ChronoZoom® beta

Architecture Guide

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

Contents

Disclaimer 4
Introduction 4
Virtual Canvas 5
Implementation 7
Viewport 9
Content objects 9
Loading content 10
Search 10
Bookmarks 11
Gestures 12
Zoom Operation 12
Pan Operation 12
Mouse Gestures 12
Keyboard Gestures 13
Multitouch Gestures 13
Math 13
Implementation 13
Layout Algorithm 15
Axis Introduction 16
Ticks layout algorithms. 17
Year-zero problem 18
Axis behavior 18
Thresholds 18
Tours 19
Tour related components 19
An audio narration 19
Tour Bookmarks 20
Tour bookmarks transitions 20
Tour control API 20

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

1st Party Authoring 22
The Future of ChronoZoom 22

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

Disclaimer

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2012 The Outercurve Foundation.

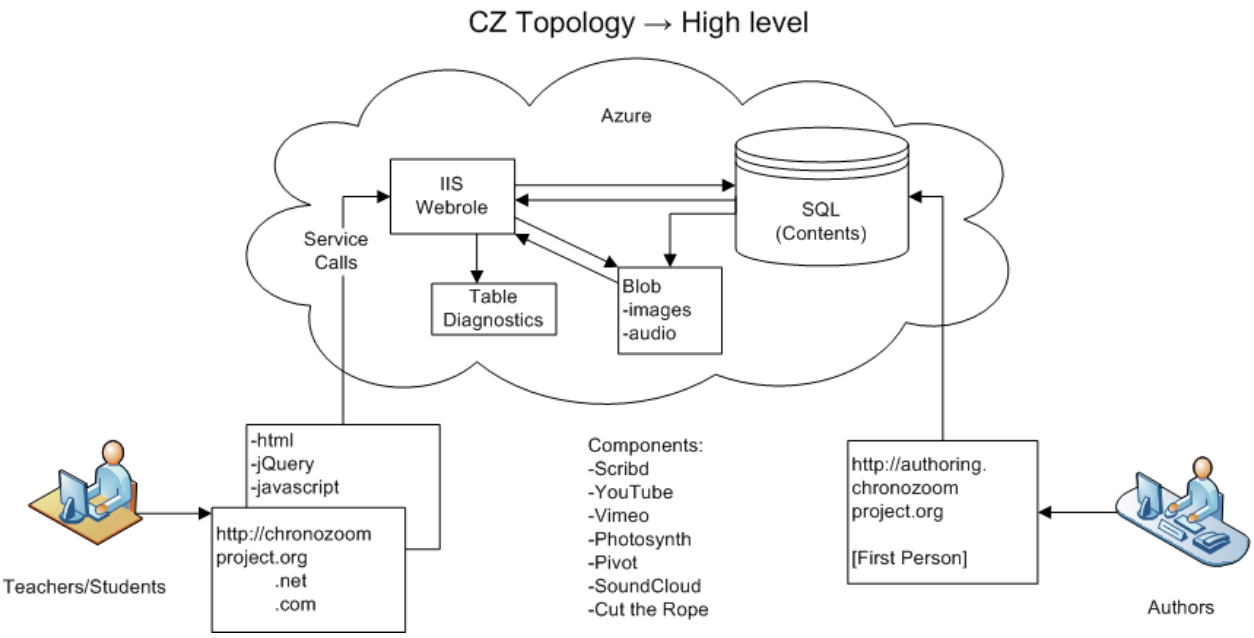
Distributed under Creative Commons Attribution 3.0 Unported License.

Microsoft, Visual Studio, and Windows are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

Introduction

ChronoZoom is an open source community project dedicated to visualizing Big History that has been funded and supported by Microsoft Research Connections in collaboration with University California at Berkeley and Moscow State University.

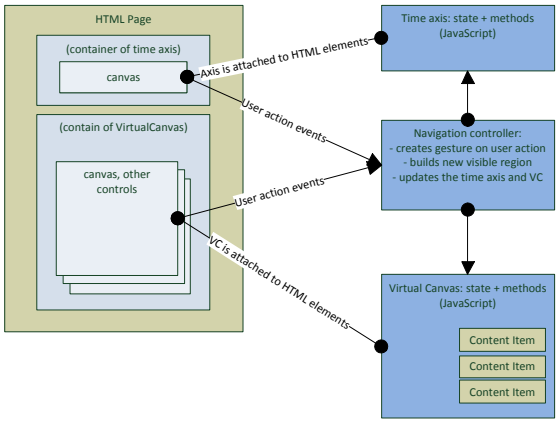
This document is a guide to the architecture of ChronoZoom.



© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

Virtual Canvas

Virtual canvas represents $\mathbb{R}^2$ space and a hierarchical set of objects placed in that space. A rectangular subset of a virtual canvas space, called visible region, can be projected to a screen rectangle; this projection defines a viewport of a virtual canvas.



The space includes a coordinate system with units of measurements: t along x-axis, and $\hbar - unit$ along y-axis. A parameter of the virtual space is an aspect ratio $\gamma\ [\hbar/t]$, which says how much $\hbar$ units is in one t unit; in other words, a rectangle of size $1 \times \gamma$ should look as square in the virtual space. Aspect ratio is a constant for each virtual canvas.

We use superscript $*^v$ to say that $p^v = (x^v, y^v) \in \mathbb{R}^2$ is a point in the virtual space coordinate system. The superscript $*^p$ indicates that screen coordinates in pixels are used. Units of a value are provided is square brackets; e.g. $x[t]$ indicates that value x is measured in t.

