Supplementary Material

This document is the supplementary material for the paper *Data Player: Automatic Generation of Data Videos with Narration-Animation Interplay*. It provides details about constraint encoding and the Z3 CSP solver to generate animations.

1 Z3 CSP Solver

To generate the animation sequence, Data Player assigns concrete values to specific variables and leverages the CSP solver to explore numerous combination alternatives in the large search space. Specifically, we encode high-level design knowledge summarized from the formative study and existing literature as computational low-level constraints. All constraints are formalized as equations and fed into the Z3 CSP solver. The solver outputs suitable animations. The following section will explain the low-level constraint encoding in detail.

1.1 Hard and soft constraints

In the Z3 CSP solver, `hard` constraints form the absolute conditions any solution must meet, while `soft` constraints are ideal but not mandatory. A solution remains valid even when it violates a soft constraint. During our design process, we realized that some constraints needed to be soft, as we may violate them to fulfil hard constraints. However, using the standard soft constraints feature of the CSP solver led to issues as it uniformly treats all soft constraints, and restricted our ability to selectively relax certain constraints.

$$\phi_{\texttt{hard}}(\texttt{echo}) \overset{\texttt{hard}}{=} \texttt{must satisfy} \tag{1.1}$$

$$\phi_{\texttt{soft}}(\texttt{echo}) \overset{\texttt{soft}}{=} \texttt{not mandatory to satisfy} \tag{1.2}$$

To overcome this, we implemented our own version of soft constraints. We created additional Boolean variables p for each soft constraint and added an implication `Implies(p, constraint)` to the solver. If a particular model fails to satisfy the entire problem (i.e., the problem is unset), we can potentially use the function `unsat-core` to determine which of the soft constraints caused the problem. Data player will then discard these problematic constraints and reattempt the solution. Meanwhile, this design feature allows the frontend to display the unsatisfied constraints to the user and lets them decide which to omit.

2 Basic visual Design Quality

2.1 Linking

First, to ensure basic visual design quality, we use the established text-visual links to generate visual structure and data facts constraints, matching textual and visual entities and grouping semantic-related visual elements. For each linked `echo`, we get the variables `echo.time`, `echo.elements` from the text-visual linking. To represent a potential animation at the echoed time, we create a new `animation` instance for the echoed `element` and encodes the `linking` constraint to set the start time of this potential animation be the `echo.time`. `linking` constraint is the only constraint that creates new `animation` instances for the elements, thus it exclusively allows the visual elements linked to specific narration segments to be animated.

$$\phi_{\texttt{linking}}(\texttt{echo}) \overset{\texttt{hard}}{=} \bigwedge_{e \in echo.elements} \text{e.animation.start\_time} = \text{echo.time} \tag{1.3}$$

2.2 Integrity

For the `intergrity` constraints, Data Player defines two intermediate variables: `animation.activate` and `element.num_activate`. `activate` determines if the potential animation will be used or not, and `num_activate` is the number of activated animations for the element. Together, the `num_activate` constraint tracks the number of activated animations for each element.

$$\phi_{\texttt{num\_activate}}(E, A_e) \overset{\texttt{hard}}{=} \bigwedge_{e \in E} \text{e.num\_activate} = \sum_{a \in A_e} \begin{cases} 1 & \text{if a.activate = true} \\ 0 & \text{if a.activate = false} \end{cases} \tag{1.4}$$

Then, Data Player encodes `integrity` constraints to ensure that all elements involved in text-visual links are animated, i.e. for each element, at least one animation is activated. For the annotation elements, the `integrity` constraint is the hard constraint, while for other types of elements, Data Player encodes the `integrity` constraint as "soft", which means if it violates other hard constraints such as time conflict, this element will not be animated. Without any animation, the element will be always on the canvas.

$$\phi_{\texttt{force\_use}}(E, A_e) \begin{cases} \overset{\texttt{hard}}{=} \bigwedge_{e \in E} \text{e.num\_activate} \geq 1 & \text{if e.type == "annotation"} \\ \overset{\texttt{soft}}{=} \bigwedge_{e \in E} \text{e.num\_activate} \geq 1 & \text{otherwise} \end{cases} \tag{1.5}$$

2.3 Group

We design `group` constraints to group the visual elements that are related to data facts in the text-visual links. Meanwhile, we design the `association` constraints to ensure that if one element is linked to narration, itself and other elements in the same data group can be animated. In addition, our `consistency` constraints specify that elements from different groups that are visually consistent should apply the same animation.

