IIC2026
Visualización de Información

Hernán F. Valdivieso López
(2023 - 1 / Clase 11)

Temas de la clase - Zoom y panning

1. Zoom en D3
2. Configurar el panning y la intuición de cómo funciona.
3. Aplicar zoom a una visualización.
4. Controlando el zoom desde el código.
5. Otra forma de aplicar zoom

Zoom en D3

Zoom en D3

D3 provee del método `d3.zoom()` para gestionar la navegación (*zoom* y *panning*).

Este método genera 3 eventos que podemos vincular con nuestras funciones:

- **`start`**: se gatilla cuando empezamos la acción de zoom o panning. Solo hacer click ya gatilla este evento.
- **`zoom`**: se gatilla mientras se hace la acción de zoom o panning. Por ejemplo, arrastrar el mouse o después de hacer doble click se llamará a este evento.
- **`end`**: se gatilla cuando finaliza la acción de zoom o panning. Por ejemplo, dejar de arrastrar el mouse o después de hacer doble click y haber ejecutado el evento zoom se llamará a este evento.

Zoom en D3

Adicionalmente, d3.zoom() provee de 3 parámetros:

- **`scaleExtent`**: permite fijar que tanto nos podemos alejar o acercar a la visualización (*zoom*).
- **`extent`**: defina el tamaño de nuestra cámara para poder moverla (*panning*).
- **`translateExtent`**: define la región máxima donde se puede mover nuestra cámara (*panning*).

Zoom en D3

Formato

```
const zoom = d3.zoom()
  .scaleExtent([0.5, 2]) // despues veremos extent y translateExtent
  .on("start", () => console.log("empecé"))
  .on("zoom", () => console.log("moviendo"))
  .on("end", () => console.log("terminé"));

// Conectamos el objeto zoom con nuestro SVG
d3.select("svg").call(zoom)
```

Zoom en D3

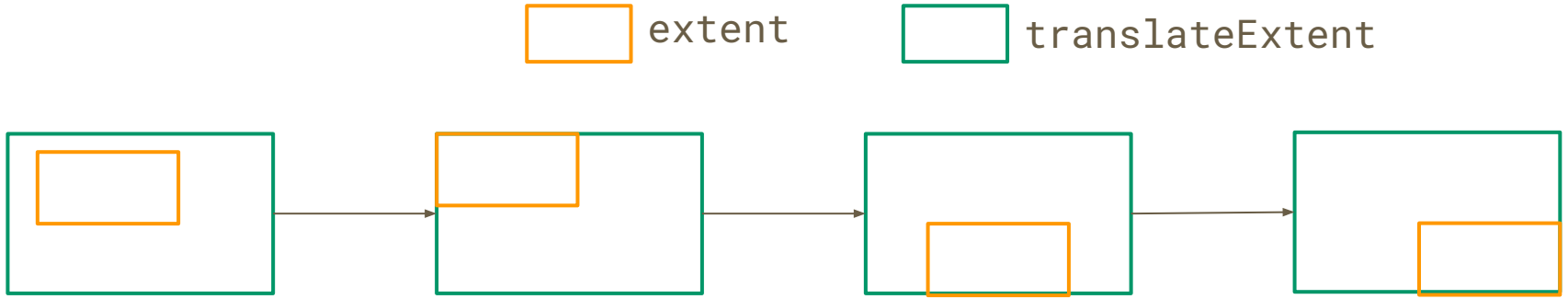
Vamos al código 👨‍💻 👩‍💻

Configurar el panning y la intuición de cómo funciona

Panning

Adicionalmente, d3.zoom() provee de 3 parámetros:

- **`scaleExtent`**: permite fijar que tanto nos podemos alejar o acercar a la visualización (zoom).
- **`extent`**: defina el tamaño de nuestra cámara para poder moverla (panning).
- **`translateExtent`**: define la región máxima donde se puede mover nuestra cámara (panning).