A *visible region* $\mathcal{V}$, which is a rectangle within the virtual space:

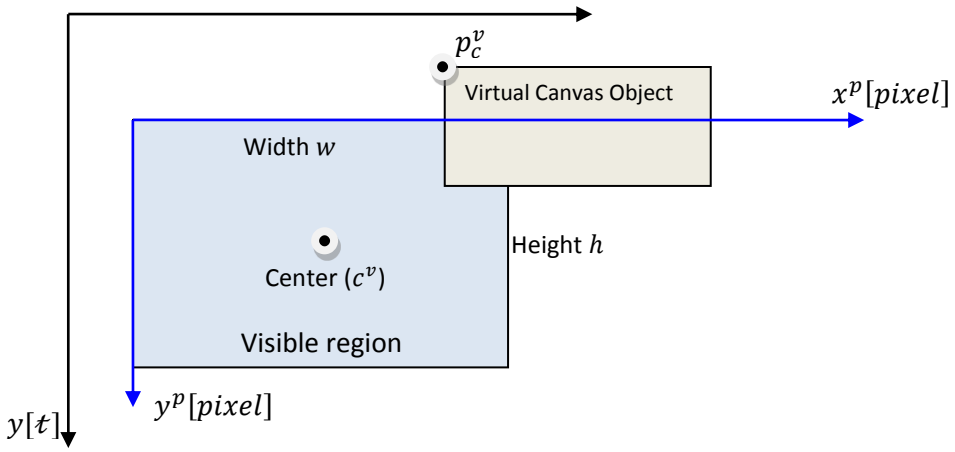
$$\mathcal{V} = (c^v, w^p, h^p, \alpha),$$

where $c^v = \left(c_x^v, c_y^v\right)$ is a center point of the $\mathcal{V}$ in virtual coordinates, $w^p \times h^p$ is a size in pixels of the visible region (i.e. size of the physical canvas to draw), $\alpha\ [t/pixel]$ is a scale, which say how many t units in a single screen pixel.

The *screen coordinate system* has origin in the left-top corner of the $\mathcal{V}$, the x-asis is directed to the right, the y-axis is directed to the bottom.

x $[\hbar]$

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License



The next function gets the width and height of the $\mathcal{V}$ in pixel coordinates:

$$w^v[\hbar] = \alpha w^p, h^v[t] = \gamma\alpha h^p.$$

The next function transforms a point p^v into a screen coordinate system of the virtual canvas $\mathcal{V}$:

$$p_x^p(p_x^v; \mathcal{V}) = \frac{p_x^v - c_x^v}{\alpha} + \frac{w^p}{2},$$

$$p_y^p(p_y^v; \mathcal{V}) = \frac{p_y^v - c_y^v}{\gamma\alpha} + \frac{h^p}{2}.$$

And finally, $virtualToScreen$: $(p^v, \mathcal{V}) \to p^p$:

$$virtualToScreen\ (p^v, \mathcal{V}) = \left(p_x^p(p_x^v), p_y^p(p_y^v)\right).$$

Screen to virtual coordinates transformation can be inferred as reverse functions of the given above.

Virtual canvas content object $\mathcal{C}$ is an object defined in the virtual coordinates which can be rendered on a physical canvas. Its location is defined as

$$\mathcal{C} = (p_c^v, w_c^v, h_c^v),$$

where $p^v = \left(p_{cx}^v, p_{cy}^v\right)$ is a left-top corner of the content object in virtual coordinates, $w_c^v \times h_c^v$ is a size of the visual representation of the object in virtual coordinates.

Transformation of $\mathcal{C}$, attached to $\mathcal{V}$, from virtual space into screen coordinates of $\mathcal{V}$:

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

$$\mathcal{C}^p = \begin{pmatrix} virtualToScreen\ (p_c^v; \mathcal{V}), \\ w_c^p = \frac{w_c^v}{\alpha}, \\ h_c^p = \frac{h_c^v}{\gamma\alpha} \end{pmatrix}.$$

Thus, the size of $\mathcal{C}$ in pixels is $w_c^p \times h_c^p$.

The function determining visibility of an object is as follows:

1. Determine whether bounding rectangle of $\mathcal{C}$ intersects visible region $\mathcal{V}$. If not, it is not rendered.
2. If either w_c^p or h_c^p is less than a predefined threshold (say, 2 pixels), it is not rendered.
3. Otherwise, rendering the object on a canvas of the layer associated with this object.

Thus the virtual canvas can be defined as follows:

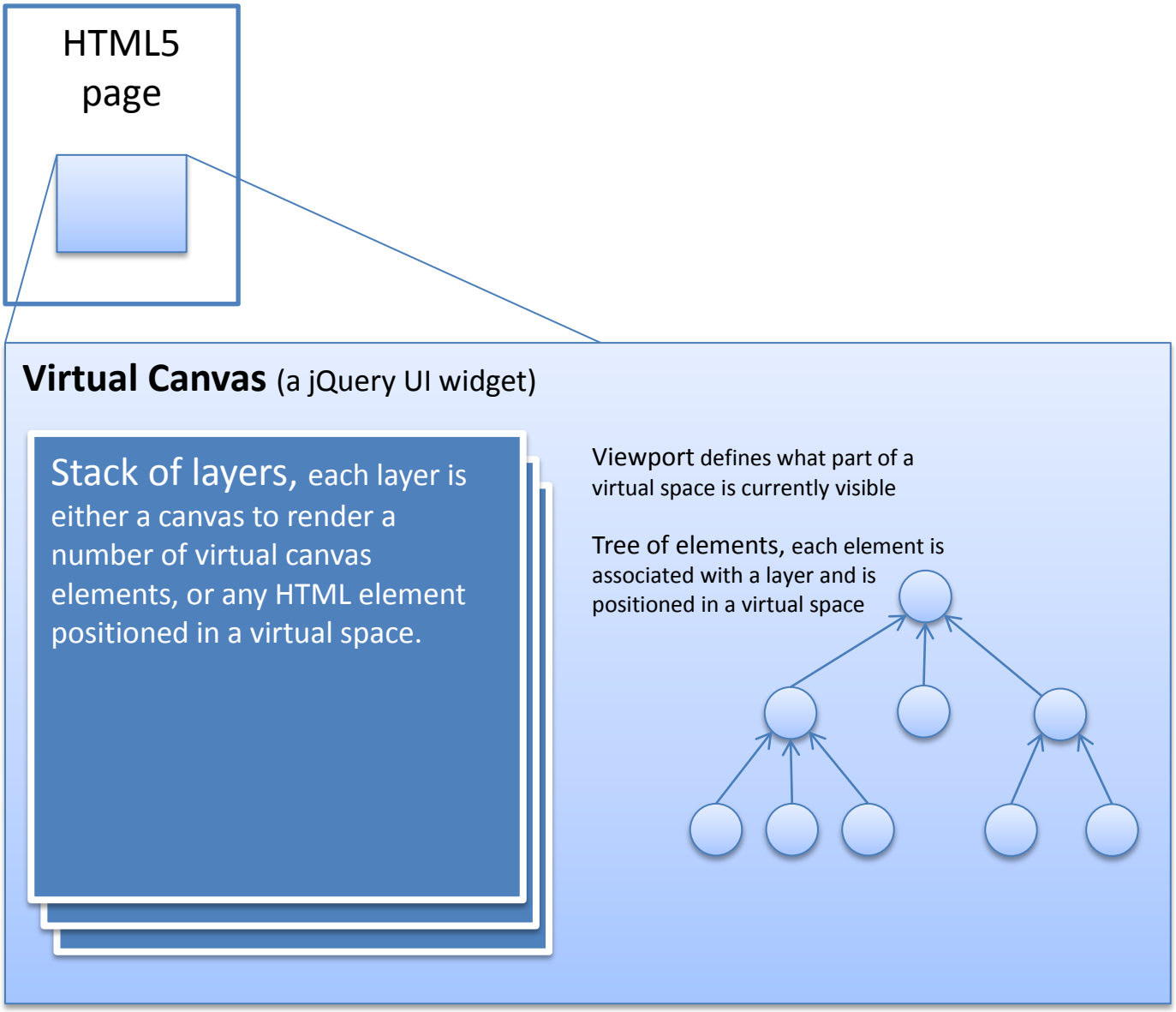
$$\mathbb{V} = (\gamma, \mathcal{V}, C\ ),$$

where $C = \{\mathcal{C}_i^p\}_{i=0}^{n-1}$ is a set of virtual canvas objects.

Implementation

Virtual canvas is implemented in JavaScript as a JQuery UI widget named *virtualCanvas* (defined in virtualCanvas.js). This plug-in makes selected element into a virtual canvas. All children elements become layers of the virtual canvas. A <canvas> element is added to each layer to render objects associated with the layer. Layers are arranged one above another and are transparent unless an opaque object is placed on it. All layers share same virtual space and visible region. All virtualCanvas elements get corresponding CSS classes when initializing the widget.

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License



The next HTML code block is an example of how looks before (i.e. in a source page) and after it becomes a virtual canvas (i.e. a snapshot of a dynamic page state):

Before:

```
<div id="vc" style="width: 100%; …">
    <div id="layerTimelines" onselectstart="return false;">
    </div>
    <div id="layerInfodots" onselectstart="return false;">
    </div>
</div>
```

After:

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

```
<div class="virtualCanvas" id="vc" style='width: 100%; height: 837px; …'>
 <div class="virtualCanvasLayerDiv" id="layerTimelines" style="width: 1904px; height: 837px;
          z-index: 0;" onselectstart="return false;">
      <canvas width="1904" height="837" class="virtualCanvasLayerCanvas" style="z-index: 1;"/>
 </div>
 <div class="virtualCanvasLayerDiv" id="layerInfodots" style="width: 1904px; height: 837px;
          z-index: 3;" onselectstart="return false;">
      <canvas width="1904" height="837" class="virtualCanvasLayerCanvas" style="z-index: 4;"/>
 </div>
</div>
```

Viewport

Virtual canvas API allows changing the visible region and getting the viewport, represented as `Viewport2d` type (defined in viewport.js). The type exposes methods for mapping between virtual canvas and screen coordinate systems.

Content objects

Virtual canvas content objects can be added through the API defined in vccontent.js. Base type for all content objects is `CanvasElement` which exposes fields for placement of the object in a virtual canvas space, associated layer and list of children. Thus content objects are represented as a tree, so that parent's bounding box includes children's bounding boxes (which in turn may intersect).

Content objects are defined in virtual canvas coordinates, and rendered on a canvas using current viewport to convert coordinates. The hierarchical structure allows performing fast rendering of a large content tree using the statement that if a parent object is invisible then its children are invisible, too.