3 Temporal Constraints

Second, we encode sets of temporal constraints to time-align animations and narration. Each group of constraints specifies how different elements of the data video should be timed and arranged on the timeline according to their type, effect, and relation to the narration. Narration text inherently contains a chronological relationship between words. We automatically generate audio narration with Microsoft Azure Text-to-Speech services and obtain the timestamps of each word in the audio, which also acts as the timeline to arrange animation effects applied to the visual elements.

3.1 Duration

Data Player encodes constraints to determine that the animation effects are triggered by the onset of the first word in each linked narration segment, and last for the duration of the corresponding text span in the audio in default. `default_duration` constraint calculates this default duration with the `first_word.time` and `last_word.time` from the linked narration.

$$\phi_{\texttt{default\_duration}}(E, A_e) \overset{\texttt{hard}}{=} \bigwedge_{a \in A_e} \text{a.default\_duration} = \texttt{last\_word.time} - \texttt{first\_word.time} \tag{1.6}$$

Meanwhile, for the same element, the animations should not have a time conflict. Therefore, the duration is also constrained by the next activated animation's start time. We define a dictionary of variables D_{next} to calculate and store the start time of the next animation. The `next_start` constraint chains the start time of the next animation along the timeline.

$$\phi_{\texttt{next\_start}}(A_e) \overset{\texttt{hard}}{=} \bigwedge_{pre_a, cur_a \in A_e} \begin{cases} D_{next}[pre_a.time] == pre_a.time & \text{if pre\_a.activate == true} \\ D_{next}[pre_a.time] == D_{next}[cur_a.time] & \text{otherwise} \end{cases} \tag{1.7}$$

With the time for the upcoming animation from D_{next}, we can now incorporate the `duration` constraint to control the duration of each animation. The exception here is the emphasis animations, which can be concurrently activated with other animations - for instance, zooming in as a bar expands.

$$\phi_{\texttt{duration}}(A_e) \overset{\texttt{hard}}{=} \bigwedge_{a \in A_e} \begin{cases} \text{a.duration} \leq \min(D_{next}[\text{a.start\_time}] - \text{a.start\_time}, \text{default\_duration}) & \text{if a.type != “emphasis”} \\ \text{a.duration} \leq \text{default\_duration} & \text{otherwise} \end{cases} \tag{1.8}$$

3.2 Conflict

Another important set of temporal constraints enforces the inherent logical order of animation actions. Firstly, for each element, we track and store the `type` variable to determine the type/effect of animation along the timeline. This on-screen status enables `conflict` and `overlap` constraints explained in the following sections.

Specifically, we specify three animation actions: “enter” animations are applied when an object appears on the screen, “exit” animations are applied when an object disappears from the canvas, and “emphasis” animations are applied to draw attention to an object that is already on the canvas. The three status is encoded in Z3 in the form of two variables ”enter” and ”exit”.

$$a.type = \begin{cases} \text{ENTER (not on canvas}\rightarrow\text{on canvas)} & \text{enter=true, exit=false} \\ \text{EXIT (on canvas}\rightarrow\text{not on canvas)} & \text{enter=false, exit=true} \\ \text{EMPHASIS1 (not on canvas}\rightarrow\text{not on canvas)} & \text{enter=false, exit=false} \\ \text{EMPHASIS2 (on canvas}\rightarrow\text{on canvas)} & \text{enter=true, exit=true} \end{cases} \tag{1.9}$$

Note that there are two types of emphasis. The first type of emphasis applies to the elements that exist on the canvas, such as bumping the text and highlighting the bar. The second type applies to the elements not on the canvas before the animation. One common usage is showing an annotation circles the number when it is mentioned. The `type` variable helps us to determine the conflict due to the animation actions along the timeline. For example, visual elements can only be emphasized or disappeared after they appear, and elements cannot be emphasized after they disappear.