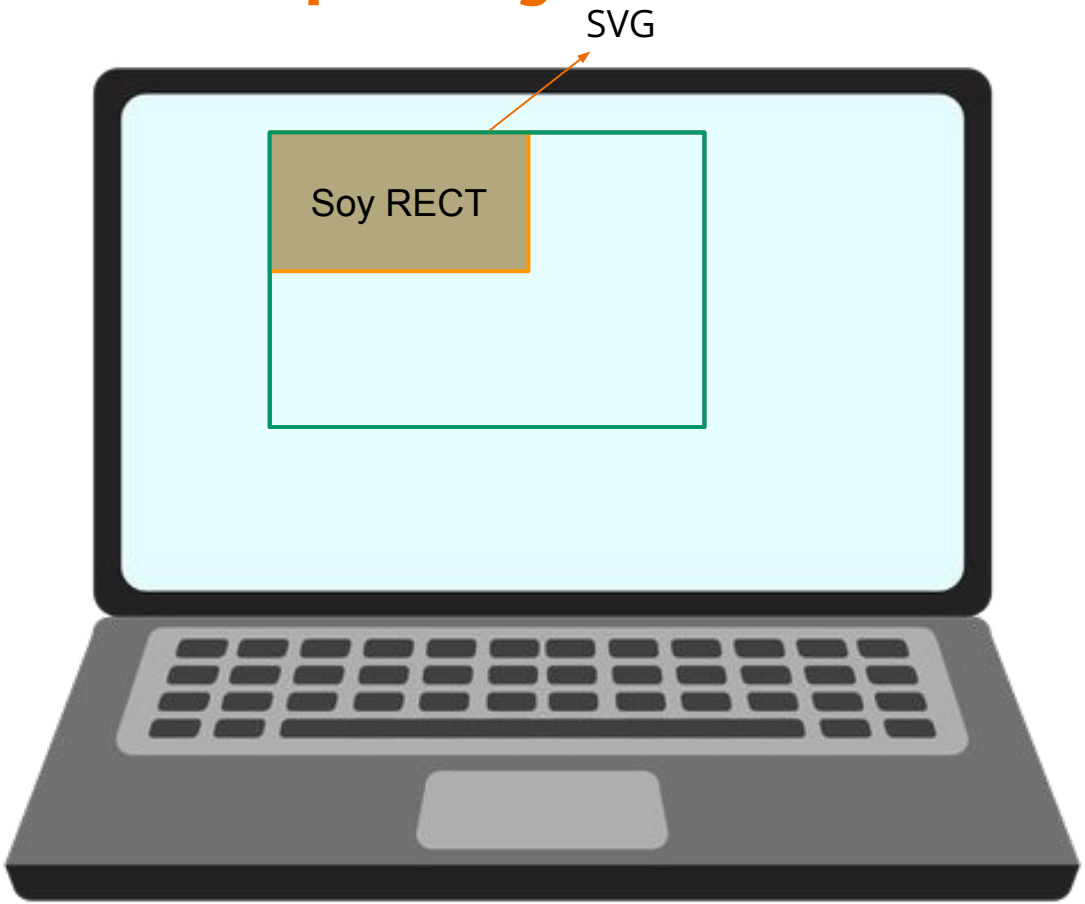
Panning

Formato

```
const zoom = d3.zoom()
  .scaleExtent([0.5, 2])
  .extent([[0, 0], [WIDTH, HEIGHT]])
  .translateExtent([[0, 0], [WIDTH, HEIGHT]])
  .on("start", () => console.log("empecé"))
  .on("zoom", () => console.log("moviendo"))
  .on("end", () => console.log("terminé"));


// Conectamos el objeto zoom con nuestro SVG
d3.select("svg").call(zoom)
```

Intuición del panning

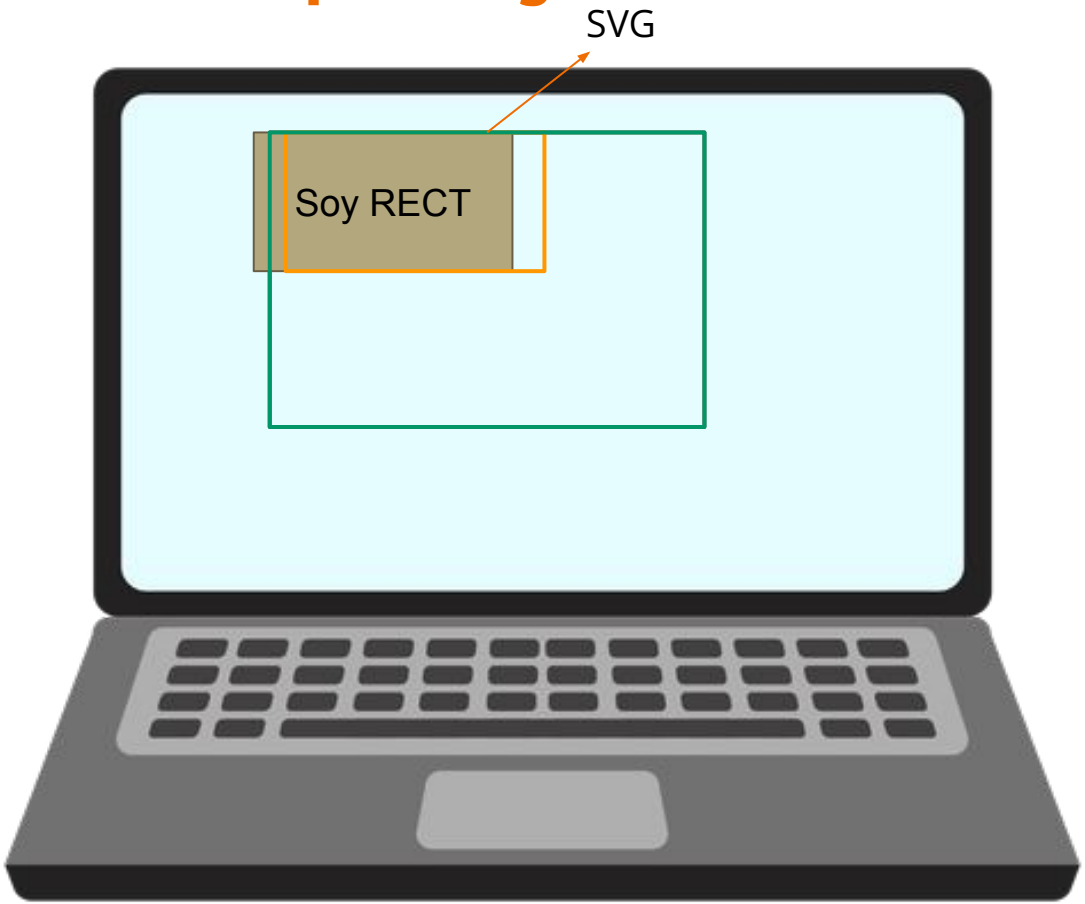


`extent`

`translateExtent`

Hacemos `extent` igual al tamaño del rect y `translateExtent` al tamaño del SVG.