Virtual canvas handles mouse events from the element and transforms them into following events for individual content objects: mouse enter, mouse move, mouse leave, mouse click and mouse hover.

There are many types of content objects implemented in vccontent.js:

- Generic types:
  - Generic element with support of dynamic level details (`CanvasDynamicLOD`);
  - Generic element hosting any HTML element (`CanvasDomItem`).
- Primitive elements:
  - Image (`CanvasImage`);
  - Rectangle (`CanvasRectangle`);
  - Circle (`CanvasCircle`);
  - Single line or multiline text on <canvas> (`CanvasText`);
  - Multiline text with scrolling on (`CanvasScrollTextItem`);
  - Multiresolution image (`CanvasLODImage`);
  - Video/audio (`CanvasVideoItem`, `CanvasAudioItem`);
  - Pdf (`CanvasPdfItem`);
  - PhotoSynth (`CanvasVideoItem`).
- Composite elements:

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

- Container of other objects without own visual representation (`ContainerElement`);
- Content item (`ContentItem`);
- Infodot (`CanvasInfodot`);
- Timeline (`CanvasTimeline`).

An infodot is an object with support for dynamic level of details. When zoomed out, it contains `CanvasImage` element with thumbnail image of different resolution depending on its actual screen size. As a user zooms in, a `ContainerElement` with a number of `ContentItems` replace the thumbnail.

Loading content

Current version of code fetches entire set of timelines, infodots and content items using Get request of the ChronoZoom WCF service when page is loaded. These objects only describe the corresponding items but don't contain any media content.

Time coordinates of the objects are transformed to "years from present" (which is a float number) and in this way are considered as x-coordinates of the objects in the virtual canvas space. Y-coordinates are assigned using the layout algorithm which recursively arranges timelines and infodots vertically. Finally, the objects are added to the virtual canvas.

Search

The search is intended to find and locate any element (timeline, infodot or content item) on a virtual canvas by a keyword. It is implemented in search.js.

A user is prompted to enter a keyword through a user interface. The keyup event is used to catch user's input; the actual request is performed when user stops typing (i.e. there is no keyup event anymore) for more than 300 ms. This avoids unnecessary frequent search queries and queries for just one or two characters when a user starts typing a keyword that can result in a very large number of found elements.

The search request is performed as an AJAX query to the Search operation of the ChronoZoom WCF service. Since the request is asynchronous, the latest request that comes after sending of the current request is performed when the current operation completes.

Search results are visually grouped into three categories: timelines, exhibits and content items. Each result contains type and unique ID of a corresponding element. If user clicks on a result item, it is searched in the virtual canvas object tree and, if found, the canvas is zoomed to that element. Since content items are loaded into object tree dynamically as an infodot is zoomed enough, it is possible that when a user tries to locate the search result, it is not loaded into the object tree. In this case it will be found in a description of a corresponding infodot, initially fetched from the service, and its placement within an infodot is computed.

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

Bookmarks

Bookmark is a string in a specific format which identifies a certain region in a virtual canvas space. Since actual layout of elements in a virtual canvas varies as new data is altered, bookmarks are defined relatively to a particular element.

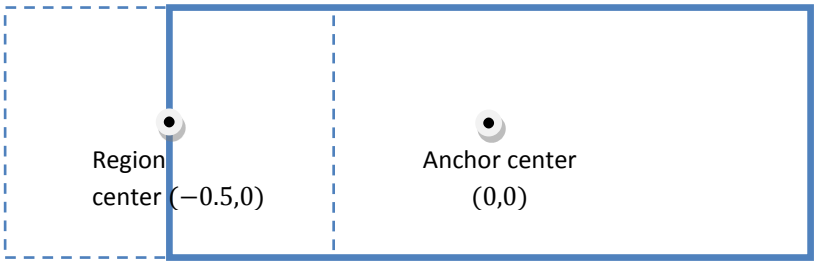
The bookmark format is:

```
[/path/to/an/element]{@x=[relative-x-of-visible-center]&y=[...]&w=[relative-width-of-visible]&h=[...]}
```

where `/path/to/an/element` contains slash-separated unique IDs of elements starting from the root element (e.g. Cosmos timeline), going through a chain of children elements and ending with an anchor element, relatively to which the bookmark is defined; `{…}` is an optional part, `x, y` are coordinates of the region relative to the anchor element, `w, h` are sizes of the region. If the part is missed, x and y are zero, and w and h are 1.

The values are relative to the element sizes, so (x = 0, y = 0) means center of the element, (0.5, 0.5) is its bottom-right corner, (-0.5,-0.5) is its top-left corner.

For example, `/t55/t174@x=-0.5&y=0&w=0.5&h=1` identifies the Earth timeline (`/t55/t174`, child of the Cosmos timeline), so that center of the visible region is on the left boundary of the timeline (*x=-0.5*) horizontally and centered vertically (*y=0*), width of the visible region must be **at least** half of timeline's width (*w=0.5*), height is **at least** equal to the timeline's height.



Bookmarks are used to save a visible region and share it with others or use it in a tour. Bookrmarks are automatically built by the ChronoZoom page and added to the address bar through '#', reflecting current visible region. Since different browser instances can have different aspect ratios, a bookmark's region is shown in a way when it is guaranteed that the region is completely on the screen (i.e. actual visible is not less than define in the bookmark).