With the `type` variables, we encode `on_screen` variables and constraints to determine when an element appears or disappears. If `on_screen` is true at time t, then the corresponding element is visible at that time. Otherwise, it is hidden. With the `conflict` constraint, we create a table $T(t) = on\_screen$ that calculates which elements are on the canvas at any given moment. The `conflict` constraint consists of 4 conditions to follow:

- If the ENTER or EMPHASIS2 state is applied to the element at time $t \in$ T(t), then the element will be on the canvas until the next animation is applied.
- If the EXIT or EMPHASIS1 state is applied to the element at time $t \in$ T(t), then the element will be removed from the canvas until the next animation is applied.
- For any time $t \in$ T(t), if one element is on the canvas, the ENTER and EMPHASIS1 state cannot be applied.
- For any time $t \in$ T(t), if one element is not on the canvas, the EXIT and EMPHASIS2 state cannot be applied.

These four conditions are applied to all the animations of the element. We achieve this by iterating the animations in the order of time in pairs. The pseudo-code in Algorithm 1 below implements the constraints, where the table T(t) controls the animation actions to avoid conflicting movements. It also helps us to avoid overlapping between elements.

3.3 Other temperoal constraints

Temporal constraint `earliest_time` calculates the earliest time each element can choose.

$$\phi_{\texttt{earliest\_time}}(E, A_e) \overset{\texttt{hard}}{=} \bigwedge_{e \in E} \text{e.earliest\_time} = \min_{a \in A_e \text{ and a.activate is true}} (\text{a.start\_time}) \tag{1.10}$$

Algorithm 1 OnScreenConstraints

procedure OnScreenConstraints
  ▷ At time 0, the element is not on screen.
  z3.addConstraint($\neg T[0]$)
  ▷ If animation is activated at time 0, it enters the canvas.
  z3.addConstraint(Implies($T.animations[0].activate, T.animations[0].enter$))
  for $cur\_time, pre\_time, cur\_ani, pre\_ani$ in $T, T.animations$ **do**
    $condition1 \leftarrow$ z3.Implies($pre\_ani.activate$, Xor($pre\_ani.exit, T[cur\_time]$))
    $condition2 \leftarrow$ z3.Implies($\neg pre\_ani.activate, T[cur\_time] == T[pre\_time]$)
    ▷ If the last activate animation exits, the element is not on the screen, and vice versa.
    z3.addConstraint(And($condition1, condition2$))
    ▷ If the current element is on screen, then the animation cannot be entered, and vice versa.
    z3.addConstraint(Implies($cur\_ani.activate$, Xor($T[cur\_time], cur\_ani.enter$))
  end for
end procedure

Every element (except annotation) default to have one animation at the earliest mentioned time. The `defalut_animation` constraints will mark the specific animation to be activated if it is at the earliest time that the element is available.

$$\phi_{\texttt{default\_animation}}(E, A_e) \overset{\texttt{soft}}{=} \bigwedge_{e \in E, e.type \neq \text{"annotation"}} \forall a \in A_e, \text{a.start\_time} = \text{e.earliest\_time} \implies \text{a.activate} \tag{1.11}$$

4 Order

Data player also encodes a set of `order` constraints that define an optional logical sequence of elements.

4.1 Optimized Elements Order Constraints

If we directly encode the constraints pairwise into the z3, there will be $O(n^2)$ constraints, where n is the number of elements.

$$\phi_{\texttt{order}}(G_1 < G_2) \overset{\texttt{hard}}{=} \forall e_1 \in G_1, e_2 \in G_2, \text{e1.start\_time} \leq \text{e2.start\_time} \tag{1.12}$$

To optimize the time complexity, we encode the constraints as follows:

$$\phi_{\texttt{order}}(G_1 < G_2) \overset{\texttt{hard}}{=} \begin{cases} \forall e_1 \in G_1, \text{e1.start\_time} \leq \text{intermediate} \\ \forall e_2 \in G_2, \text{intermediate} \leq \text{e2.start\_time} \end{cases} \tag{1.13}$$

with the intermediate variable, the number of constraints is optimized to be linear.