Intuición del panning



`extent`

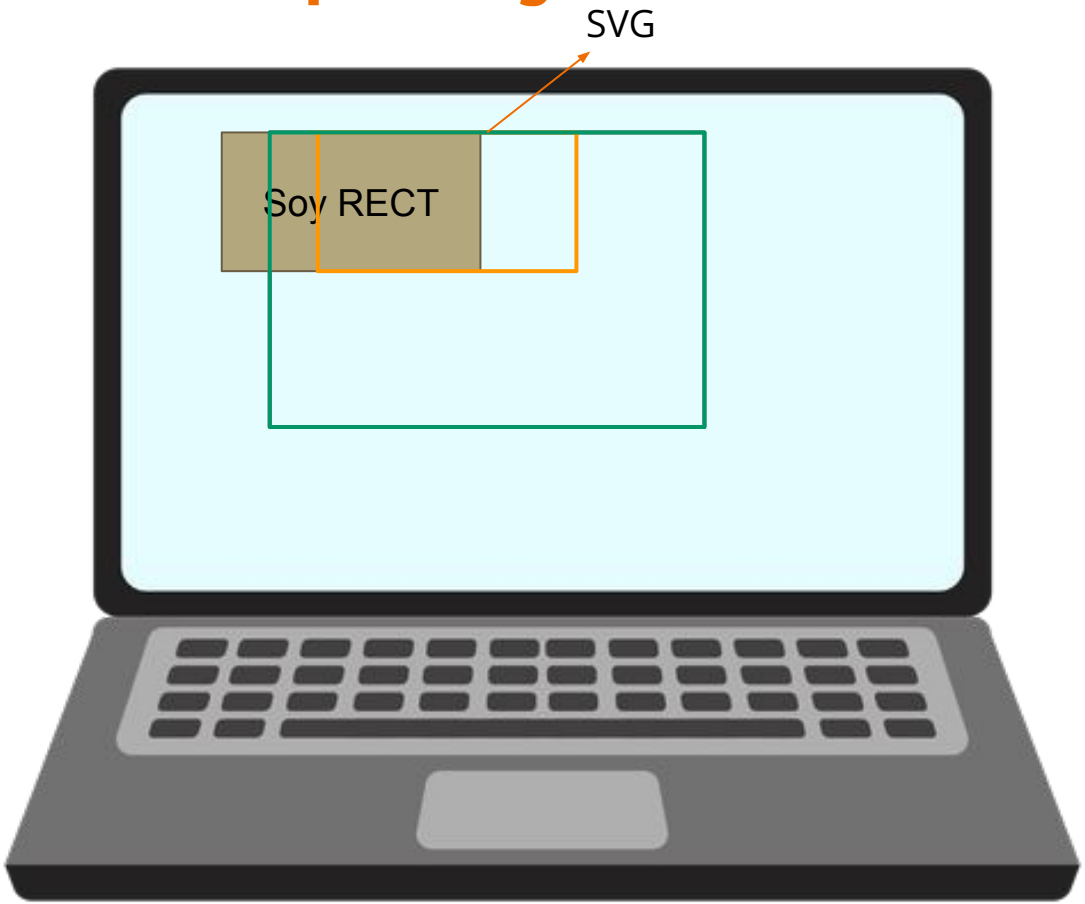
`translateExtent`

Arrastramos el mouse a la derecha mientras lo presionamos.

Eso genera un evento que nos entrega la información para trasladar el rect.

El cuadro de extent también ajusta su posición como si se estuviera alejando del rect.