In the ChronoZoom page, a bookmark is refreshed in the address bar when an animation is completed, and it is computed as relative to the last clicked or zoomed by a navigation link element (including regime links, search results and thresholds). For example, if user clicks a content item and then pan/zoom the canvas, the bookmark is relative to that content item element.

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

Gestures

We will work in terms of 2 visible rectangles, which display some region of entire data and consider common gestures of how to move from one of them to another. There are 2 common operations which will be used below to transform one visible rectangle to another: zoom and pan. Figure 1 shows these operations.

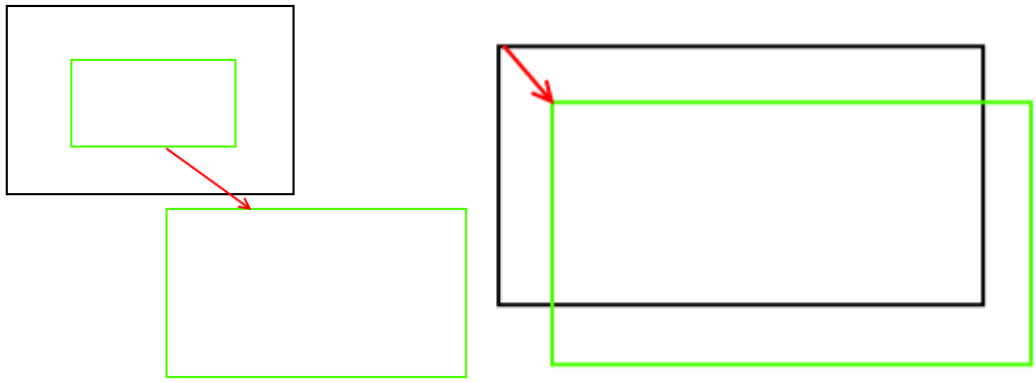


Figure 1. Zoom and Pan operations (black – old rect, green – new rect)

We will operate in screen coordinates, considering sizes in screen pixels

Zoom Operation

Parameters: X0, Y0 - zoom center point (stationary point of operation), Scale - zoom factor

Zoom({X,Y}, { X0, Y0, Scale }) -> {X0 + (X - X0) * S, Y0 + (Y-Y0) * S}

Pan Operation

Parameters: DeltaX, DeltaY - offset for horizontal and vertical directions

Pan({X,Y}, {DeltaX, DeltaY}) -> {X + DeltaX, Y - DeltaY}

Mouse Gestures

- **Mouse pan**. When user points on specific position and holds left button, then moves mouse, point under mouse cursor remains under it until left button will be released
- **Mouse zoom**. When user uses mouse wheel, visible region is zoomed with point under mouse cursor as zoom center
- **Mouse left button double click**. When user double clicks with left mouse button on a point, zoom operation is performed with point under mouse cursor as a center point.

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

- **Rectangle Selection***. If user presses “ctrl” button and selects a rectangle with mouse and left mouse button being pressed, then, after release of the left mouse button, selected rectangle will become new visible rect.

Keyboard Gestures

- **Keyboard arrows**. Using keyboard arrows user can pan visible rect to each of the 4 directions
- **+/- Zoom.** User can perform zoom operations by pressing “+” and “-“ buttons on the keyboard. Center of current visible rect will be used as zoom center.

Multitouch Gestures

- **Pinch and Spread**. Zoom operations can be performed with pinch and spread gestures. Zoom center point will be a middle point between two fingers.

- **Drag and Flick.** Pan operations can be performed with drag and flick operations. Points between fingers will remain under them while panning

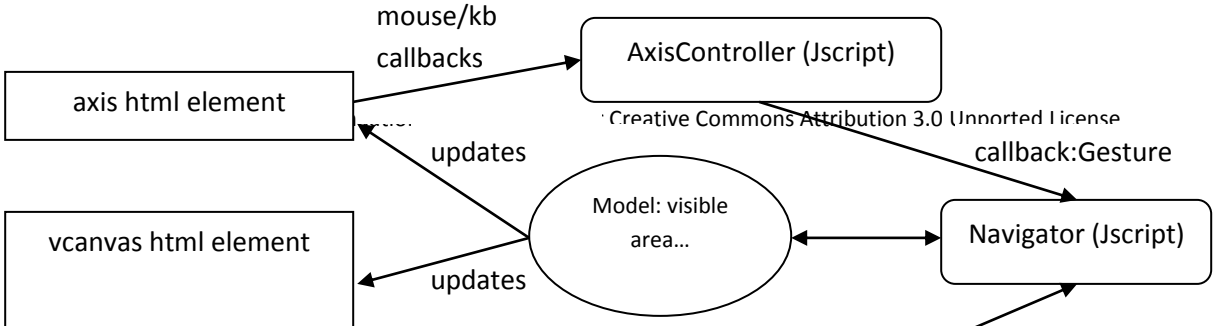
Math

We build the function

$$f: (g, \mathcal{V}) \to \mathcal{V}$$

Gesture g	Result
Pan: δ^p – the offset vector	$\delta^v = screenToVirtual(\delta^p, \mathcal{V}),$ $\mathcal{V}(c^v + \delta^v, w^p, h^p, \alpha)$
Zoom: p_0^p - zoom origin point, scale – value to zoom	$p_0^v = screenToVirtual(p_0^p, \mathcal{V}),$ $c_{new}^v = p_0^v + (c^v - p_0^v) * scale$ $\alpha_{new} = \alpha \times scale$ $\mathcal{V}(c_{new}^v, w^p, h^p, \alpha_{new})$

