google Maps no carga por defecto la librería [6] que permite realizar estos cálculos. La librería en cuestión se llama "geometry" y debe cargarse manualmente.

¿Os acordáis que en el apartado Crear un mapa explicamos que para acceder a la API de Google Maps era necesario Incluir la API de Google Maps? Para hacerlo, incorporábamos las siguientes líneas en nuestro código:

```
<script type="text/javascript"
    src="http://maps.googleapis.com/maps/api/js?sensor=false">
</script>
```

También comentamos que, modificándolas apropiadamente, se podía acceder a otras capacidades de Google Maps, entre ellas las librerías.

Los ejemplos de este capítulo utilizan las funciones de la librería "geometry". Para que funcionen correctamente, es necesario modificar las líneas anteriores de la manera siguiente:

```
<script type="text/javascript"
        src="http://maps.googleapis.com/maps/api/js?libraries=geometry&sensor=false">
```

Observad que tan solo se ha agregado el parámetro "libraries=geometry".

Longitud de una polilínea

En el ejemplo siguiente se calcula la longitud en metros de la ruta París - Moscú - Pequín que ya habíamos dibujado anteriormente:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>París - Moscú - Pequín</title>
<script type="text/javascript"
        src="http://maps.googleapis.com/maps/api/js?libraries=geometry&sensor=false">
</script>
<script type="text/javascript">
function InicializarMapa() {
        // establece las opciones y crea el mapa
        var opcionesMapa = {
                center: new google.maps.LatLng(47.0, 59.0),
                zoom: 2,
                mapTypeId: google.maps.MapTypeId.TERRAIN
        };
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opcionesMapa);

        // coordenadas de París, Moscú y Pequín
        posicionParis = new google.maps.LatLng(48.856578,   2.351828);
        posicionMoscu = new google.maps.LatLng(55.752222,  37.615556);
        posicionPekin = new google.maps.LatLng(39.905556, 116.391389);

        // dibuja la ruta París - Moscú - Pequín, utilizando segmentos rectos
        // respecto a la proyección (polilínea estándar)
```

```
        var opcionesRutaParisPekin = {
                path: [ posicionParis, posicionMoscu, posicionPekin ],
                strokeColor: "#ff0000",
                strokeWeitght: 10,
                strokeOpacity: 0.5,
                geodesic: true,
                map: mapa
        };
        var rutaParisPekin = new google.maps.Polyline(opcionesRutaParisPekin);

        // calcula la longitud en metros de la ruta
        var longitud = google.maps.geometry.spherical.computeLength(rutaParisPekin.getPath());

        // muestra el longitud en km, redondeando el valor
        var opcionesVentana = {
                content: "Longitud total: " + Math.round(longitud / 1000) + " km",
                position: posicionPekin
        }
        var ventanaInfo = new google.maps.InfoWindow(opcionesVentana);

        ventanaInfo.open(mapa);
}
</script>
</head>
<body onload="InicializarMapa()">
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
</body>
</html>
```

El resultado es el siguiente:

Como se puede observar, la longitud de la ruta la calculamos mediante la función "google.maps.geometry.spherical.computeLength()", que, atención!, espera como parámetro un vector de objetos "LatLng" que representa la secuencia (ordenada) de vertices de la polilínea. Es importante tener claro que la función no espera un objeto "Polyline" sino solo sus vertices, Afortunadamente, obtener un vector con los vértices de una polilínea es tan fácil como llamar a la función "getPath()" de la clase "Polyline":