Intuición del panning



`extent`

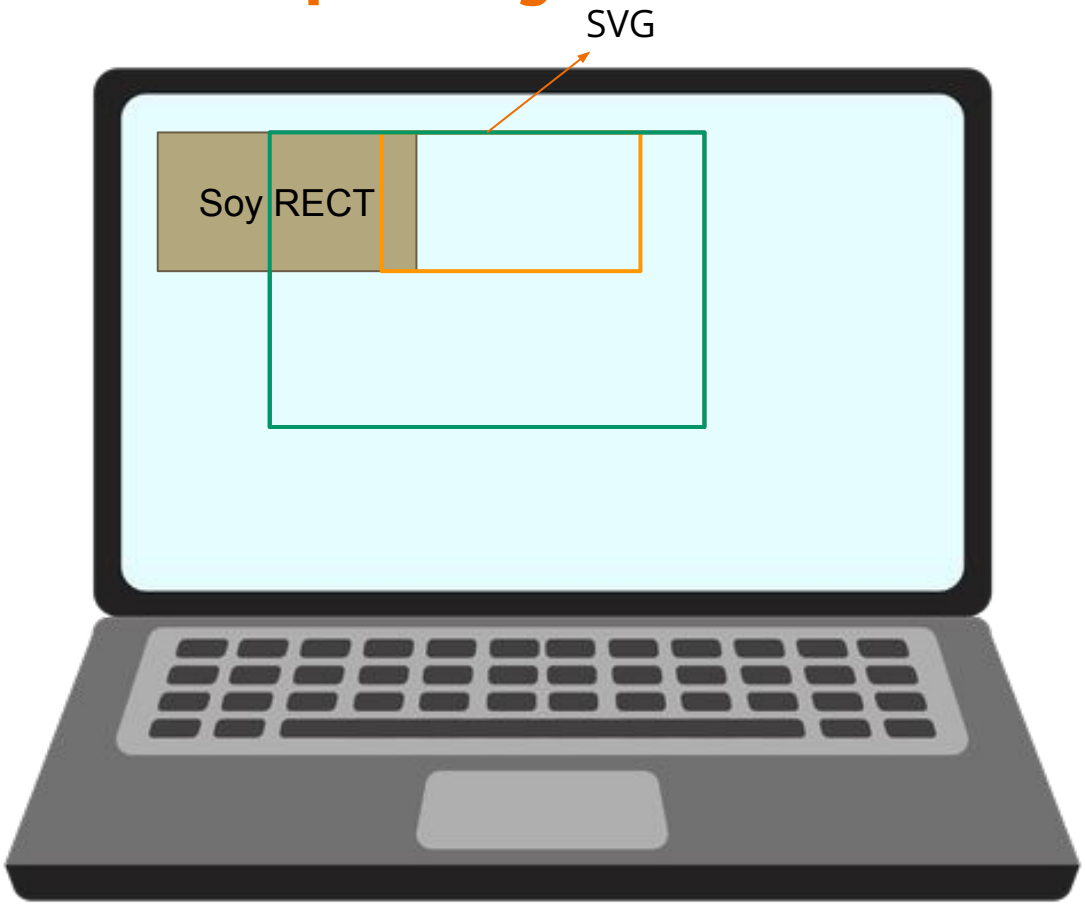
`translateExtent`

Arrastramos el mouse a la derecha mientras lo presionamos.

Eso genera un evento que nos entrega la información para trasladar el rect.

El cuadro de extent también ajusta su posición como si se estuviera alejando del rect.

Intuición del panning



extent

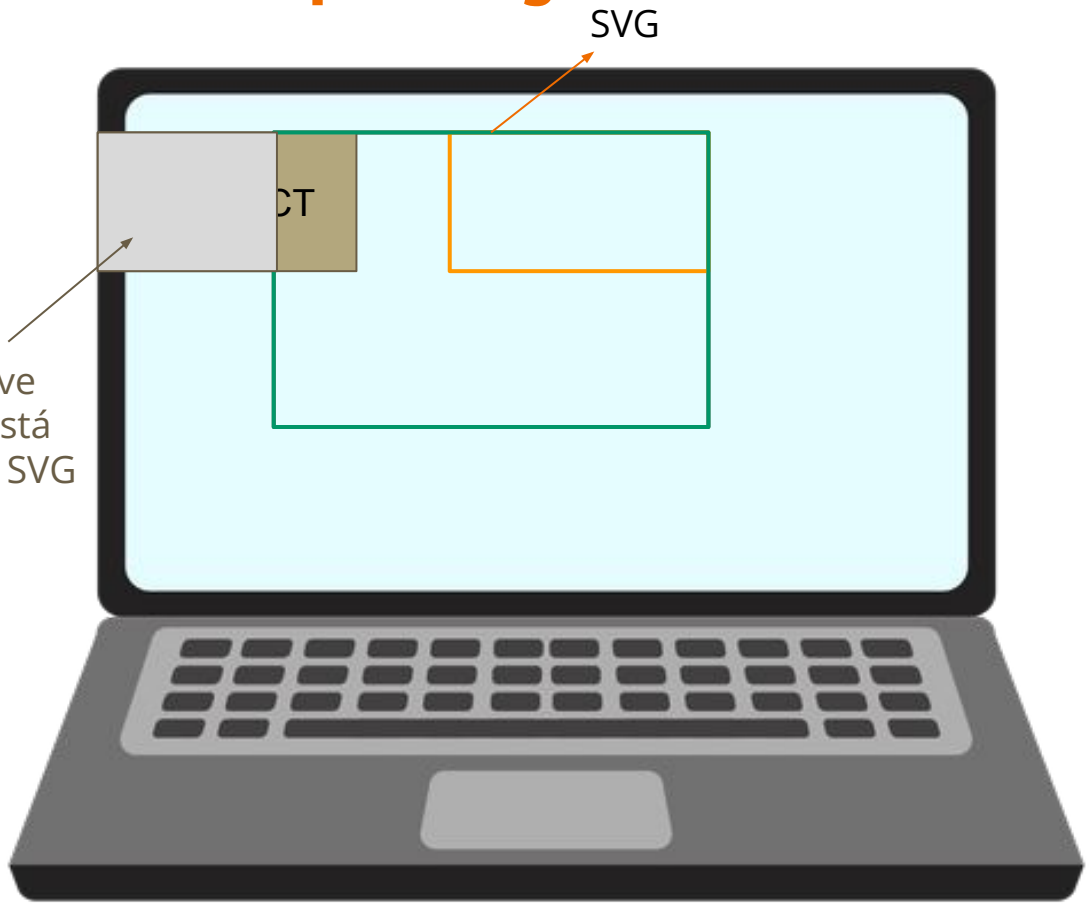
translateExtent

Arrastramos el mouse a la derecha mientras lo presionamos.

Eso genera un evento que nos entrega la información para trasladar el rect.

El cuadro de extent también ajusta su posición como si se estuviera alejando del rect.