Implementation



Creative Commons Attribution 3.0 Unported License

Implementation pseudo-code:

HTML:

```
<div id=”axis”></div>
<div id=”vcanvas”></div>
```

Javascript:

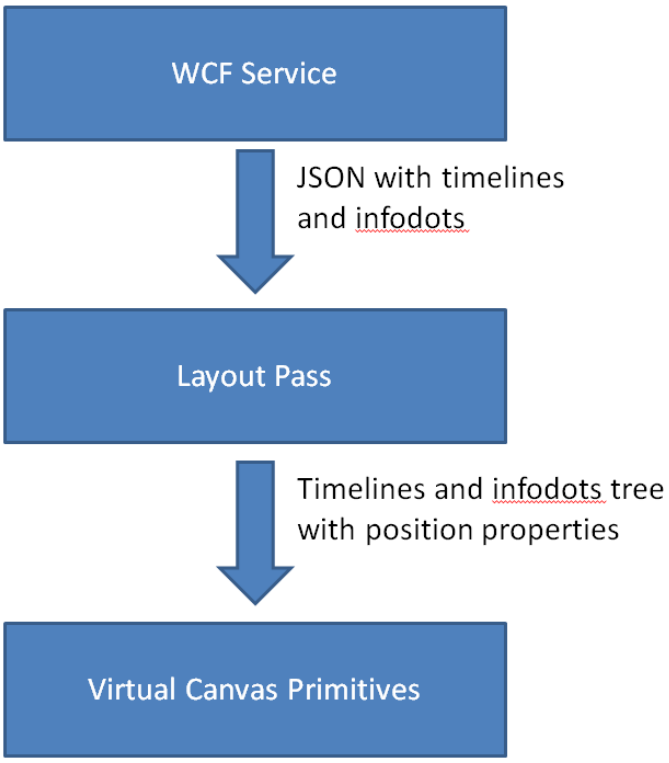
```
var axisController = new AxisController(“axis”); // Subscribes to mouse events inside
var canvasController = new CanvasController(“vcanvas”); // Subscribes inside
var navigator = new CZNavigator(model); // Navigator interacts with model
axisController.addSubscriber(navigator);
navigationController.addSubscriber(navigator);
```


© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

Layout Algorithm

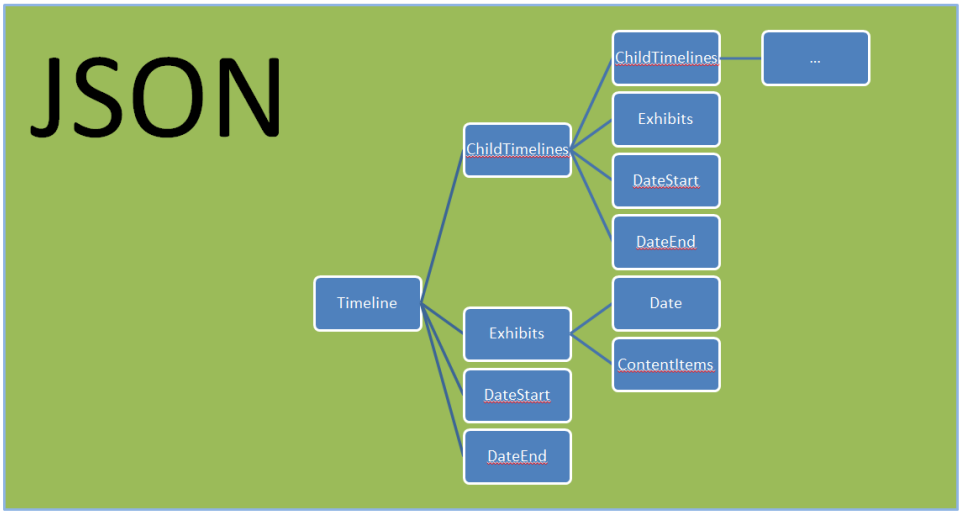
ChronoZoom layout system performs all actions to convert data from WCF service response to Virtual Canvas primitives. It performs parsing if input JSON to tree-structure of timelines and exhibits, converts dates of each element from Date Time to double Virtual Canvas coordinate and performs vertical positioning of each element, calculationg height for each timeline and exhibit. After that, it applies color which corresponds to timeline regime of each particular timeline and filles virtual canvas with all calculated and positioned primitives.

High-level architecture:



WCF service response sends the following data:

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License



Layoutpass does the following with the data received from the service

Axis Introduction

Time axis is a control that allows mapping of virtual coordinates (years from present day) to dates.

Axis has a set of options that describe its state at current moment. These options include range, mode and present day. Along with physical size they completely define and specify the appearance of axis.

Axis range is a time region currently visible on a screen. It is defined by two decimal values: right and left time bounds. Left bound should be less then right and both of them should be less than 0. These values are calculated as years from present time and passed to axis by setRange method. Default range is [13.7Ga, 0] and these values are extreme bounds, so axis won't show anything before 13.7Ga or in possible future.

Axis can be in one of three modes: 'cosmos', 'calendar', 'date'.

- 'Cosmos' mode corresponds to billions, millions or thousands of years ago. In this mode range bounds are calculated not from present day, but from 0.
- 'Calendar' mode corresponds to CE or BCE years. From here range bounds are defined as years from present day.
- 'Date' mode corresponds to CE or BCE years when zoom is deep enough to render months or days.