4.2 Order by Element Type

With the `order` constraint, we can define constraints that force the order between any two groups of elements. For example, in a bar chart, the axes need to enter before the bar shows. Such requirements are formed into the following constraints: The `element_order` constraint forces the elements in different data groups to follow the order: background →title→axis →[data-item | annotation].

$$\phi_{\texttt{element\_order}}(E, A_e) \overset{\texttt{hard}}{=} \begin{cases} \phi_{\texttt{order}}(G_{background} < G_{title}) \\ \phi_{\texttt{order}}(G_{title} < G_{axis}) \\ \phi_{\texttt{order}}(G_{axis} < G_{data-item}) \\ \phi_{\texttt{order}}(G_{axis} < G_{annotation}) \end{cases} \tag{1.14}$$

4.3 Synchronization

To ensure that elements in the same data group activate together, we define `synchronization` constraints as a soft constraint.

$$\phi_{\texttt{synchronization}}(G) \overset{\texttt{soft}}{=} \forall e_1 \in G, \text{e1.start\_time} == \text{intermediate} \tag{1.15}$$

5 Overlap

Elements on the canvas may overlap. However, these overlapping elements do not necessarily lead to conflicts, provided they are not simultaneously visible on the screen. As mentioned in the section conflict, the `on_screen` table $T(t)$ can be used to track the overlapping conflict at any time along the timeline. We first define `is_overlap` constraint to check if all the elements in the group overlap.

$$\phi_{\texttt{is\_overlap}}(E) \overset{\texttt{hard}}{=} \bigwedge_{e_1,e_2 \in E} \begin{cases} e_1.\texttt{bbox.max\_x} < e_2.\texttt{bbox.min\_x} \\ e_1.\texttt{bbox.min\_x} > e_2.\texttt{bbox.max\_x} \\ e_1.\texttt{bbox.max\_y} < e_2.\texttt{bbox.min\_y} \\ e_1.\texttt{bbox.min\_y} > e_2.\texttt{bbox.max\_y} \end{cases} \tag{1.16}$$

Secondly, for each time t along a timeline, we check all the on-screen elements to ensure they don't overlap.

$$\phi_{\texttt{overlap}}(T) \overset{\texttt{hard}}{=} \bigwedge_{t \in T} (\phi_{\texttt{is\_overlap}}(|e, e.T[t] == true|) == false) \tag{1.17}$$

6 Animation effect

We utilize a small set of pre-designed animation effects based on the GSAP animation platform for different actions as a technology probe and proof-of-concept to explore our main research concern. The animations cover common chart types (e.g., pie chart, bar chart, line chart, and scatterplot), and can be divided into three behavior types: Enter, Exit, and Emphasis. Each animation preset incorporates controls of multiple visual elements. For instance, in a pie chart, the sector and its corresponding legend elements (e.g., symbol and label) are usually bound into a group. So we define a new animation called "pie-wheel-and-legend-fly-in", which means that the pie chart's sector will wheel clockwise and the legend-related elements will fly in at the same time. As a result, we can apply only one animation to multiple elements, avoiding specifying animations for each element individually.
Each animation contains multiple predefined sub-animations for a set of visual components. Define that animation is designed for the visual components $V := V_h \cup V_s$, where V_h is the element set of hard constraints and V_s is the element set of soft constraints (marked [ ]). V_c is the set of selected elements in the text-visual link. For each animation in the library, only if $V_s \subseteq V_c \subseteq V$, it can be applied.
On this basis, we define an objective function to minimize the number of animations used: $\min \sum_{i=1}^{m} A_i$, where A_i is the number of animations applied to the i-th text-visual link and m is the number of text-visual links. This function ensures that the module uses our predefined animation combinations as much as possible to maintain narrative coherence.