Intuición del panning



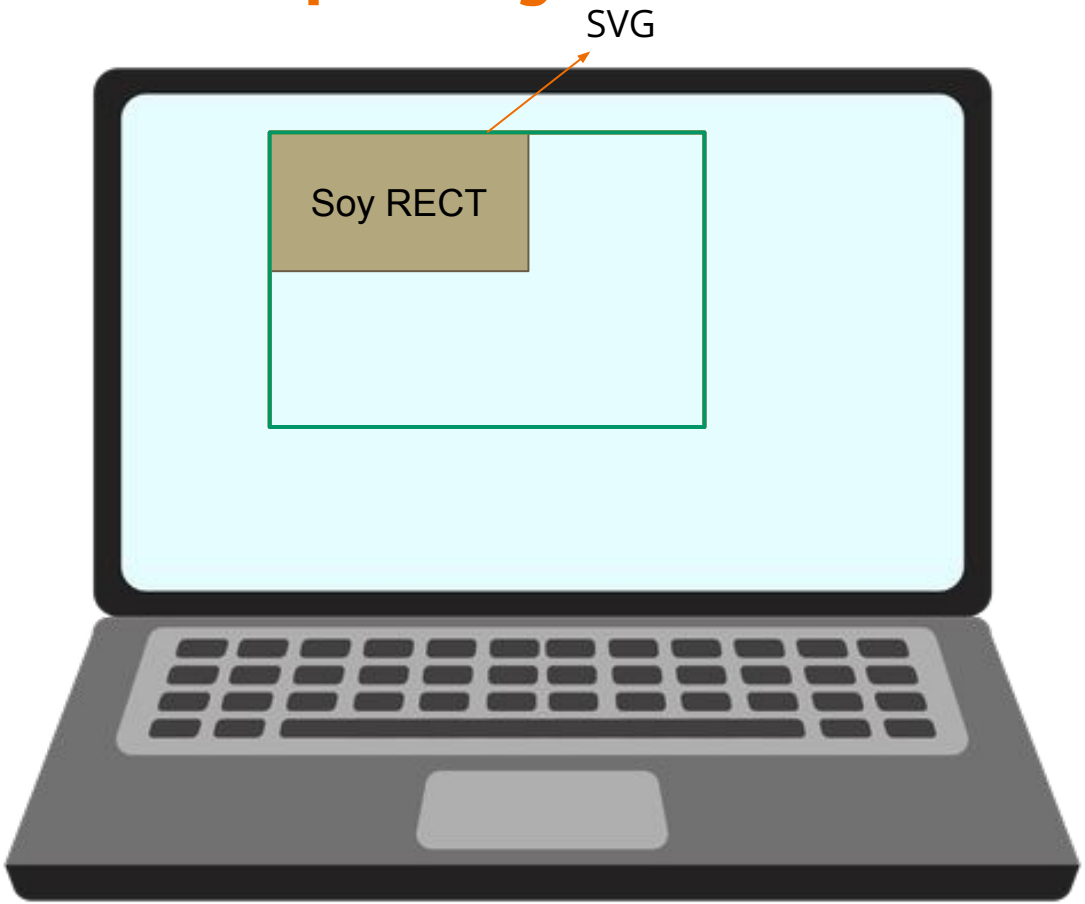
Ya no se ve porque está fuera del SVG

`extent`

`translateExtent`

Ya no se puede arrastrar el rect porque el cuadro de `extent` chocó con un borde de `translateExtent`