Axis modes switch automatically when axis visible range is updated and the choice depends only on order of range bounds.

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

Ticks layout algorithms.

Every time the user changes visible area by panning or zooming axis gets new range.

It invokes setRange method that causes recalculating ticks and labels values and placement coordinates.

The number of currently rendered ticks is specified by two values: beta and delta. Beta is the order of range length and delta is number of ticks that can divide region of current order. At the beginning of algorithm they have default values:

$$delta = 1,$$
$$beta = \log_{10}(right - left).$$

Then the main step of algorithm begins.

For 'cosmos' and 'calendar' mode at first we find span between two neighboring ticks and count of ticks:

$$span = delta * \ 10^{beta},$$
$$count = max - min + 1, \qquad min = \frac{left}{span}, \ \ max = \frac{right}{span}.$$

Then the coordinate is calculated for each tick:

$$x_i = min * span + i * span.$$

In 'date' mode calculating ticks is a little bit different.

When axis gets new range we find coordinate of the beginning of 1CE year. This value shows how much time has passed from present day till 1CE. Then we find start and end dates of range. Ticks are placed every month or day according to zoom level defined by beta and delta values.

These calculations use two date conversion methods. . First of them calculates number of years (as decimal value) that passes between two particular dates. The other calculates one date by given second date and number of years that have passed. We need to notice that beta version of ChronoZoom doesn’t support leap years, so every year includes 365 days.

After all the ticks values and coordinates are calculated, labels (text boxes over each tick) arrangement should be checked to avoid their possible overlay. If labels are too close to each other than we need to decrease number of ticks. It is made by changing beta and delta:

$$delta: \ \ 1 \to 2, 2 \to 5, 5 \to 1;$$
$$beta: \ if \ delta: 5 \to 1 \ then \ beta = beta + 1;$$

If labels are too distant from each other we need to increase number of ticks:

$$delta: \ \ 5 \to 2, 2 \to 1, 1 \to 5;$$

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

$$beta:\ if\ delta: 1 \rightarrow 5\ then\ beta = beta - 1;$$

Ticks coordinates and labels are recalculated for new beta and delta values. This loop of creating ticks works until the satisfactory result of arrangement checking.

After ticks were created we can fill space between every two neighboring of them with minor ticks. Minor ticks are arranged so that there are more than 10 pixels between every two of them.

Year-zero problem

In the Gregorian calendar there is no year zero and the year 1 BCE is followed by CE 1. However, a year zero is used in astronomical year numbering and there it is coincided with the year 1 BCE. Our solution to this problem was to consider 1BCE as year zero and while ticks rendering shift labels of BCE years.

Axis behavior

Time axis is an interactive control that supports instant reaction on every user action. The algorithm that recalculates ticks the same way every time the range changes allows smooth panning or zooming. Besides that it allows smooth transition from billions of years ago to CE or BCE years and even to months and days of each year.

Thresholds

Thresholds are interactive controls that provide navigation to particular dates or content items and react on mouse movements.

Information for thresholds is getting from database. It contains title, description, time coordinate and link to content item that is associated with this particular threshold. This information is getting once as application is launched and stored locally.

Each threshold can have three states: collapsed, partially expanded and completely expanded. The state of one particular threshold at particular moment depends on mouse navigation over time axis:

- When mouse is not over axis all thresholds are collapsed. In this case they are rendered as small rectangles over main axis line.
- When mouse is over axis than all thresholds partially expand to show their titles.
- If the user clicks on one of partially expanded thresholds than it fully expands to show all the content and other thresholds collapse.

Technically each threshold in expanded (partially or completely) mode is a ‘div’ element that is located on axis and is possessed according to its time coordinate and some simple rules. Thresholds shouldn’t overlay each other, they shouldn’t expand over the edge of the visible region and there should be at list one pixel between any two of them. These ‘div’ elements are stored temporally as thresholds are expanded and are deleted when thresholds collapse.

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

Thresholds expand and collapse with animation. This animation is provided by jQuery as a sequence of callback methods.

Tours

Tours components are defined in the Scripts/tours.js

Tour related components

Tours playing system consist of a set of UI components and a logical tour control object.

The tour related UI components are:

- Tour control buttons (upper menu)
- Bookmark description panel (lower-left corner)
- Bookmark description text (inside bookmark description panel)
- Tour selection menu (left panel)

Each of these UI components has a 2 state: shown and collapsed. The state of the UI components are changed by a logical tour control object during the tour playing.

Logical tour control object is a javasript **Tour** object, defined in a scripts/tours.js

Logical tour control object has a groups of a states. They are

- Audio enabled state. Can be On or Off. Must be set by the **`toggleAudio`** function after the tour object is created and before it is started to play. After the tour is started the audio enabled state must not be changed
- Play/Pause state. Tour can be in either Play or Pause state. The transition between these states can be initiated by the play/pause button of the “tour control buttons (upper menu)” being pressed, or by the user interaction with a canvas. E.g. if the user zoom the canvas whilst the tour is in a play state, the tour state is changed to the paused.

When the tour object is constructed it accepts a title, an array of a **`TourBookmark`**

objects and a URL of the audio blob with a narration for the tour.

An audio narration

For each of the tours an HTML5 audio element is allocated. When the tour is created it is supplied with a audioBlobUrl attribute which must be set to one of the audio BLOB urls where the audio narration track is located. Using this URL a HTML5 audio element is populated with a data sources to play. Along with audioBlobUrl a urls with different file extensions but with the same URL schema and file location are added as a sources to the HTML5 audio element. **`toursAudioFormats`** array defined in a cz.settings.js indicates which audio file extension must be tried to fetch.