```
var longitud = google.maps.geometry.spherical.computeLength(rutaParisPekin.getPath());
```

Cabe destacar que la longitud obtenida corresponde a la suma de las longitudes de las líneas geodésicas que unen los vértices adyacientes de la polilínea. Es decir, el número que obtenemos és la distancia mínima real (sobre la superfície terrestre) que se debe recorrer para ir des del origen al fin de la ruta pasando por todos los puntos de paso intermedios.

Es importante entender que la longitud de una polilínea depende única y exclusivamente de la posición de sus vértices. Esto significa que, aunque representemos la polilínea mediante segmentos rectos respecto a la proyección (recordad: especificando la opción "geodesic: false" o, simplemente, obviando esta opción) la longitud de la polilínea no cambiará.

El resto del código no presenta sorpresas. Tan solo merece ser destacado el uso de la función "Math.round()" que redondea un número decimal al número entero más próximo. En el ejemplo se utiliza para redondear la distancia calculada.

Área de un polígono

En el ejemplo siguiente se calcula el área de la Plaça Catalunya de Barcelona cuya superfície ya habíamos delimitado con un polígono anteriormente:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Área de la Plaça de Catalunya de Barcelona</title>
<script type="text/javascript"
        src="http://maps.googleapis.com/maps/api/js?libraries=geometry&sensor=false">
</script>
<script type="text/javascript">
function InicializarMapa() {
        // establece las opciones y crea el mapa
        var opcionesMapa = {
                center: new google.maps.LatLng(41.387024, 2.169960),
                zoom: 16,
                mapTypeId: google.maps.MapTypeId.ROADMAP
        };
        var mapa = new google.maps.Map(document.getElementById("elemento_mapa"), opcionesMapa);

        var esquinasPlaza = [
                new google.maps.LatLng(41.387877, 2.169971),
                new google.maps.LatLng(41.387418, 2.168748),
                new google.maps.LatLng(41.385856, 2.170143),
                new google.maps.LatLng(41.386919, 2.171205)
        ];

        // delimita el contorno de la plaza
        var contornoPlaza = new google.maps.Polygon({
                paths: esquinasPlaza,
                strokeColor: "#ff0000", // color del contorno
                strokeOpacity: 0.5,     // opacidad del contorno
                strokeWeight: 10,       // ancho del contorno
                fillColor: "#ff0000",   // color de relleno
                fillOpacity: 0.2,       // opacidad del relleno
                map: mapa               // mapa sobre el que se debe dibujar el polígono
        });

        // calcula el área en metros cuadrados del polígono
        var area = google.maps.geometry.spherical.computeArea(contornoPlaza.getPath());

        // muestra el resultado
        var opcionesVentana = {
                content: "Área total: " + Math.round(area) + " m<sup>2</sup>",
                position: mapa.getCenter()
        }
        var ventanaInfo = new google.maps.InfoWindow(opcionesVentana);

        ventanaInfo.open(mapa);
}
</script>
</head>
<body onload="InicializarMapa()">
<div id="elemento_mapa" style="width: 500px; height: 300px"></div>
</body>
</html>
```

El resultado es el siguiente:

Como se puede observar, el área del polígono la calculamos mediante la función "google.maps.geometry.spherical.computeArea()", de forma muy parecida a como calculábamos la longitud de una polilínea. En este caso, la función también espera un vector de objetos "LatLng" que representa la secuencia (ordenada) de vertices del polígono, no el objeto "Polygon" en sí.

Resumen

Después de aprender a programar aplicaciones SIG de escritorio, en este módulo se ha hecho lo propio con aplicaciones web. Si en el caso de las aplicaciones de escritorio nos hemos basado en un sistema ya implementado (gvSIG), con las aplicaciones web hemos hecho algo parecido: hemos utilizado la API pública de Google Maps.

Antes de abordar la estructura y las funciones de la API, hemos aprendido a crear una página web simple y a incorporarle un mapa. Después, hemos utilizado las funciones de la API para personalizar la presentación del mapa (coordenadas iniciales, zoom y tipo de mapa) y representar puntos, y hemos aprendido a transformar las coordenadas del mapa a coordenadas geográficas.

Finalmente, se han estudiado las funciones que permiten dibujar líneas y polígonos, caracterizando rutas y áreas que, más tarde, mediante funciones dedicadas, hemos podido medir.

Aunque todos los ejemplos están basados en la API de Google Maps, las técnicas explicadas en este módulo se pueden extrapolar más o menos directamente a otras API, libres o comerciales.

Notas

1. ? Pere Barnola Augé, *Introducción a la creación de páginas web*, Septiembre 2008, http://mosaic.uoc.edu/2009/06/20/introduccion-a-la-creacion-de-paginas-web.
2. ? La documentación completa de la API de Google Maps se puede encontrar en: http://code.google.com/intl/es/apis/maps/documentation/javascript.
3. ? Una polilínea es una serie de segmentos conectados.
4. ? Este ejemplo sólo pretende mostrar cómo trazar líneas con un objeto "Polyline". Google Maps tiene mecanismos propios para trazar rutas entre dos puntos automáticamente. Si nuestro objetivo es obtener una ruta por carretera para ir de una dirección a otra, deberíamos usar la clase "DirectionsService".
5. ? El código de color se puede obtener de cualquier herramienta de dibujo o edición fotográfica. También hay paletas de colores disponibles en la Web (por ejemplo, en Wikipedia: http://es.wikipedia.org/wiki/Colores_HTML#Tabla_de_colores).
6. ? Una librería es un conjunto de funciones que se ofrecen a un programa. Las librerías son independientes de los programas y normalmente se desarrollan por separado.