Intuición del panning



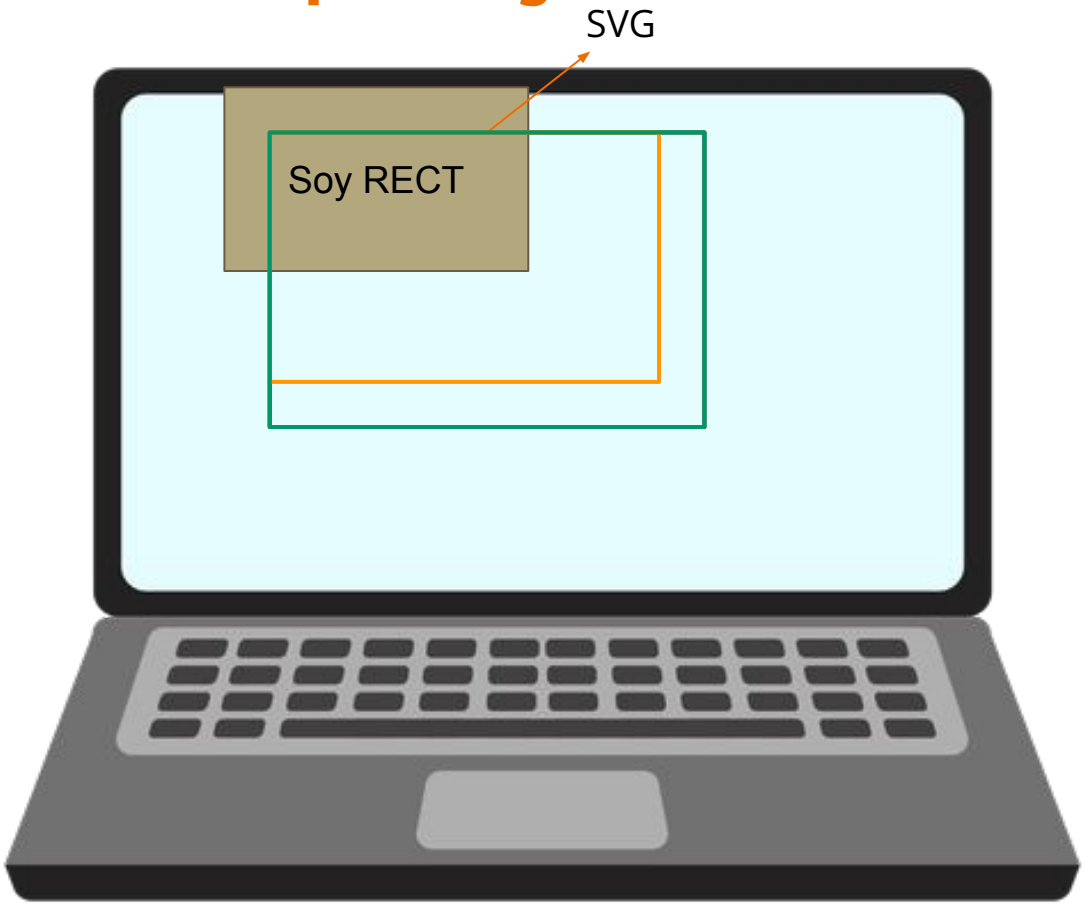
`extent`

`translateExtent`

Lo estándar es hacer que `extent` y `translateExtent` sean iguales.

Además, ambos deben ser del tamaño del SVG.

Intuición del panning + zoom



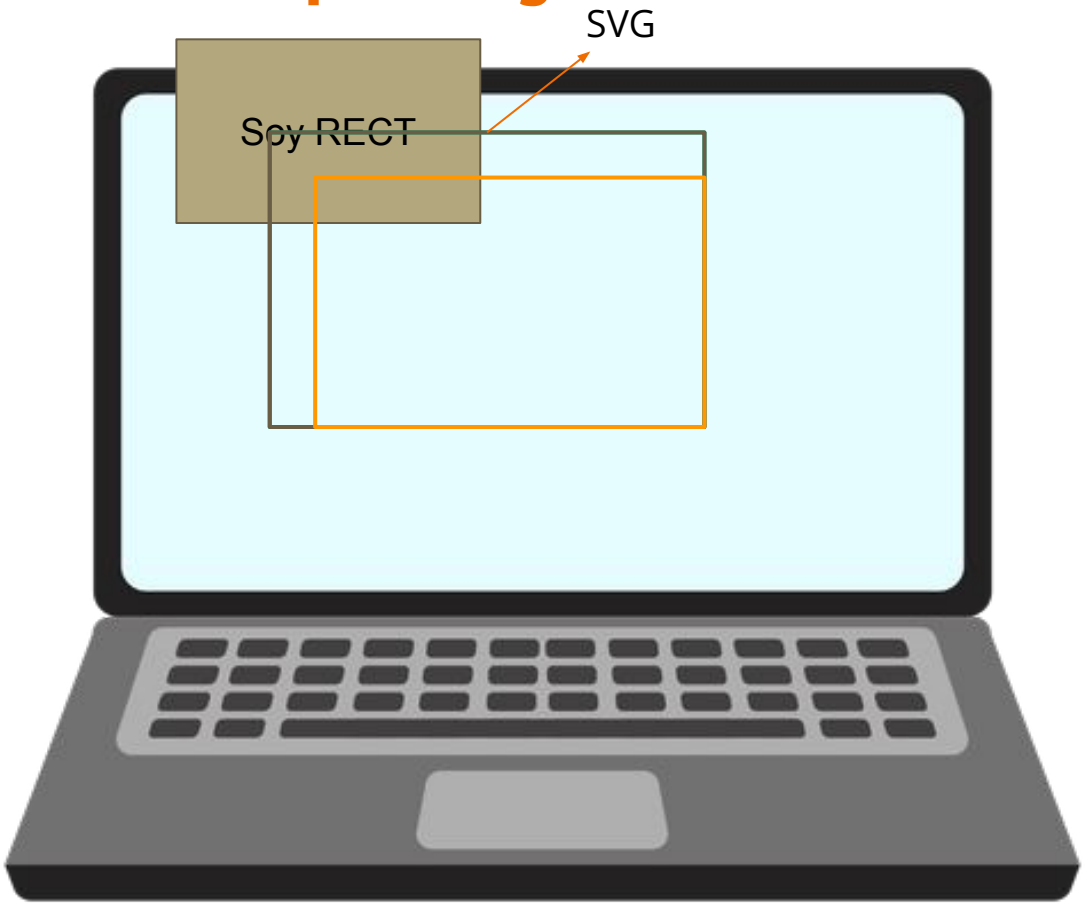
`extent`

`translateExtent`

Si hacemos *zoom*, nuestro cuadro de `extent` se achica simulando como que acercamos la cámara.

¡Ahora sí podemos hacer *panning*!