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

A preferred audio file begins to download for the each of the the tours in a background after the web site is loaded. This is done to eliminate a delay for audio track buffering when the user starts the tour.

Now an mp3 is used for the IE, Chrome, Safari browsers and a low quality WAV is used for the FireFox.

Tour Bookmarks

Each TourBookmark object has a url that depicts the area of the canvas that must be shown for this bookmark, a textural caption and description and a lapse time parameter. The lapse time parameter is a time that is used for navigation across the audio narration for the tour. It indicates the time period from the beginning of the tour at which the corresponding audio track region is placed. The duration of the n-th bookmark is calculated from the lapse time of the (n+1)-th bookmark. A duration of the last bookmark of the tour is calculated from the audio track metadata.

Tour bookmarks transitions

Tour bookmarks transitions are caused by the 2 types of events:

- An animation of zooming to the next bookmark is ended
- A duration of the bookmark show is ended

First, when the user starts the tour en elliptical zoom is performed to the first bookmark. An audio and a bookmark showing timer is not started. When the animation complete event is fired the audio playing is activated (if it is enabled) and the bookmark viewing timer is set.

When the bookmark viewing timer fires, the audio track playback position is set to the next bookmark lapse time and an elliptical zoom animation to the next bookmark is started and a duration of the bookmark timer is set. Thus the bookmark describing audio starts to play during the transition animation to the bookmark and a transition to the bookmark time is included to the bookmark showing duration.

Pressing of a prev or next button by the user causes the same effect as a firing of the bookmark showing duration timer.

Tour control API

Tours can be control through the set of the global functions. They are:

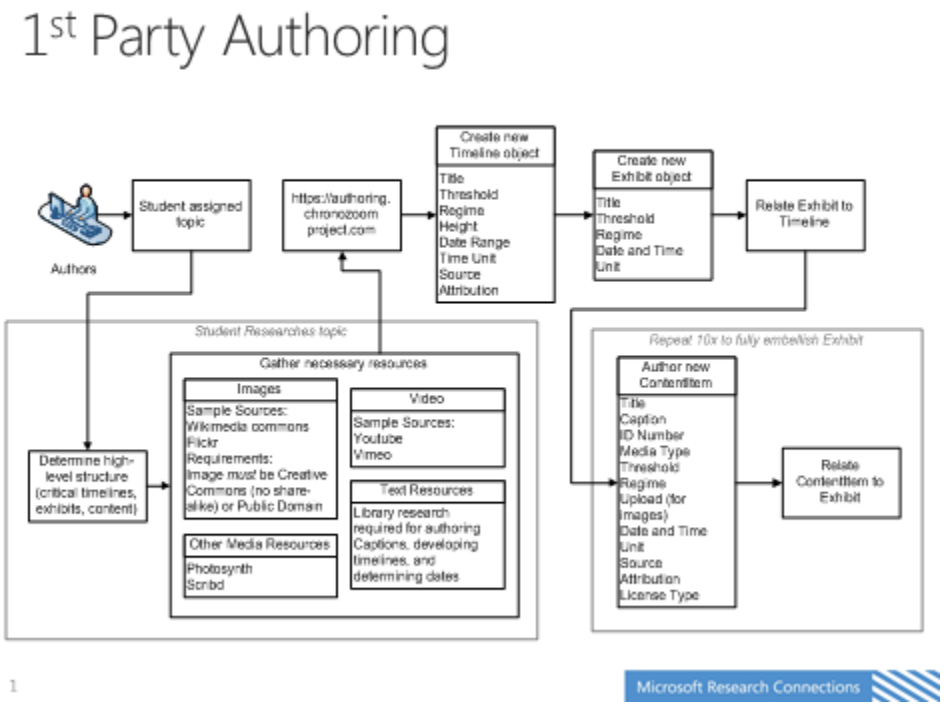
- activateTour – starts the new tour. Brings up a tour control UI components
- tourAbort – stops the ongoing tour. Hides all tour control UI components
- tourNext – move to the next bookmark
- tourPrev – move the the previous bookmark
- tourPause – pause the playing tour
- tourResume – continues the paused tour

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

1st Party Authoring

These two diagrams show our first party authoring system. This is the first generation tool that allows us to interact with the ChronoZoom database. The first diagram shows the process of adding a single Artifact to the canvas. We plan to simplify this process by creating a new user interface that makes authoring visual and intuitive. Once completed, this system will be known as our third party authoring system, allowing content partners to add new Artifacts and Exhibits themselves. The second diagram shows the process of creating a tour. This is also a first party system, requiring precise input and timing for tour stops."



The Future of ChronoZoom

We envision a world where scientists, researchers, students, and teachers collaborate through ChronoZoom to share information via data, tours, and insight.

Imagine a world where the leading academics publish their findings to the world in a manner that can easily be accessed and compared to other data.

Imagine a tool that allows teachers to generate tours specific to their classroom needs.

This can happen with your support. As ChronoZoom through the beta release, we need your feedback and support to continue to mold this project to suit your needs.

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License

Help ChronoZoom evolve by taking this survey so we can provide the best possible future features: http://www.zoomerang.com/Survey/WEB22EFZBLQL4B/

© 2012 The Outercurve Foundation. Distributed under Creative Commons Attribution 3.0 Unported License