Intuición del panning + zoom



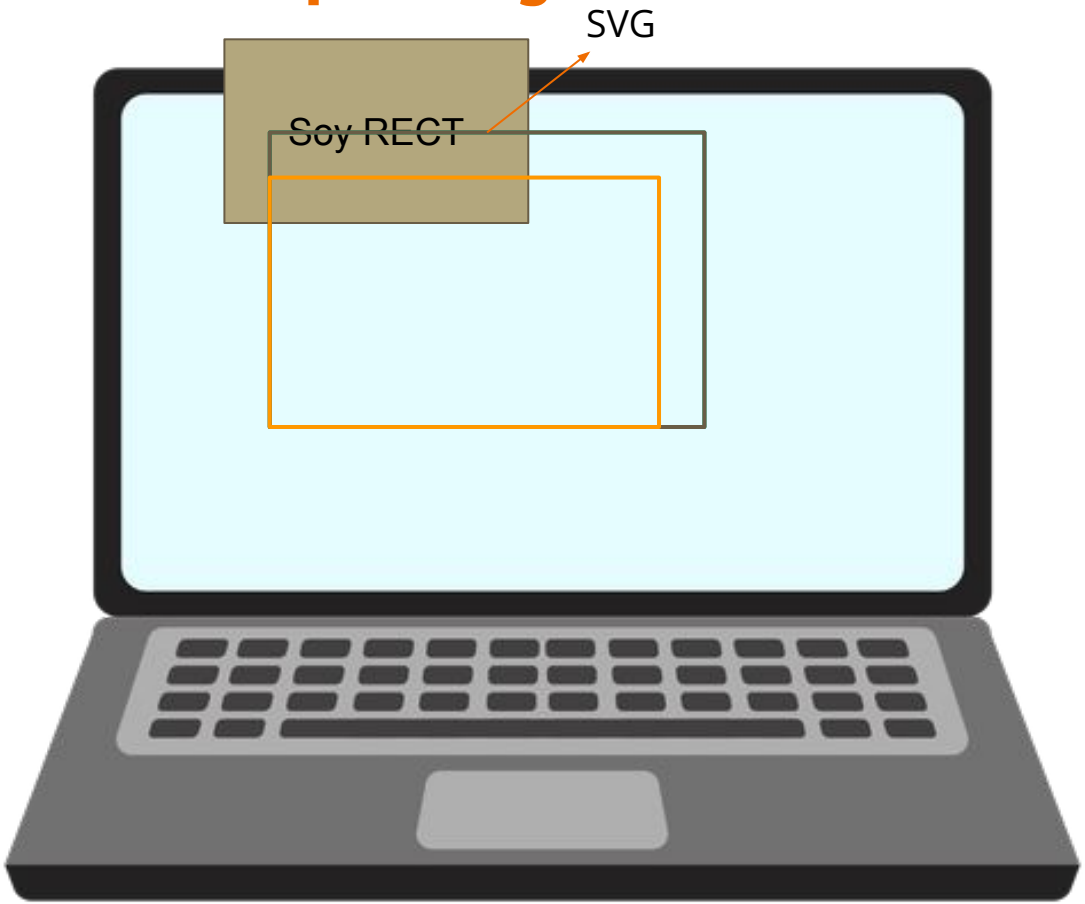
`extent`

`translateExtent`

Si hacemos *zoom*, nuestro cuadro de `extent` se achica simulando como que acercamos la cámara.

¡Ahora sí podemos hacer *panning*!

Intuición del panning + zoom



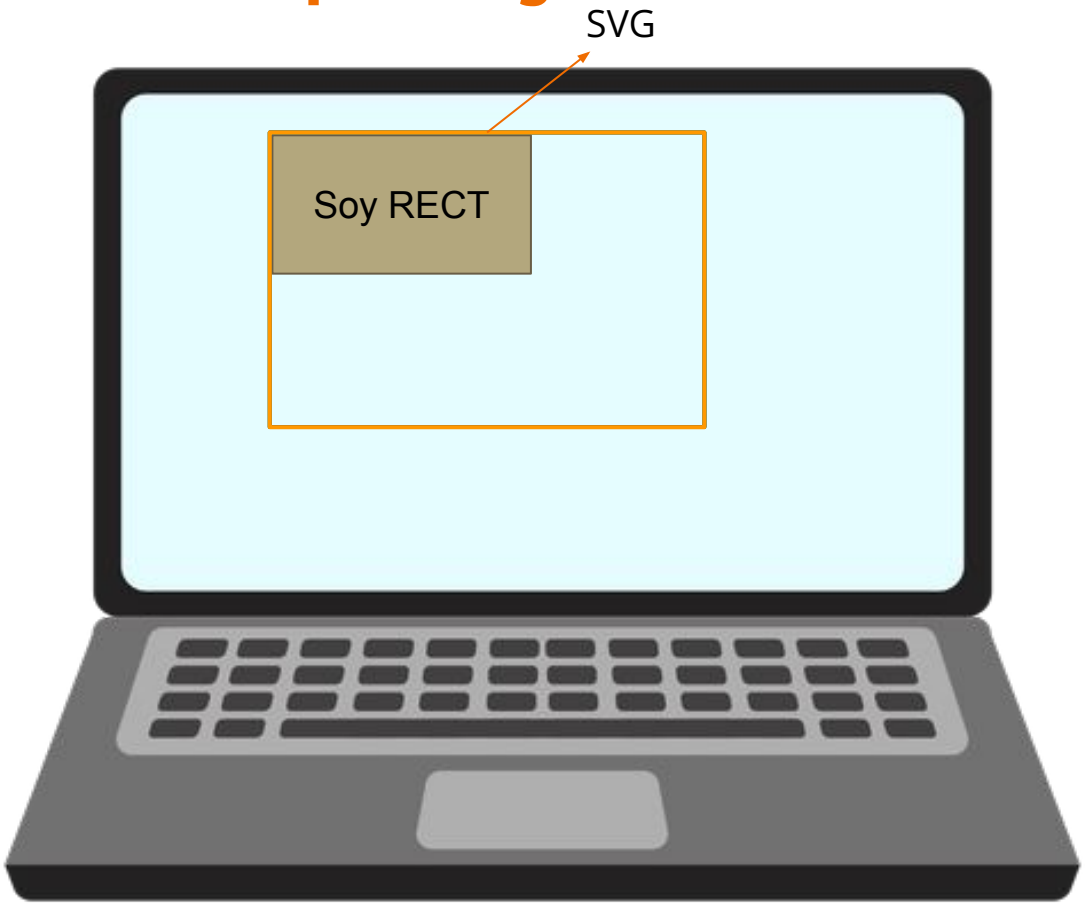
`extent`

`translateExtent`

Si hacemos *zoom*, nuestro cuadro de `extent` se achica simulando como que acercamos la cámara.

¡Ahora sí podemos hacer *panning*!

Intuición del panning + zoom



`extent`

`translateExtent`

Cuando deshacemos el *zoom*, el cuadro de `extent` vuelve a ser igual al de `translateExtent` y la figura vuelve a su tamaño normal.

Panning + zoom

Vamos al código 👨‍💻 👩‍💻

Aplicar zoom a una visualización

Aplicar zoom a una visualización

Vamos al código 👨‍💻 👩‍💻

Aplicar zoom a una visualización

Resumen

1. Definir objeto zoom y función encargada de actualizar los elementos visuales cuando se gatilla el evento zoom.
2. Reescalar lor ejes (`const escalaX2 = transformacion.rescaleX(escalaX)`).
3. Actualizar ejes (`contenedorEjeX.call(ejeX.scale(escalaX2))`).
4. Definir `clipPath` para solo ver elementos dentro de nuestra región definida.

```
svg
  .append("clipPath")
  .attr("id", "clip")
  .append("rect")
  .attr("width", WIDTHVIS)
  .attr("height", HEIGHTVIS);
```

```
const contenedorVis = svg
  .append("g")
  .attr("transform", `translate(...)`)
  .attr("clip-path", "url(#clip)");
```

Controlando el zoom desde el código

Controlando el zoom desde el código

- Un problema que puede ocurrir al permitir navegación, es que sea difícil volver al estado inicial o punto de vista.
- Otro problema es esperar que el usuario navegue a una zona en particular.
- Surge la necesidad de proveer un medio para controlar la navegación desde código. Por ejemplo:
    - Apretar un botón permite al zoom volver a su estado inicial.
    - Apretar un botón haga que automáticamente la visualización navegue a un punto de interés.

Controlando el zoom desde el código

🤔 ¿Qué acciones podemos programar?

- `(zoom.translateBy, X, Y)` → Traslada la transformación actual en X unidades e Y unidades.
- `(zoom.translateTo, X, Y)` → Posiciona la transformación en (X,Y)
- `(zoom.scaleBy, K)` → Multiplica la escala actual de la transformación K.
- `(zoom.scaleTo, K)` → Setea la escala de la transformación en K
- `(zoom.transform, transformacion)` → reemplaza la transformación actual por la nueva transformación. Se debe utilizar `d3.zoomIdentity` para crear el objeto transformación.

Controlando el zoom desde el código

Ejemplos:

1. Volver al inicio de la navegación

```
// Obtenemos una transformación identidad (x=0, y=0, k=1)
const transformacion = d3.zoomIdentity;
svg.call(zoom.transform, transformacion);
```

2. Trasladar 20 unidades el zoom a la derecha y 10 unidades hacia abajo.

```
svg.transition().call(zoom.translateBy, -20, -10);
```

4. Duplicar el zoom actual

```
svg.transition().call(zoom.scaleBy, 2);
```

Controlando el zoom desde el código

Vamos al código 👨‍💻 👩‍💻

Otra forma de aplicar zoom

Otra forma de aplicar zoom

- Actualmente usamos `"transform"` para aplicar cambios a nuestros puntos.
- 🤔 ¿Si queremos sólo hacer *zoom* en el eje X? ¿si no queremos cambiar su tamaño?
    - Ya no podemos ocupamos `.attr("transform", tranformacion)` 😓

Solución: Actualizaremos la posición de los puntos usando la nueva escala.

Otra forma de aplicar zoom

```
// Ajustamos escalas. Esta función solo sirve con escalas continuas.
const escalaX2 = transformacion.rescaleX(escalaX);

// Actualizamos posición en X.
puntos.attr("cx", (d) => escalaX2(d.x));

// Actualizamos el ejeX
contenedorEjeX.call(ejeX.scale(escalaX2));
```

Otra forma de aplicar zoom

Vamos al código 👨‍💻 👩‍💻

Otra forma de aplicar zoom 2

🤔 ¿Si nuestra escala no es continua y además solo queremos hacer zoom en un eje?

- Ya no podemos ocupamos `.attr("transform", tranformacion)` 😓
- Tampoco podemos ocupar `transformacion.rescaleX(escalaX)` 😓

Solución:

```
const transformacion = evento.transform;
// Actualizamos el rango de la escala considerando la transformación realizada.
escalaX.range([transformacion.applyX(0), transformacion.applyX(WIDTHVIS)])
```

Con eso actualizamos el rango de una escala a mano y luego podemos actualizar la posición de los elementos.

En la ayudantía de *zoom* verán un ejemplo de esta otra forma 😁 .

Próximos eventos

Próxima clase

- Facetas y Reducción (teoría).

Ayudantía de la próxima semana

- Sala de ayuda T3.
- Aprovechar de trabajar y preguntar al profesor y ayudantes.

IIC2026
Visualización de Información

Hernán F. Valdivieso López
(2023 - 1 / Clase 11)